Chapter 8
Answers and Appendix: Unix Guide

8.1 Answers

Answer 1 You can either repeat `mkdir`

```
mkdir TestDir1
mkdir TestDir2
```

or write more succinctly

```
mkdir TestDir1 TestDir2
```

followed by

```
ls
```

Answer 2 The command

```
cd
```

changes to your "home directory". Try the command `pwd` to "print which directory" you are in.

Answer 3 The star, or "wildcard", matches any completion of `TestDir`. So

```
rmdir TestDir*
```

deletes `TestDir1` and `TestDir2`.

Answer 4 This causes the error message

```
rmdir: failed to remove TestDir1: Directory not empty
```

You can now either go into `TestDir1` and delete its contents or apply the "recursive" version of `rm`

```
rm -r TestDir1
```

Answer 5 Recall the wildcard character introduced in Problem 3:

```
rm *
```

The original version of the backmatter was revised: For detailed information please see Erratum. The erratum to this chapter is available at https://doi.org/10.1007/978-3-319-67395-0_9

B. Haubold and A. Börsch-Haubold, *Bioinformatics for Evolutionary Biologists*, https://doi.org/10.1007/978-3-319-67395-0_8

Answer 6 Rename file `a`

```
mv a b
```

The command `mv` stands for "move". By moving a file from one name to another, it is renamed. To get more feedback on the action of `mv`, you can use `-v` to switch on its verbose mode.

Answer 7 The command `history` lists all previous commands, with `history` as the last one.

Answer 8 The shell prints

```
>
```

and waits for further input until the single quote in "can't" is closed. Alternatively, the command can be terminated using `C-c` twice. This is a very useful command for getting out of trouble.

Answer 9 The command goes into auto-repeat mode.

Answer 10 The command `ls -t` lists the files modified most recently first.

Answer 11 The help page contains a section on searching explaining that

```
/pattern
```

generates a search, and the next match can be found by pressing `n`.

Answer 12 `Wc` returns three numbers, the number of lines, words, and bytes in the input. In this case, the number of lines is identical to the number of words and both are equal to the number of files in the current directory.

Answer 13 Look up the `bash` man page

```
man bash
```

and search for pipe

```
/pipe
```

which gets you to the right place.

Answer 14 The command

```
x=1+1; echo $x
```

performs no addition, but simply returns the string `1+1`.

Answer 15 The command

```
((x=1+1)); echo 'The result is $x'
```

prints the literal string rather than *interpolating* the variable `$x` as done with double quotes.

Answer 16 As with the double brackets, if `let` is dropped, `x` is assigned `1+1` as a string rather than as a calculation:

```
x=1+1; echo $x
1+1
```

Answer 17

```
let y=2**10; echo $y
1024
```

For mental arithmetic, it is useful to remember, $2^{10} \approx 10^3$.

Answer 18

```
let y=10-20; echo $y
-10
```

So, negative numbers work, too.

Answer 19

```
let y=10/3; echo $y
3
```

because the `bash` only computes with integers.

Answer 20

```
4^10
1048576
```

Alternatively, you could have estimated in your head

$$4^{10} = 2^{20} = \left(2^{10}\right)^2 \approx \left(10^3\right)^2 = 10^6.$$

Answer 21

```
echo 10/3 | bc
3
```

In other words, without `-l`, the default integer mode of `bc` is switched on.

Answer 22 List the contents of the root directory

```
ls /
```

Count the files and directories listed

```
ls / | wc -l
```

Answer 23 Change into `BiProblems`

```
cd BiProblems
```

List all files

```
ls -a
.  ..
```

This means the directory is empty. Nevertheless, there are two directories we can refer to. The first is denoted by one dot, which stands for the current directory, and the second by two dots, the parent directory in the tree.

Answer 24 Use

```
ls Data/ | wc -l
```

Answer 25 Count the number of FASTA files in `Data`:

```
ls Data/*.fasta | wc -l
```

Answer 26 Copy

```
cp ../Data/mgGenes.txt .
```

check

```
ls
```

and count the genes

```
wc -l mgGenes.txt
525 mgGenes.txt
```

Answer 27 The command `cat -n` adds numbers to the printed lines. For re-counting, we type

```
cat -n mgGenes.txt
```

and look at the last line of the output

```
525  MG_470  579224  580033  -
```

Answer 28 List the first ten lines:

```
head mgGenes.txt
MG_001  686    1828   +  dnaN
MG_002  1828   2760   +
MG_003  2845   4797   +  gyrB
MG_004  4812   7322   +  gyrA
MG_005  7294   8547   +  serS
MG_006  8551   9183   +  tmk
MG_007  9156   9920   +
MG_008  9923   11251  +
MG_009  11251  12039  +
MG_010  12068  12724  +
```

List the last ten lines:

```
tail mgGenes.txt
MG_462  566186  567640  -  gltX
MG_463  567627  568406  -
MG_464  568399  569556  -
MG_465  569528  569914  -  rnpA
MG_466  569883  570029  -  rpL34
MG_467  570055  570990  -
MG_468  570994  576345  -
MG_526  576351  577205  -
MG_469  577268  578581  -
MG_470  579224  580033  -
```

There are five columns: a key, start and end positions, the strand, and the gene symbol, if available.

Answer 29

```
grep + mgGenes.txt | wc -l
299
grep - mgGenes.txt | wc -l
227
```

The total number of genes (Problem 26) is 525, while $299 + 227 = 526$. One gene is counted twice, presumably because its name contains a "-".

Answer 30

```
cut -f 5 mgGenes.txt | grep -
rpmG-2
polC-2
```

Find the strand of *rpmG*

```
grep rpmG mgGenes.txt
MG_473  64367   64513   -   rpmG-2
MG_325  408793  408954  -   rpmG
```

and of *polC*

```
grep polC mgGenes.txt
MG_031  32359   36714   -   polC
MG_261  315701  318325  +   polC-2
```

So it was *polC-2* which resulted in the extra count for a gene on the minus strand.

Answer 31 Extract the genes

```
cut -f 1-4 mgGenes.txt | grep - > minus.txt
cut -f 1-4 mgGenes.txt | grep + > plus.txt
```

Check results

```
wc -l minus.txt
226 minus.txt

wc -l plus.txt
299 plus.txt
```

which add up to 525 genes. The simplest method to see that the genes on the plus strand do not form a homogeneous block is to enter

```
head -n 20 mgGenes.txt
```

which shows five genes on the minus strand. We can further quantify the mixing of genes between both strands: If the genes on the plus strand did form a homogeneous block, the first 299 genes in `mgGenes.txt` would all be on the plus strand. So we look at the first 299 genes in `mgGenes.txt` and count how many of them are on the plus strand:

```
head -n 299 mgGenes.txt | grep + | wc -l
246
```

Hence, there are only $299 - 246 = 53$ genes on the minus strand among the first 246 genes on the plus strand.

Answer 32 Leftward redirection (`<`) writes the contents of a file to the standard input stream, stdin, from where it can be read by any tool, for example, `grep`:

```
grep + < mgGenes.txt
```

Answer 33

```
man sudo
```

explains that `sudo` switches into sys-admin mode.

Answer 34 Without the ampersand, the command line freezes until the child window running `emacs` is closed again.

Answer 35 Saving to `mgGenes2.txt` can be done in `emacs` via the menu. Searching for `polC-2` can also be done using the menu. Now check the difference between `mgGenes.txt` and `mgGenes2.txt`:

```
diff mgGenes.txt mgGenes2.txt
56c56
< MG_473       64367   64513   -      rpmG-2
---
> MG_473       64367   64513   -      rpmG_2
288c288
< MG_261       315701  318325  +      polC-2
---
> MG_261       315701  318325  +      polC_2
```

This means line 56 of `mgGenes.txt` was changed (`c`) into line 56 of `mgGenes2.txt` and similarly for line 288.

Answer 36 Extract, for example, line 56 of `mgGenes.txt`:

```
head -n 56 mgGenes.txt | tail -n 1
```

Answer 37 The one exception is `C-w`, which deletes the word to the left of the cursor in `bash`, but deletes a selected region in `emacs`.

Answer 38 Exit `emacs` with

```
C-x C-c
```

Notice the file `mgGenes2.txt`, which is a backup file created by `emacs` when you worked on `mgGenes2.txt`. You can ignore it for now, but if in some later `emacs` session something disastrous happens to the original file, you can always go to the backup.

Answer 39 The command is not found:

```
drawGenes
drawGenes: command not found
```

Answer 40 On Ubuntu, we get

```
which ls
/bin/ls
```

Answer 41

```
find ~/ -name ".bashrc"
```

Answer 42 Draw the figure with

```
drawGenes exampleGenes.txt |
gnuplot -p -e 'unset ytics; plot[][-10:10] "< cat" title ""
    with lines'
```

If `gnuplot` is not installed on your system, you need to install it at this point. Notice also that we have distributed the commands of the pipeline over two lines to make them easier to read. You can enter them either all on the same line or on separate lines as shown.

Answer 43 The shortest version we found was

```
drawGenes exampleGenes.txt |
gnuplot -p -e 'uns yti; p[][-10:10] "< cat" t "" w l'
```

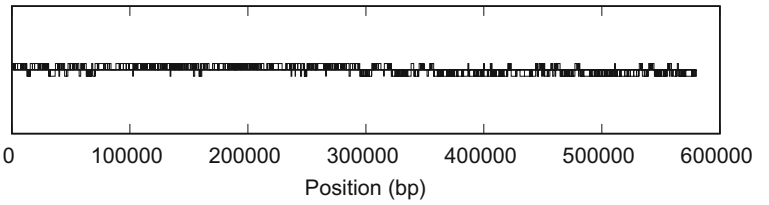
Answer 44 The command `set xlabel` can be abbreviated to `se xl`:

```
drawGenes exampleGenes.txt |
gnuplot -p -e 'se xl "Position (bp)"; uns yti; p[][-10:10]
   "< cat" t "" w l'
```

Answer 45 Draw the genes:

```
cut -f 2-4 mgGenes.txt |
drawGenes              |
gnuplot -p -e 'se xl "Position (bp)"; unset ytics; plot
    [][-10:10] "<cat " t "" w l'
```

to get



On the 5'-half of the genome, the genes are predominantly on the forward strand, on the 3'-half predominantly on the reverse strand.

Answer 46

```
./drawGenes.sh
bash: ./drawGenes.sh: Permission denied
```

Answer 47

```
drawGenes.sh
drawGenes.sh: No such file or directory
```

Answer 48 As before, we reset the environment

```
source ~/.bashrc
```

and should then be able to execute `drawGenes.sh`.

Answer 49 The preliminaries were

```
cd              # change into the home directory
cd BiProblems
mkdir UnixScripts
cd UnixScripts
```

As to “Hello World!”,

```
echo 'Hello   World!'
```

or

```
echo Hello'   'World!
```

Answer 50 When executing

```
for((i=1; i<=10; i++)); do echo -n 'Hello World!'; done
```

the newline usually added by `echo` is omitted. This is useful for printing more than one item on the same line.

Answer 51

```
for i in $(seq 10)
do
    echo $i
done
```

Answer 52

```
for i in $(seq 10)
do
    echo -n $i ' '
done
```

Note the blank in single quotes at the end of the `echo` command to separate the numbers. This might also have been surrounded by double quotes. However, recall from Problem 15 that interpolation of variable values only works inside double quotes:

```
for i in $(seq 10)
do
    echo -n "$i "
done
```

Answer 53 The command

```
man seq
```

tells us how to count in steps of two

```
seq 0 2 10
```

or backward

```
seq 10 -2 0
```

Answer 54

```
for i in $(seq 525)
do
    cut -f 4 mgGenes.txt |
          head -n $i     |
          grep +         |
          wc -l
done
```

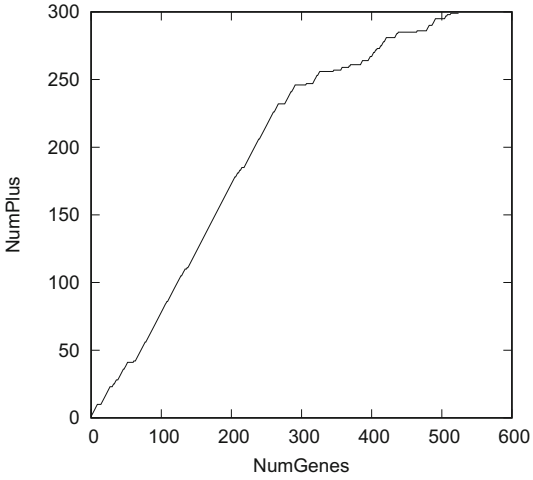
Answer 55 All that is missing is a command for printing the number of genes analyzed:

```
for i in $(seq 525)
do
    echo -n $i ' '        # print number of genes analyzed
    cut -f 4 mgGenes.txt  |
head -n $i                |
grep +                    |
wc -l
done
```

execute this

```
bash countGenes.sh |
gnuplot -p -e 'se xl "NumGenes"; se yl "NumPlus"; p "< cat "
     t "" w l'
```

to get the number of *M. genitalium* genes on the plus strand as a function of the number of genes.



Answer 56 The script is

```
for strand in + -
do
    for i in $(seq 525)
    do
        echo -n $i ' '
        cut -f 4 mgGenes.txt|
        head -n $i          |
        grep $strand        |
        wc -l
    done
    echo ' '
done
```

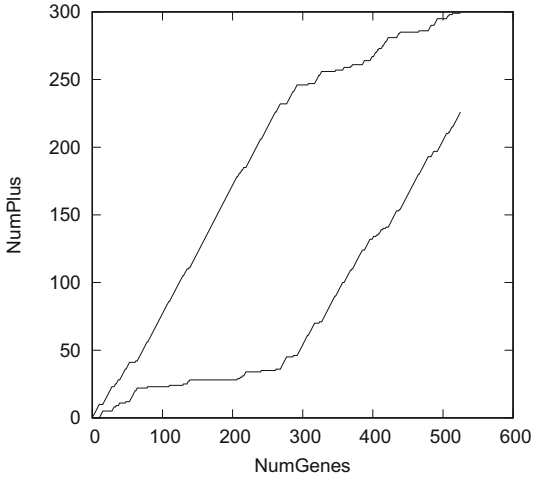
Notice the command

```
echo ' '
```

in the penultimate line, which separates the two data sets by a blank line. Run this

```
bash countGenes.sh |
gnuplot -p -e 'se xl "NumGenes"; se yl "NumPlus"; p "< cat "
     t "" w l'
```

to get



The blank line in the data set causes `gnuplot` to draw a second graph in the same plot window, which is useful for getting a first impression of the data. However, it does not use two plot symbols for distinguishing between the + and the – data. We learn how to do this later when we have gained more experience with `gnuplot`.

Answer 57 Carry out the substitution:

```
sed 's/-2/_2/' mgGenes.txt > mgGenes3.txt
```

and check its result

```
diff mgGenes2.txt mgGenes3.txt
```

No differences should be found.

Answer 58

```
cut -f 5 mgGenes.txt | sed '/^$/d' | wc -l
219
```

Answer 59 Save the edited file in a new file name

```
sed '56p' mgGenes3.txt > mgGenes4.txt
```

and compare the two files

```
diff mgGenes3.txt mgGenes4.txt
56a57
> MG_473         64367   64513     -    rpmG_2
```

To find out what this means, look at the two lines mentioned, 56 and 57, in both files:

```
sed -n '56p' mgGenes3.txt
MG_473  64367  64513    -    rpmG_2
sed -n '57p' mgGenes3.txt
MG_474  64527  64910    -
```

and

```
sed -n '56p' mgGenes4.txt
MG_473  64367  64513    -   rpmG_2
sed -n '57p' mgGenes4.txt
MG_473  64367  64513    -   rpmG_2
```

In other words, line 56 in `mgGenes4.txt` was appended to form line 57, which was summarized by `diff` as `56a57`.

Answer 60

```
sed -n '1,10p' mgGenes3.txt > t1.txt
head mgGenes3.txt > t2.txt
diff t1.txt t2.txt
```

Answer 61 The original command was

```
cut -f 5 mgGenes3.txt | grep -
```

Its `sed` version is

```
cut -f 5 mgGenes3.txt | sed -n '/-/p'
```

Answer 62 Smallest position:

```
cut -f 2,3 mgGenes3.txt | sed 's/\t/\n/' | sort -n | head -n
     1
```

Largest position:

```
cut -f 2,3 mgGenes3.txt | sed 's/\t/\n/' | sort -n | tail -n
     1
```

Answer 63 By accidentally omitting `-n` from `sort`, the smallest gene position would seem to be

```
cut -f 2,3 mgGenes3.txt | sed 's/\t/\n/' | sort | head -n 1
102454
```

Answer 64 Cut out the gene positions as given:

```
cut -f 2,3 mgGenes3.txt > t1.txt
```

Cut out the gene positions and sort them:

```
cut -f 2,3 mgGenes3.txt | sort -n > t2.txt
```

Check that the two files are identical

```
diff t1.txt t2.txt
```

Answer 65 Overlapping genes induce a list of positions that is not strictly ascending; in our example, this is

```
1000
2000
1990
3000
```

To find out whether any genes overlap in *M. genitalium*, we need to compare the sorted and the unsorted list of positions:

```
cut -f 2,3 mgGenes3.txt | sed 's/\t/\n/' | sort -n > t1.txt
cut -f 2,3 mgGenes3.txt | sed 's/\t/\n/' > t2.txt
diff t1.txt t2.txt
```

This returns a long list of differences. Overlapping genes appear to be quite common in *M. genitalium*.

Answer 66 There are many ways to count the gyrases; one example is to remove all other genes:

```
cut -f 5 mgGenes3.txt | sed -n -f filter.sed
gyrB
gyrA
```

where `filter.sed` contains a single line:

```
/gyr/p   # print gyrases
```

Answer 67

```
awk '{print $5}' mgGenes3.txt
```

Notice that the code inside the curly brackets is executed for every input line.

Answer 68 Enter

```
awk '{print "Sum: " $1 + $2}'
```

Then enter two numbers separated by a blank to get their sum. Press `C-c` twice to get out of this infinite loop.

Answer 69 Print genes on the plus strand

```
awk '$4~/[+]/{print}' mgGenes3.txt
```

and on the minus strand

```
awk '$4~/[-]/{print}' mgGenes3.txt
```

Notice that without argument, `print` prints the entire input line, which is equivalent to

```
print $0
```

Answer 70 Without the action block, each matching line is printed, for example

```
awk '$4~/[-]/' mgGenes3.txt
```

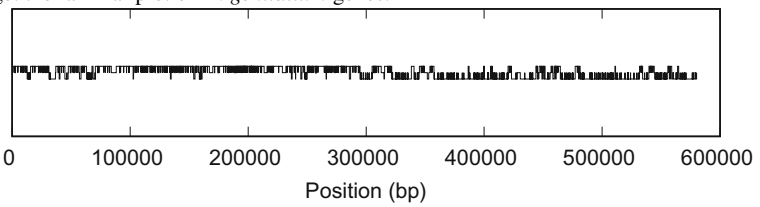
Answer 71 Use

```
awk -f drawGenes.awk mgGenes3.txt |
gnuplot -p pipe.gp
```

where `drawGenes.awk` is

```
$4-/[+]/{
    x = 1
}
$4-/[-]/{
    x = -1
}
{
    print $2 "\t" 0
    print $2 "\t" x
    print $3 "\t" x
    print $3 "\t" 0
}
```

to get the familiar plot of *M. genitalium* genes:



Answer 72 Find the shortest gene:

```
awk '{print $1 "\t" $3-$2+1}' mgGenes3.txt | sort -k 2 -n |
    head
MG_479   74
MG_489   74
MG_493   74
MG_496   74
MG_499   74
MG_495   75
MG_501   75
MG_502   75
MG_504   75
MG_512   75
```

So there are five genes of length 74. Find the longest gene:

```
awk '{print $1 "\t" $3-$2+1}' mgGenes3.txt | sort -k 2 -n |
    tail
MG_309   3678
MG_338   3813
MG_340   3879
MG_064   3996
MG_341   4173
MG_191   4335
```

```
MG_031   4356
MG_386   4851
MG_468   5352
MG_218   5418
```

There is a single gene of length 5418. Notice the option "`-k 2`" directing `sort` to use entries in the second column as sort keys.

Answer 73

```
awk '{s += $3-$2+1; c++}END{print "Avg: " s/c}' mgGenes3.txt
Avg: 1029.42
```

If, by any chance, you wrote = instead of +=, the program prints the length of the last gene divided by 525, approximately 1.54.

Answer 74 The program to compute the variance:

```
{
    l = $3 - $2 + 1        # length
    len[n++] = l           # store
}END{
    for(i=0; i<n; i++)     # mean
        m += len[i]
    m /= n
    for(i=0; i<n; i++) {   # variance
        x = m - len[i]
        s += x * x
    }
    printf "Var:  %e\n", s / (n - 1)
 }
```

Notice that the mean `for` loop has no curly brackets, while the variance `for` loop does. A `for` loop with a single line as action block needs no curly brackets, though they would not cause an error either. An action block consisting of more than one line needs to be delineated by curly brackets. This rule equally applies to `if`. When run on the gene lengths of *M. genitalium*, `var.awk` produces

```
awk -f var.awk mgGenes3.txt
Var:  6.623986e+05
```

The program `var` calculates the same, very large, variance:

```
awk '{print $3-$2+1}' mgGenes3.txt | var
#mean   var
1.029423e+03   6.623986e+05
```

Answer 75 Here is the program.

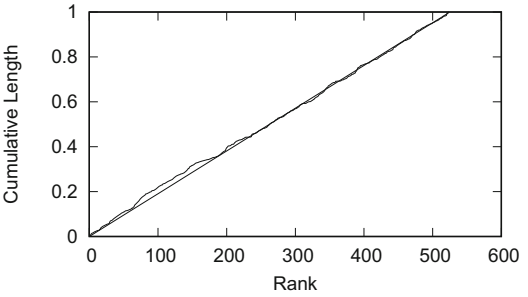
```
{
    l = $3-$2+1
    len[n++] = l
    sum += l
}END{
```

```
    for(i=0; i<n; i++){    # observed
        s += len[i] / sum
        print i "\t" s
    }
    print ""
    for(i=0; i<n; i++)     # expected
        print i "\t" i/n
}
```

Piping its result through

```
gnuplot -p -e 'set xlabel "Rank"; set ylabel "Cumulative
    Length"; plot
"< cat " title "" wi li'
```

gives



So there is no systematic bias with respect to gene length along the genome.

Answer 76 The number of names used

```
cut -f 5 mgGenes3.txt | sed '/^$/d' | wc -l
219
```

is equal to the number of unique names:

```
cut -f 5 mgGenes3.txt | sed '/^$/d' | sort | uniq | wc -l
219
```

So all names are unique.

Answer 77

```
awk '{print $3-$2+1}' mgGenes3.txt | sort | uniq | wc -l
365
```

Answer 78

```
awk '{print $3-$2+1}' mgGenes3.txt | sort | uniq -c | sort -
    n -r | head -n 5
      9 77
      7 76
      6 75
      5 74
      4 936
```

Answer 79 Generate the output with `uniq -c`

```
awk '{print $3-$2+1}' mgGenes3.txt |
sort                               |
uniq -c                            |
sed 's/^ *//'                      |
sort > t1.txt
```

The second `sort` is necessary to achieve sorting according to the count printed by `uniq`. Now reproduce this result with `uniqC.awk`

```
awk '{print $3-$2+1} ' mgGenes3.txt |
awk -f uniqC.awk                    |
sort > t2.txt
```

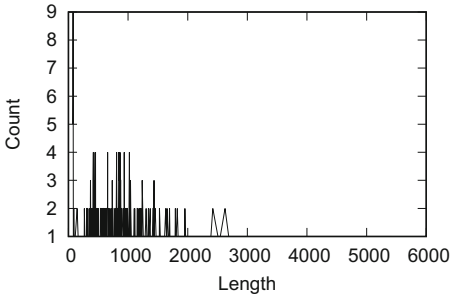
Check the two files are identical

```
diff t1.txt t2.txt
```

Answer 80

```
awk '{print $3-$2+1} ' mgGenes3.txt      |
awk -f uniqC.awk                         |
awk '{print $2 "\t" $1}'                 |
sort -n                                  |
gnuplot -p -e 'set xlabel "Length"; set ylabel "Count"; plot
     "< cat" title "" wi li'
```

gives



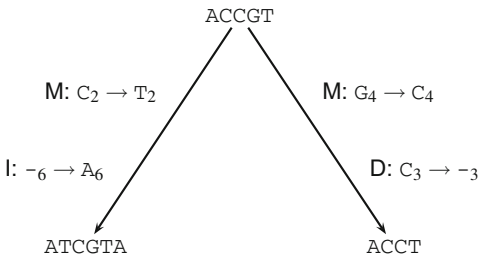
Answer 81 Replace the 10 in the `for` loop by `seqLen`

```
BEGIN{
     print ">Random_Sequence"
     srand(seed)
     s[0] = "A"; s[1] = "T"; s[2] = "C"; s[3] = "G"
     for(i=0; i<seqLen; i++){
          j = int(rand() * 4)
          printf("%s", s[j])
     }
     printf("\n")
}
```

and run

```
awk -v seed=$RANDOM -v seqLen=30 -f ranSeq.awk
```

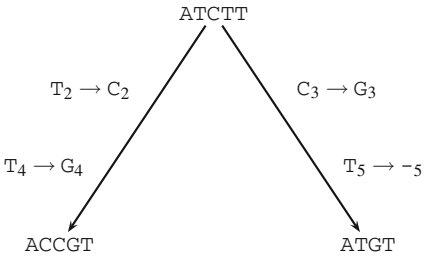
Answer 82 Here is one possible answer:



where M is the mutation, I is the insertion, and D is the deletion. You might have been tempted to mark the deletion event as a gap in the final sequence, but remember that sequences exist as gapless molecules. The three types of evolutionary events marked in the above graph, mutation, inversion, and deletion, only become visible by sequence comparison, that is, in an alignment:

```
ATCGTA
AC-CT-
```

Answer 83 Here is one solution out of many:



Answer 84 Here are five possible answers, you might well have found others:

```
ACCGT
-ATGT
```
Events = 3

```
ACCGT-
-A-TGT
```
Events = 6

```
ACCGT
A-TGT
```
Events = 2

```
ACCGT
AT-GT
```
Events = 2

```
ACCGT-
AT-GT
```
Events = 5

Answer 85 The re-scored results look like this:

ACCGT -ATGT Events = 3 Score = −11	ACCGT- -A-TGT Events = 6 Score = −30	ACCGT A-TGT Events = 2 Score = −7
ACCGT AT-GT Events = 2 Score = −7	ACCGT- AT-GT Events = 5 Score = −21	

Answer 86 This is expressed by

$$g = g_o + (l - 1) \times g_e.$$

Answer 87 Read `man echo` to learn that `-e` enables the interpretation of "unprintable" characters such as a newline:

```
echo -e '>Seq1\nACCGT'
>Seq1
ACCGT
```

And saved:

```
echo -e '>Seq1\nACCGT' > seq1.fasta
```

Without `-e`:

```
echo '>Seq1\nACCGT'
>Seq1\nACCGT
```

Answer 88 The command

```
gal -i seq1.fasta -j seq2.fasta
```

returns

```
Query: 1 ACCGT 5
           |  ||
Sbjct: 1 A-TGT 4
```

with a score of -7. Since by default gap opening (g_o) is -5 and gap extension (g_e) is -2, the gap scoring scheme is

$$g = g_o + l \times g_e.$$

Another alignment with the same score is

```
Query: 1 ACCGT 5
           |  ||
Sbjct: 1 A-TGT 4
```

Answer 89 Copy `hbb1.fasta` and `hbb2.fasta`

```
cp ../Data/hbb1.fasta .
cp ../Data/hbb2.fasta .
```

Note the dot at the end of the command, which means copy into the current directory. Look up the function of `hbb1.fasta`

```
head -n 1 hbb1.fasta
>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta
    (HBB), mRNA
```

and of `hbb2.fasta`

```
head -n 1 hbb2.fasta
>gi|6003531|gb|AF181832.1|AF181832 Homo sapiens hemoglobin
    beta subunit variant (HBB) mRNA, partial cds
```

So the first sequence is the mRNA of human beta hemoglobin, and the second a partial coding sequence (CDS) of the same protein. Run `gal`

```
gal -i hbb1.fasta -j hbb2.fasta
```

to find these two DNA sequences differ by large gaps at the 5' and 3' ends, and a single mismatch in between.

Answer 90 Running `gal` with default options shows that the mismatch is close to position 400. So rerunning `gal` with `-l 100` lets you conveniently find that the mutation is in position 397 in `hbb1.fasta` and 290 in `hbb2.fasta`.

Answer 91 The mutation is in position 290 of `hbb2.fasta`, which is translated in frame. The position of the mutation within the codon can thus be found as the remainder when dividing the position by 3, the modulo operation, $290 \bmod 3 = 2$. Hence, the mutation affects the two codons

```
GCC
GAC
```

which encode alanine and aspartate. The mutation is non-synonymous.

Answer 92 The probability of finding a stop codon in a random sequence is 3/64, so given a start codon, the expected distance to the next stop codons is $64/3 \approx 21$ amino acids. This is the expected length of open reading frames in random DNA sequences.

Answer 93 Compute the average ORF length:

```
awk -v seed=$RANDOM -v n=10000 -f simOrf.awk |
awk '{s+=$1;c++}END{print s/c}'
21.0919
```

This is close to the expectation of 21.

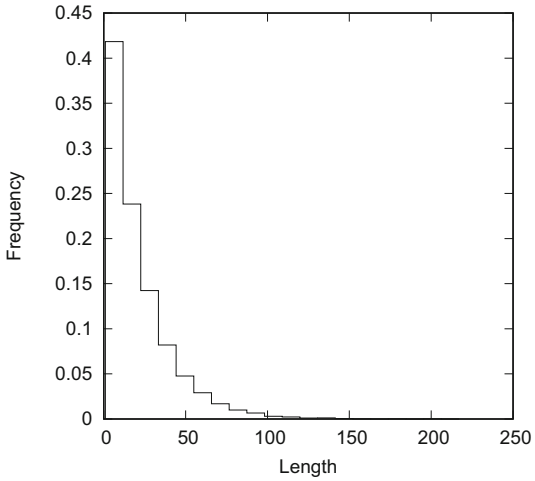
Answer 94

```
awk -v seed=$RANDOM -v n=1000 -f simOrf.awk |
histogram                                   |
gnuplot -p orf.gp
```

where`orf.gp` is

```
set xlabel "Length"
set ylabel "Frequency"
plot "< cat" title "" with lines
```

to get



Answer 95 The two most polar amino acids are aspartate and glutamate.

Answer 96 `Phe→Leu`: 1; `Phe→Trp`: 2; `Phe→Glu`: 3.

Answer 97 Set up the session:

```
mkdir AminoAcidMat
cd AminoAcidMat
cp ../Data/polarity.dat .
```

Find the least polar amino acid:

```
sed '/#/d' polarity.dat | sort -k 2 -n | head -n 1
Cys      4.8
```

The most polar:

```
sed '/#/d' polarity.dat | sort -k 2 -n | tail -n 1
Asp      13.0
```

Answer 98 Smallest: 1, `M` and `W`; largest: 6, `L`, `S`, and `R`.

Answer 99 There are two examples at the first codon position: `TTA` and `TTG` encode `L`, and so do `CTA`/`CTG`. There are no synonymous mutations at the second codon position.

Answer 100 The number of possible arrangements for n books on a shelf is n factorial

$$n(n-1)(n-2)...2 = n!$$

Compute

```
awk 'BEGIN{p = 1; for(i=2;i<=20;i++)p *= i; printf("%e\n", p
    )}'
2.432902e+18
```

to find $n! \approx 2.4 \times 10^{18}$.

Answer 101 The three possible changes at the second position yield serine (S), tyrosine (Y), and cysteine (C). Changes at the third position again yield leucine (L). With the exception of serine, these mutant amino acids also have a similar polarity as phenylalanine.

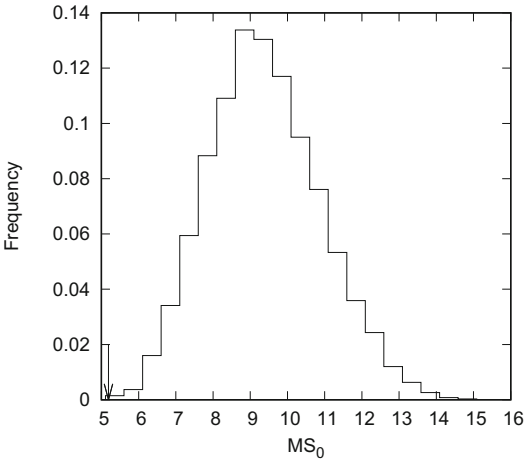
Answer 102 Execute

```
genCode -p polarity.dat  |
cut -f 2                 |
histogram                |
gnuplot -p ms0.gp
```

where `ms0.gp` is

```
set ylabel "Frequency"
set xlabel "MS_0"
set arrow from 5.19,0.02 to 5.19,0
plot "< cat" title "" with lines
```

This gives



The arrow indicates the MS_0 value of the natural code, 5.19, which is quite untypical when compared to the MS_0 values from random codes. Notice that this is the result of a simulation—yours is bound to differ slightly.

Answer 103 We got between 0 and 2 better codes. Since `genCode` goes through 10^4 random codes, the proportion of better codes is about 10^{-4}.

Answer 104 Run `genCode` with, say, 10^6 iterations

```
genCode -n 1000000 polarity.dat | grep ms0: | wc -l
117
```

When rejecting the null hypothesis that the natural code is not mutation-optimized, the error probability is $P = 117 \times 10^{-6} \approx 1.2 \times 10^{-4}$, so we reject and conclude that the code is indeed significantly optimized with respect to polarity.

Answer 105 Run commands like

```
genCode -n 1000000 hydropathy.dat  | grep ms0: | wc -l
```

to find

Attribute	P
Polarity	1.2×10^{-4}
Hydropathy	7.8×10^{-3}
Volume	0.11
Charge	0.56

Answer 106 A→S: 0.0028; S→A: 0.0035.

Answer 107

```
tail -n 20 pam1.txt | awk '{c++; s+=$(c+1)*0.05}END{print
  (1-s)*100}'
0.977
```

In other words, there is approximately a one percent probability of finding a mismatched amino acid, as would be expected for 1 PAM.

Answer 108 Compute M^{100}

```
pamPower -n 100 pam1.txt
```

to get (edited for legibility)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**.32**	.03	.07	.07	.03	.05	.08	.10	.03	.05	.03	.03	.04	.02	.11	.13	.13	.01	.02	.08
R	.01	**.44**	.02	.01	.01	.05	.01	.01	.05	.02	.01	.09	.03	.01	.03	.03	.02	.05	.00	.01
N	.03	.02	**.21**	.10	.01	.03	.05	.03	.07	.02	.01	.05	.01	.01	.02	.06	.04	.01	.02	.01
D	.04	.01	.11	**.31**	.00	.05	.16	.04	.04	.01	.01	.03	.01	.00	.02	.04	.03	.00	.01	.01
C	.01	.01	.01	.00	**.77**	.00	.00	.00	.01	.01	.00	.00	.01	.00	.00	.02	.01	.00	.02	.01
Q	.02	.04	.03	.04	.00	**.32**	.09	.01	.09	.01	.02	.03	.02	.00	.03	.02	.02	.00	.00	.01
E	.05	.01	.06	.17	.00	.12	**.32**	.03	.03	.02	.01	.03	.01	.00	.03	.03	.03	.00	.01	.02
G	.11	.02	.07	.07	.02	.03	.06	**.55**	.02	.02	.01	.03	.02	.01	.04	.10	.05	.00	.01	.04
H	.01	.04	.06	.03	.01	.08	.02	.01	**.43**	.01	.01	.02	.01	.01	.02	.01	.01	.01	.02	.01
I	.02	.01	.02	.01	.01	.01	.01	.01	.01	**.32**	.05	.01	.06	.04	.01	.01	.03	.00	.01	.12
L	.03	.02	.02	.01	.00	.04	.01	.01	.03	.12	**.61**	.02	.21	.09	.02	.02	.03	.03	.03	.10
K	.03	.18	.10	.05	.01	.07	.05	.02	.04	.03	.02	**.51**	.09	.01	.03	.05	.06	.01	.01	.02
M	.01	.01	.00	.00	.00	.01	.00	.00	.00	.02	.04	.02	**.29**	.01	.00	.01	.01	.00	.00	.02
F	.01	.01	.01	.00	.00	.00	.00	.01	.02	.04	.04	.00	.03	**.60**	.00	.01	.01	.02	.17	.01
P	.06	.03	.02	.02	.01	.04	.03	.02	.03	.01	.02	.02	.01	.01	**.49**	.06	.04	.00	.00	.02
S	.10	.05	.10	.05	.05	.04	.05	.08	.03	.03	.01	.05	.03	.02	.08	**.25**	.12	.03	.02	.03
T	.09	.03	.06	.04	.02	.03	.03	.03	.02	.05	.02	.05	.04	.01	.04	.10	**.31**	.01	.02	.05
W	.00	.01	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.00	.00	.00	**.79**	.01	.00
Y	.01	.00	.01	.00	.02	.00	.01	.00	.02	.01	.01	.00	.01	.13	.00	.01	.01	.02	**.59**	.01
V	.06	.02	.02	.02	.02	.02	.02	.02	.02	.21	.07	.02	.09	.02	.03	.03	.06	.00	.02	**.42**

Compute M^{1000}

```
pamPower -n 1000 pam1.txt
```

to get

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**.09**	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.09	.07	.08	.09
R	.04	**.04**	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.05	.04	.04
N	.04	.04	**.04**	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04
D	.05	.05	.05	**.05**	.04	.05	.05	.05	.05	.05	.05	.05	.05	.04	.05	.05	.05	.04	.04	.05
C	.03	.03	.03	.03	**.09**	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.02	.03	.03
Q	.04	.04	.04	.04	.03	**.04**	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.03	.03	.04
E	.05	.05	.05	.05	.05	.05	**.05**	.05	.05	.05	.05	.05	.05	.05	.05	.05	.05	.04	.04	.05
G	.09	.09	.09	.10	.09	.09	.10	**.10**	.09	.09	.09	.10	.09	.08	.10	.09	.10	.07	.08	.09
H	.03	.03	.03	.03	.03	.03	.03	.03	**.03**	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03
I	.04	.04	.04	.04	.04	.04	.04	.04	.04	**.04**	.04	.04	.04	.04	.04	.04	.04	.03	.04	.04
L	.09	.08	.08	.08	.08	.08	.08	.08	.08	.09	**.10**	.09	.09	.10	.09	.08	.09	.09	.09	.09
K	.08	.09	.09	.09	.07	.09	.09	.08	.08	.08	.08	**.09**	.08	.08	.09	.08	.09	.08	.07	.08
M	.02	.01	.02	.01	.01	.02	.01	.01	.01	.02	.02	.02	**.02**	.02	.02	.02	.02	.01	.02	.02
F	.04	.04	.04	.04	.04	.04	.04	.04	.04	.05	.05	.04	.05	**.07**	.04	.04	.04	.05	.07	.04
P	.06	.05	.06	.06	.05	.06	.06	.06	.05	.05	.05	.06	.05	.05	**.06**	.06	.06	.05	.05	.05
S	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	.07	**.07**	.07	.06	.06	.07
T	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	.06	**.06**	.05	.05	.06
W	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	.01	**.10**	.01	.01
Y	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03	.05	.03	.03	.03	.04	**.05**	.03
V	.06	.06	.06	.06	.06	.06	.06	.06	.06	.07	.07	.07	.07	.07	.07	.06	.07	.06	.06	**.07**

which has much more uniform values within a row than M^{100}.

Answer 109 Here is the computation:

```
for a in 1 2 5 10 20 50 100 200 500 1000
do
    echo -n $a ' '
    pamPower -n $a pam1.txt    |
      tail -n +2               |
      awk '{s+=$(NR+1)}END{print (1-s/20)*100}'
done
```

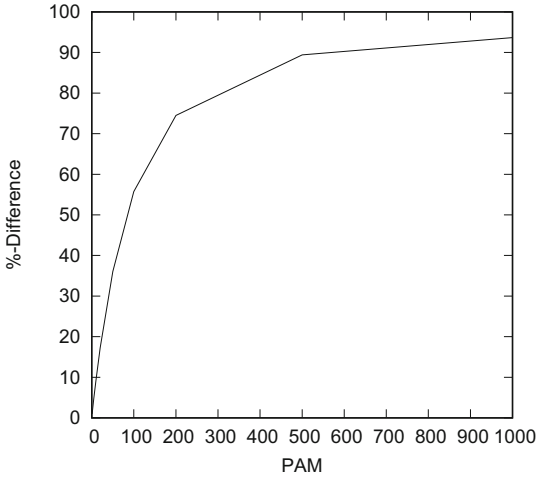
Visualize the result

```
bash pamPower.sh  |
gnuplot -p pamPower.gp
```

where `pamPower.gp` is

```
set xlabel "PAM"
set ylabel "%-Difference"
plot "< cat" title "" with lines
```

to get



Answer 110 To make it easier to connect the sorted frequencies with amino acids, we printed the amino acids in front of the frequencies:

```
awk -f labelFreq.awk aa.txt | sort -k 2 -n
```

where `labelFreq.awk` is

```
/^#/{
    # Store amino acid designations
    for(i=2; i<=21; i++)
       aa[i-1] = $i
}
!/^#/{
    c++  # Count lines
    print aa[c] "\t" $1
}
```

This tells us that the least frequent amino acid is tryptophane (`W`), the most frequent glycine (`G`).

Answer 111 The amino acid frequencies

```
 A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
.09 .04 .04 .05 .03 .04 .05 .09 .03 .04 .09 .08 .01 .04 .05 .07 .06 .01 .03 .07
```

are similar to the columns in M^{1000}. In other words, after 1000 PAM of evolutionary time, homologous proteins contain pairs of amino acids at frequencies similar to the frequencies in random sequence pairs.

Answer 112 Here is the pipeline:

```
pamPower -n 70 pam1.txt |
pamNormalize -a aa.txt
```

Observe that normalization makes the matrix symmetrical, which is what we need when scoring pairwise alignments, where we cannot tell the direction of change.

Answer 113 Here is the final pipeline

```
pamPower -n 70 pam1.txt |
pamNormalize -a aa.txt  |
pamLog > pam70sm.txt
```

This pipeline generates the following matrix (typeset to improve readability):

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	-4	-1	-1	-4	-2	-1	0	-4	-2	-4	-4	-3	-5	0	1	1	-9	-5	-1
R	-4	**8**	-3	-6	-5	0	-5	-6	0	-3	-6	2	-2	-6	-2	-1	-4	0	-8	-5
N	-1	-3	**6**	3	-7	-1	0	-1	1	-3	-5	0	-5	-5	-3	1	0	-7	-3	-4
D	-1	-6	3	**6**	-9	0	3	-1	-2	-5	-8	-2	-7	-9	-4	-1	-2	-11	-7	-5
C	-4	-5	-7	-9	**9**	-9	-9	-7	-5	-4	-10	-9	-6	-8	-6	-1	-4	-11	-2	-4
Q	-2	0	-1	0	-9	**7**	2	-4	2	-5	-3	-1	-2	-8	-1	-3	-3	-9	-8	-4
E	-1	-5	0	3	-9	2	**6**	-2	-2	-4	-6	-2	-5	-9	-3	-2	-3	-11	-6	-4
G	0	-6	-1	-1	-7	-4	-2	**6**	-6	-6	-7	-5	-7	-6	-3	0	-3	-10	-9	-3
H	-4	0	1	-2	-5	2	-2	-6	**8**	-6	-4	-3	-6	-3	-2	-3	-4	-6	-1	-5
I	-2	-3	-3	-5	-4	-5	-4	-6	-6	**7**	0	-4	1	-1	-5	-4	-1	-10	-4	3
L	-4	-6	-5	-8	-10	-3	-6	-7	-4	0	**6**	-5	2	-1	-4	-6	-4	-6	-4	0
K	-4	2	0	-2	-9	-1	-2	-5	-3	-4	-5	**6**	0	-9	-4	-2	-1	-7	-8	-5
M	-3	-2	-5	-7	-6	-2	-5	-7	-6	1	2	0	**10**	-2	-5	-3	-2	-9	-7	0
F	-5	-6	-5	-9	-8	-8	-9	-6	-3	-1	-1	-9	-2	**8**	-7	-4	-5	-2	4	-5
P	0	-2	-3	-4	-6	-1	-3	-3	-2	-5	-4	-4	-5	-7	**7**	0	-2	-9	-9	-4
S	1	-1	1	-1	-1	-3	-2	0	-3	-4	-6	-2	-3	-4	0	**5**	2	-3	-5	-3
T	1	-4	0	-2	-4	-3	-3	-3	-4	-1	-4	-1	-2	-5	-2	2	**6**	-8	-5	-1
W	-9	0	-7	-11	-11	-9	-11	-10	-6	-10	-6	-7	-9	-2	-9	-3	-8	**13**	-2	-11
Y	-5	-8	-3	-7	-2	-8	-6	-9	-1	-4	-4	-8	-7	4	-9	-5	-5	-2	**9**	-5
V	-1	-5	-4	-5	-4	-4	-4	-3	-5	3	0	-5	0	-5	-4	-3	-1	-11	-5	**6**

The score of the alignment is accordingly $1 + 0 + 6 + 5 + 3 = 15$.

Answer 114 There are 14 pairs of mismatched amino acids with a positive score:

#	Amino acid pair	Score
1	AS	1
2	AT	1
3	RK	2
4	ND	3
5	NH	1
6	NS	1
7	DE	3
8	QE	2
9	QH	2
10	IM	1
11	IV	3
12	LM	2
13	FY	4
14	ST	2

Such residue pairs tend to have similar structures and polarities. Phenylalanine (F) and tyrosine (Y) have the largest positive score. Both amino acids have similar shapes (Fig. 2.3) and polarities (Fig. 2.4).

Answer 115 The correct reading frame contains a long stretch of amino acids without stop codons (*):

```
transeq -frame 3 -filter < hbb1.fasta > hbb1prot.fasta
transeq -frame 1 -filter < hbb2.fasta > hbb2prot.fasta
```

Align the sequences

```
gal -i hbb1prot.fasta -j hbb2prot.fasta -p -m pam70sm.txt
```

to get

```
Query:              >NM_000518.4_3 Homo sapiens hemoglobin, beta (HBB),
     mRNA
   Length:          208
Subject:            >AF181832_1 Homo sapiens hemoglobin beta subunit
                    variant (HBB) mRNA, partial cds
   Length:          163
Score:              964.0


Q: 1 ICF*HNCVH*QPQTDTMVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFE 58
                                          NVDEVGGEALGRLLVVYPWTQRFFE
S: 1 ----------------------------------NVDEVGGEALGRLLVVYPWTQRFFE 25


Q: 59 SFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENF 118
      SFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENF
S: 26 SFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENF 85


Q: 119 RLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH*ARFLAVQFLLKVPLFP 177
       RLLGNVLVCVL HHFGKEFTPPVQAAYQKVVAGVANALAHKYH*ARFLAVQFLLKVPLFP
S: 86  RLLGNVLVCVLDHHFGKEFTPPVQAAYQKVVAGVANALAHKYH*ARFLAVQFLLKVPLFP 144


Q: 178 KSNY*TGGYYEGP*ASGFCLIKNIYFHC 203
       KSNY*TGGYYEGP*ASG
S: 145 KSNY*TGGYYEGP*ASG----------X 160
//
```

The row between query and subject indicates the match state: the residue for an exact match and a blank for a mismatch with a score of zero or less. The alanin (A) at position 130 in the query is mismatched with the aspartate (D) at position 97 in the subject, as we had seen before.

Answer 116 As the PAM number goes toward infinity, the scores for homologous pairs of amino acids all become 0. The last pair of amino acids with a score > 0 is tryptophan/tryptophan in PAM2000, which means this is the most conserved, or most slowly evolving, amino acid.

```
pamPower -n 1000 pam1.txt |
pamNormalize -a aa.txt    |
pamLog
```

```
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V
A  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
R  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
N  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
D  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
C  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
Q  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
E  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
G  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
H  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
I  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
L  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
K  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
M  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
F  0  0  0  0  0  0  0  0  0  0  1  0  0  2  0  0  0  1  1  0
P  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
S  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
T  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
W -1  0  0 -1 -1  0 -1 -1  0  0  0  0  0  1  0  0  0  7  1 -1
Y  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  1  2  0
V  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0

pamPower -n 2000 pam1.txt |
pamNormalize -a aa.txt    |
pamLog

   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V
A  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
R  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
N  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
D  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
C  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Q  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
E  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
G  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
H  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
I  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
L  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
K  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
M  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
F  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
P  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
S  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
T  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
W  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0
Y  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
V  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

pamPower -n 3000 pam1.txt |
pamNormalize -a aa.txt    |
pamLog
```

```
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V
A  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
R  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
N  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
D  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
C  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Q  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
E  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
G  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
H  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
I  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
L  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
K  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
M  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
F  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
P  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
S  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
T  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
W  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Y  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
V  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Answer 117 With `pam1000sm.txt`, the C-terminal end of the alignment is changed; the rest of the alignment remains the same. However, with `pam2000sm.txt` and `pam3000sm.txt`, the alignment consists almost entirely of mismatches. Homologous positions cannot be distinguished from random matches using these score matrices.

Answer 118 We can type

```
bash pamPower2.sh | gnuplot -p pamPower.gp
```

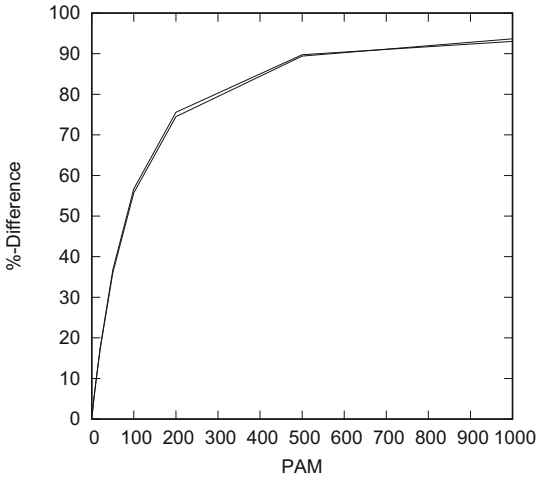
where `pamPower2.sh` is

```
for f in approx percentDiff
do
    for a in 1 2 5 10 20 50 100 200 500 1000
    do
        echo -n $a ' '
        pamPower -n $a pam1.txt    |
          tail -n +2               |
          awk -f ${f}.awk
    done
    echo ' '
done
```

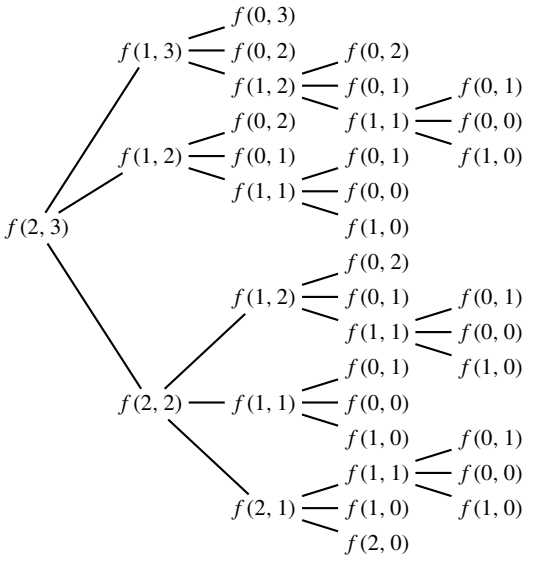
and `approx.awk` is essentially the AWK code from Problem 107:

```
{
    c++
    s += $(c+1) * 0.05
}END{
    print (1-s) * 100
}
```

The `gnuplot`-script `pamPower.gp` is unchanged—which illustrates the usefulness of storing more complex plot commands in a file. The final plot shows that the approximation is very similar to the exact computation:



Answer 119 Here is the desired tree:



The end result is the number of leaves in that tree, that is nodes without descendants. There are 25 leaves and hence 25 possible global alignments between two sequences of lengths 2 and 3.

Answer 120

	0	1	2	3
0				
1				
2				

→

	0	1	2	3
0	1	1	1	1
1	1			
2	1			

→

	0	1	2	3
0	1	1	1	1
1	1	3		
2	1			

→

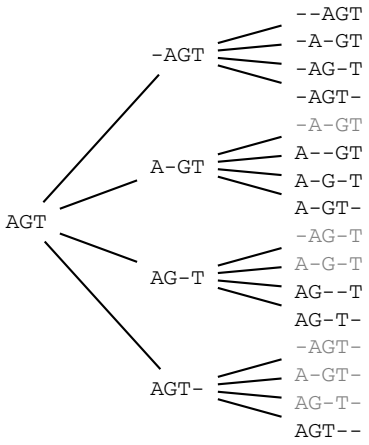
	0	1	2	3
0	1	1	1	1
1	1	3	5	
2	1			

→→

	0	1	2	3
0	1	1	1	1
1	1	3	5	7
2	1	5	13	**25**

This time we obtain 25 much quicker.

Answer 121 The first thing to realize is that no global alignment of S_1 and S_2 can be shorter than the longer sequence, that is 3, or longer than the sum of their lengths, that is, 5. This means that, say, S_1 can only occur with 0, 1, or 2 gaps. We can write down the longer sequence with 0 gaps and then construct all sequences with 1 gap, and again from these the sequences with 2 gaps. For the latter, we need to watch out for repeats, which are shown in gray:



From these, we can construct all 25 alignments:

AGT AC-	AGT A-C	AGT -AC	-AGT AC--	-AGT A-C-
-AGT A--C	A-GT -AC-	A-GT -A-C	A-GT AC--	AG-T -AC-
AG-T A-C-	AG-T --AC	AGT- --AC	AGT- -A-C	AGT- A-C
--AGT AC---	-A-GT A-C--	-AG-T A--C-	-AGT- A---C	A--GT -AC-
A-G-T -A-C-	A-GT- -A--C	AG--T --AC-	AG-T- --A-C	AGT-- ---AC

Answer 122 The command

```
numAl -m 106 -n 106
```

tells us there are 7.8×10^{79} possible alignments. This is also approximately the number of atoms in the observable universe spcitewik:obs.

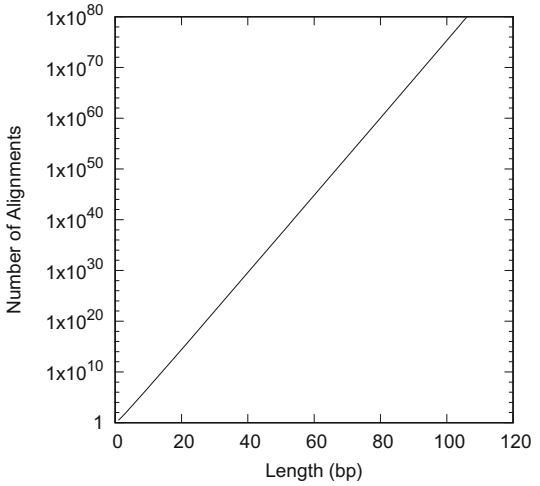
Answer 123 Run

```
bash numAl.sh          |
awk '{print ++c, $3}' |
gnuplot -p -e 'set xl "Length (bp)"; set yl "Number of
   Alignments"; set log y; p "< cat" t "" w l'
```

where `numAl.sh` is

```
for a in $(seq 106)
do
   numAl -m $a -n $a
done
```

to get



Answer 124 Run

```
bash numAl2.sh
```

where `numAl2.sh` is

```
for a in $(seq 106)
do
   numAl -t -m $a -n $a
done
```

to observe that the computations quickly become very slow. We gave up at length 14.

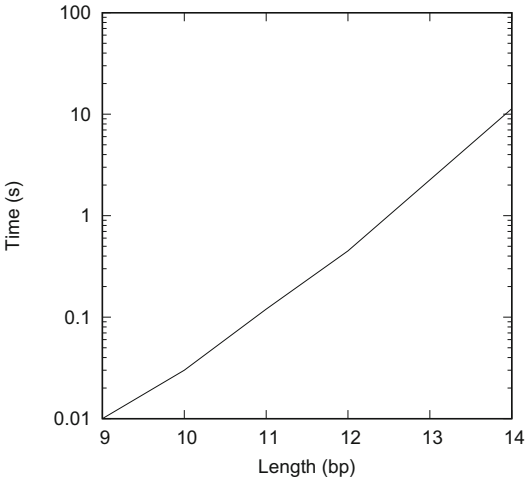
Answer 125 To make subsequent manipulations more convenient, first write the run times to a file

```
for a in $(seq 14); do numAl -t -m $a -n $a; done > times.
    txt
```

Then filter out the times greater than zero and plot them:

```
cat times.txt                  |
awk '{if($6>0)print ++c, $6}' |
gnuplot -p -e 'set xl "Length (bp)"; set yl "Time (s)"; set
     log y; p "< cat" t "" w l'
```

to get



Answer 126 Linear functions are described by

$$f(x) = ax + b.$$

The slope of the straight line, a, is found by considering, for example, the run times for sequences lengths 13 and 14, which were 2.25 s and 11.33 s in our case. Take logarithms

```
awk 'BEGIN{print log(2.25)/log(10)}'
0.352183
awk 'BEGIN{print log(11.33)/log(10)}'
1.05423
```

to get $a = 1.05 - 0.35 = 0.70$ and $b = 1.05 - 14 \times 0.7 = -8.75$. So our run time is

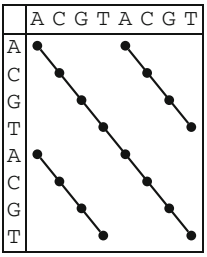
$$10^{0.7 \times 106 - 8.75} = 10^{65.45}$$

seconds or

$$\frac{10^{65.45}}{60 \times 60 \times 24 \times 365.25} \approx 8.9 \times 10^{57}$$

years. This dwarfs the 10^{10} years the universe has existed. Choosing the appropriate algorithm can make an enormous difference.

Answer 127
Here is our dot plot:



The matches off the main diagonal indicate duplications.

Answer 128 The fist two lines of `dmAdhAdhdup.fasta` are

```
head -n 2 dmAdhAdhdup.fasta
>DMADH X78384.1 D.melanogaster Adh and Adh-dup genes.
TGTATTTTCCAATTAGGTGATAGAACTTGTGTGCACACACACATATAGTTCTATATCAAC
```

So `dmAdhAdhdup.fasta` contains the genes *Adh* and its duplication *Adh-dup* from *D. melanogaster*. The first two lines of `dgAdhAdhdup.fasta` are

```
head -n 2 dgAdhAdhdup.fasta
>DGADHDUP X60113.1 D.guanche Adh and Adh-dup genes for
    alcohol dehydrogenase.
TCTAGATTGCATCACTCGTGCCGCCCTACGTTGTGAAGCACCACGCCCTGGACCCCGTTT
```

Correspondingly, `dgAdhAdhdup.fasta` contains the *D. guanche* versions of *Adh* and *Adh-dup*. Now count the nucleotides:

```
cchar dmAdhAdhdup.fasta
# Total number of input characters: 4761
# Char  Count   Fraction
A       1417    0.297627
C       1007    0.211510
G       989     0.207729
T       1348    0.283134
```

and

```
cchar dgAdhAdhdup.fasta
# Total number of input characters: 4433
# Char  Count   Fraction
A       1279    0.288518
C       998     0.225130
G       936     0.211144
T       1220    0.275209
```

In summary,

File	Genes	Organism	Length (bp)
`dmAdhAdhdup.fasta`	*Adh &Adh-dup*	*D. melanogaster*	4761
`dgAdhAdhdup.fasta`	*Adh & Adh-dup*	*D. guanche*	4433

Answer 129 The two sequences are 4761 and 4433 bp long, and the corresponding dot plot would therefore contain $4761 \times 4433 = 21,105,513$ cells. This is quite a large number. We therefore need an approach that is more efficient than checking each cell and placing a dot for every match.

Answer 130 Run

```
#len|strId:pos_1|...|strId:pos_n|seq
37|f2:3292|f1:3287|AGCAAGGTTCTCATGACCAAGAATATAGCGGTGAGTG
```

The longest repeat has length 37.

Answer 131 When running

```
cat *.fasta | randomizeSeq | repeater
```

a couple of times, we commonly find repeats with lengths 11 to 14. Since these lengths are not even close to the naturally occurring longest repeat of 37, it is extremely unlikely that this repeat has occurred by chance.

Answer 132 The raw result is

```
cat dmAdhAdhdup.fasta dgAdhAdhdup.fasta |
repeater -m 12                          |
head -n 3
#len|strId:pos_1|...|strId:pos_n|seq
13|f2:3096|f1:3129|AAAATAGATAAAT
12|f2:3371|f1:3389|AAACTAATTAAG
```

This is converted by `dotPlotFilter.awk` to

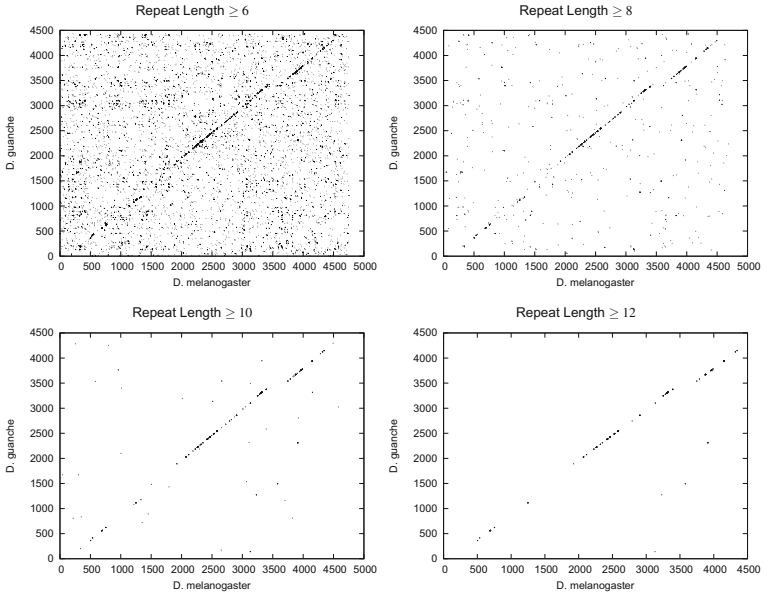
```
cat dmAdhAdhdup.fasta dgAdhAdhdup.fasta |
repeater -m 12                          |
head -n 3                               |
awk -f dotPlotFilter.awk
3129    3096
3141    3108

3389    3371
3400    3382
```

For a given repeat, `dotPlotFilter.awk` prints the x- and y-coordinates of the start and end of the repeat separated by blank lines.

Answer 133 Commands like

```
cat dmAdhAdhdup.fasta dgAdhAdhdup.fasta |
repeater -m 12                          |
awk -f dotPlotFilter.awk                |
gnuplot -p -e 'set xl "D. melanogaster"; set yl "D. guanche
      "; p "< cat" t "" w l'
```

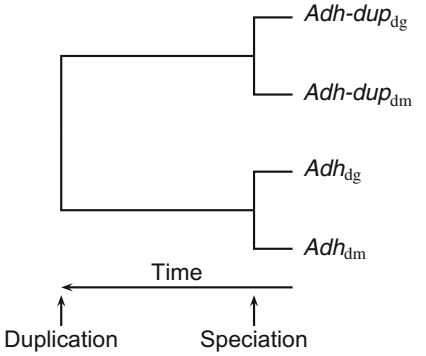
give results like



The repeat length determines the density of the plot.

Answer 134 Since the paralogs are contained in both species, the duplication must predate the speciation event. Accordingly, the orthologs have a more recent last common ancestor than the paralogs.

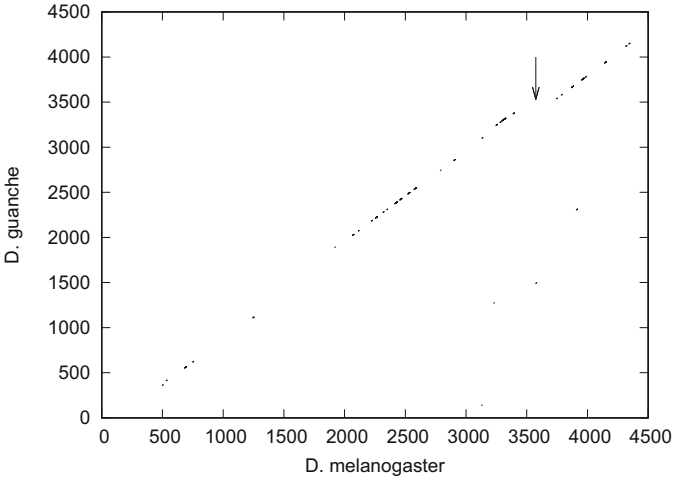
Answer 135 A cartoon phylogeny would look like this:



The branch lengths are unknown, but the branch order *is* known, and hence the relative order of gene divergence due to duplication and speciation.

Answer 136 The homology between the orthologous pairs of genes is clearly visible along the main diagonal of the plot. However, there are no significant off-diagonal repeats, and hence the paralogous relationships between the duplicated genes is so ancient as to have become invisible in a dot plot.

Answer 137 The insertion, marked by an arrow below, causes a shift in the main diagonal of homology:



There was an insertion into the *D. melanogaster* sequence.

Answer 138 The coordinates for *D. melanogaster* are found by typing

```
grep CDS dmAdhAdhdup.gb
```

and for *D. guanche* by entering

```
grep CDS dgAdhAdhdup.gb
```

to get

Organism	*Adh*	*Adh-dup*
D. melanogaster	2021..2119,2185..2589,2660..2926	3226..3321,3748..4152,4204..4521
D. guanche	1984..2076,2145..2549,2613..2879	3221..3316,3540..3944,4007..4345

Answer 139 There are many ways of converting the CDS coordinates recorded in the genbank file to a set of intervals. Our version is

```
grep CDS dmAdhAdhdup.gb |
sed -f cds.sed          |
awk -f break.awk > cdsDm.txt
```

where `cds.sed` is

```
s/.*(//      # remove everything up to opening bracket
s/)//        # remove closing bracket
s/\.\./ /g   # substitute blanks for pairs of dots
s/,/ /g      # substitute blanks for commas
```

and `break.awk` is

```
{
    printf "%s\t%s\n%s\t%s\n%s\t%s\n", $1, $2, $3, $4, $5, $6
}
```

Next, write the dot plot data to file:

```
cat dmAdhAdhdup.fasta dgAdhAdhdup.fasta |
repeater -m 12                          |
awk -f dotPlotFilter.awk > dotPlot.dat
```

Append the exon coordinates:

```
awk -f boxesX.awk cdsDm.txt >> dotPlot.dat
```

where `boxesX.awk` is

```
BEGIN{
    h = 150 # height
}{
    s = $1  # start
    e = $2  # end
    printf("%d\t%d\n%d\t%d\n%d\t%d\n%d\t%d\n", s, 0, s, h, e
      , h, e, 0)
}
```

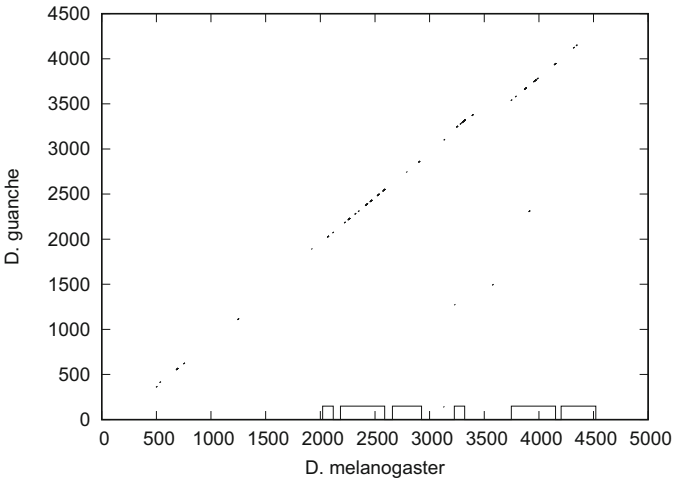
and `cdsDm.txt` contains the CDS coordinates of *D. melanogaster*:

```
2021 2119
2185 2589
2660 2926
3226 3321
3748 4152
4204 4521
```

Now plot the result:

```
cat dotPlot.dat |
gnuplot -p -e 'set xl "D. melanogaster"; set yl "D. guanche
      "; p "< cat" t "" w l'
```

to get



It seems the insertion affected only an intron.

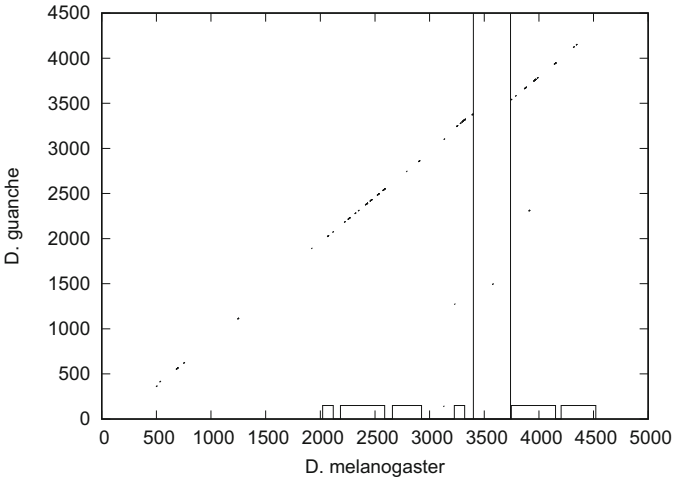
Answer 140 Draw the plot as

```
gnuplot -p cdsDm.gp
```

where `cdsDm.gp` is

```
set xlabel "D. melanogaster"
set ylabel "D. guanche"
set arrow from 3400,0 to 3400,4500 nohead
set arrow from 3740,0 to 3740,4500 nohead
plot "dotPlot.dat" title "" with lines
```

We plotted directly from the data file `dotPlot.dat`; alternatively, it could have been read from the standard input. However, in a script it is usually more convenient to read from a file containing the input data. By drawing vertical lines along the gap borders, we can clearly see that the insertion affected an intron:



Answer 141 Generate the exon coordinates for *D. guanche*:

```
grep CDS dgAdhAdhdup.gb |
sed -f cds.sed          |
awk -f break.awk > cdsDg.txt
```

Add an empty line to the data file

```
echo >> dotPlot.dat
```

Add the exons for *D. guanche*:

```
awk -f boxesY.awk cdsDg.txt >> dotPlot.dat
```

where `boxesY.awk` is

```
BEGIN{
    h = 150 # height
}{
    s = $1  # start
    e = $2  # end
    printf("%d\t%d\n%d\t%d\n%d\t%d\n%d\t%d\n", 0, s, h, s, h,
         e, 0, e)
}
```

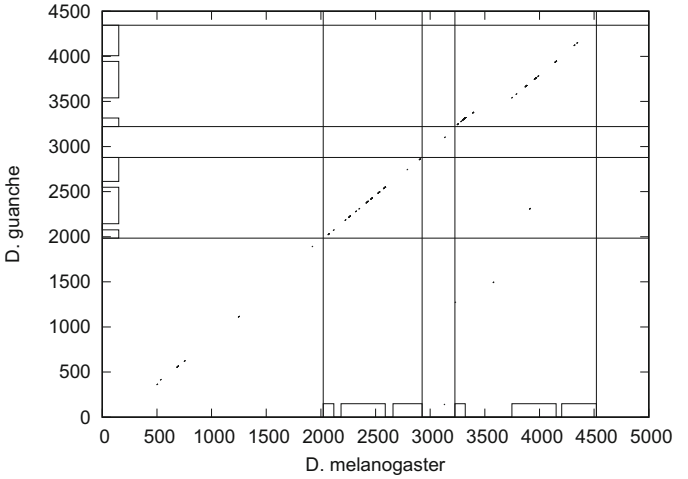
Now plot the data

```
gnuplot -p adhCds.gp
```

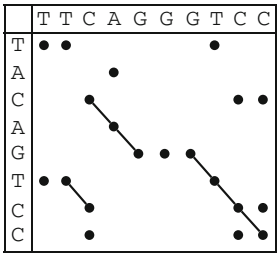
where the `gnuplot` script `adhCds.gp` is

```
set termoption dash
reset
set xlabel "D. melanogaster"
set ylabel "D. guanche"
# D. m. CDS
set arrow from 2021,0 to 2021,4500 nohead lt 2
set arrow from 2926,0 to 2926,4500 nohead
set arrow from 3226,0 to 3226,4500 nohead
set arrow from 4521,0 to 4521,4500 nohead
# D. g. CDS
set arrow from 0,1984 to 5000,1984 nohead
set arrow from 0,2879 to 5000,2879 nohead
set arrow from 0,3221 to 5000,3221 nohead
set arrow from 0,4345 to 5000,4345 nohead
# plot
plot "dotPlot.dat" title "" with lines
```

to get the CDS of *Adh* and *Adh-dup* for *D. melanogaster* and *D. guanche*:



Answer 142 Here is the dot plot:



There are three possible global alignments with the maximum number of matches:

```
TTCAGGGTCC    TTCAGGGTCC    TTCAGGGTCC
TACA--GTCC    TACA-G-TCC    TACAG--TCC
```

The last column of these alignments corresponds the bottom right corner of the dot plot matrix.

Answer 143 An alignment gap corresponds to a shift in a match diagonal.

Answer 144 $F(2, 1)$ refers to $S_1[1..1] =$ A and $S_2[1..2] =$ AC.

Answer 145

	-	A	C	T
	0	1	2	3
- 0	0	$\leftarrow -1$	$\leftarrow -2$	$\leftarrow -3$
A 1				
C 2				

Answer 146

	-	A	C	T
	0	1	2	3
- 0	0	$\leftarrow -1$	$\leftarrow -2$	$\leftarrow -3$
A 1	$\uparrow -1$			
C 2	$\uparrow -2$			

Answer 147

	-	A	C	T
	0	1	2	3
- 0	0	$\leftarrow -1$	$\leftarrow -2$	$\leftarrow -3$
A 1	$\uparrow -1$	$\nwarrow 1$	$\leftarrow 0$	$\leftarrow -1$
C 2	$\uparrow -2$	$\uparrow 0$	$\nwarrow 2$	$\leftarrow 1$

Answer 148 The traceback returns

```
ACT
AC-
```

This has score 1, as expected from the entry in the lower right-hand cell of the alignment matrix.

Answer 149 In a global alignment, the bottom right-hand cell of the alignment matrix contains the score of the optimal alignment between S_1 and S_2; in our case, this is 4.

Answer 150 There are three cooptimal alignments implied by the global alignment matrix:

```
TTCAGGGTCC
| ||  ||||
TACA--GTCC

TTCAGGGTCC
| || | |||
TACA-G-TCC

TTCAGGGTCC
| |||  |||
TACAG--TCC
```

all of which have score 4, the entry in the cell from where the traceback started.

Answer 151 Construct directory and change into it:

```
mkdir OptimalAlignment
cd OptimalAlignment
```

Run `gal` with the desired score scheme:

```
gal -i seq1.fasta -j seq2.fasta -O 0 -E -1 -I -1

Query:              >seq1
   Length:          10
Subject:            >seq2
   Length:          8
Score:              4.0
Strand:             Plus / Plus


Query: 1 TTCAGGGTCC 10
         | ||  ||||
Sbjct: 1 TACA--GTCC 8
```

This can be changed; for example, if the mismatch score is ≤ -3 and all other parameters are left unchanged, the optimal alignment becomes

```
TT-CAGGGTCC
 | ||  ||||
-TACA--GTCC
```

Answer 152 Again, the answer is yes, but only for gap score schemes lacking biological plausibility: For example, we might reward gaps by setting the gap extension parameter to 1. In other words, the score increases rather than decreases when gaps are inserted. The alignment then becomes

```
gal -E 1 -i s1.fasta -j s2.fasta
TTCAGGGT-----CC
        |     ||
-------TACAGTCC
```

Answer 153 The *Adh-dup* from *D. guanche* should contain a large gap, which corresponds to an insertion in *D. melanogaster*.

Answer 154 Enter

```
gal -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta | less
```

and find that between positions 3426 and 3531 in *D. guanche* there are large gaps. Now look up the CDS coordinates for *D. guanche*

```
grep CDS dgAdhAdhdup.gb
CDS join(1984..2076,2145..2549,2613..2879)
CDS join(3221..3316,3540..3944,4007..4345)
```

The first CDS refers to *Adh*, the second to *Adh-dup*. The intron between the first and the second exon of *Adh-dup* has coordinates 3317–3539, which corresponds to the gapped region. Therefore, the gap is located in the first intron of the *D. guanche Adh-dup*.

Answer 155 First initialize, the fill in the local alignment matrix:

Initialize

		-	T	A	C	G	T
		0	1	2	3	4	5
-	0	0	0	0	0	0	0
G	1	0					
A	2	0					
C	3	0					
G	4	0					
A	5	0					

Fill in

		-	T	A	C	G	T
		0	1	2	3	4	5
-	0	0	0	0	0	0	0
G	1	0	0	0	0	↖ 1	0
A	2	0	0	↖ 1	0	0	0
C	3	0	0	0	↖ 2	← 1	0
G	4	0	0	0	↑ 1	↖ 3	← 2
A	5	0	0	↖ 1	0	↑ 2	↖ 2

Answer 156 The traceback looks like this:

		-	T	A	C	G	T
		0	1	2	3	4	5
-	0	0	0	0	0	0	0
G	1	0	0	0	0	↖ 1	0
A	2	0	0	↖ 1	0	0	0
C	3	0	0	0	↖ 2	← 1	0
G	4	0	0	0	↑ 1	↖ 3	← 2
A	5	0	0	↖ 1	0	↑ 2	↖ 2

The resulting alignment is

```
ACG
ACG
```

Its score is 3, the entry in the cell from which the traceback started.

Answer 157 The command

```
lal -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
```

returns

```
Query:            >DMADH X78384.1 D.MELANOGASTER ADH AND ADH
                  -DUP GENES.
   Length:        4761
   Hit:           2182 - 2594
Subject:          >DGADHDUP X60113.1 D.guanche Adh and Adh-
                  dup genes for alcohol dehydrogenase.
   Length:        4433
   Hit:           2142 - 2554
Score:            217.0
Strand:           Plus / Plus


Query: 2182 TAGAACCTGGTGATCCTCGACCGCATTGAGAACCCGGCTGCCATTGCCGAGCTGAAGGCA 2241
            |||||||||||| ||||| || |||||||| || || ||||||||||||||| |||||||||
Sbjct: 2142 TAGAACCTGGTCATCCTGGATCGCATTGACAATCCAGCTGCCATTGCCGAACTGAAGGCA 2201


Query: 2242 ATCAATCCAAAGGTGACCGTCACCTTCTACCCCTATGATGTGACCGTGCCCATTGCCGAG 2301
             ||||||| ||||||||||||||||||||||| |||||||||| || ||  | || |||
Sbjct: 2202 GTCAATCCCAAGGTGACCGTCACCTTCTACCCTTATGATGTGACTGTACCTGTCGCAGAG 2261


Query: 2302 ACCACCAAGCTGCTGAAGACCATCTTCGCCCAGCTGAAGACCGTCGATGTCCTGATCAAC 2361
            |||||||| || ||||||||||||||| |||||| | |||||| |||||||||||||| |||
Sbjct: 2262 ACCACCAAACTCCTGAAGACCATCTTTGCCCAGATCAAGACCATCGATGTCCTGATAAAC 2321


Query: 2362 GGAGCTGGTATCCTGGACGATCACCAGATCGAGCGCACCATTGCCGTCAACTACACTGGC 2421
            || ||||| ||||| |||||||| ||||| ||||| || |||||||| ||||||||||||
Sbjct: 2322 GGTGCTGGCATCCTCGACGATCATCAGATTGAGCGTACTATTGCCGTTAACTACACTGGC 2381


Query: 2422 CTGGTCAACACCACGACGGCCATTCTGGACTTCTGGGACAAGCGCAAGGGCGGTCCCGGT 2481
            |||||||||||||| || |||||||||| |||||||||||||||||||||| || |||
Sbjct: 2382 CTGGTCAACACCACCACAGCCATTCTGGATTTCTGGGACAAGCGCAAGGGCGGCCCAGGT 2441


Query: 2482 GGTATCATCTGCAACATTGGATCCGTCACTGGATTCAATGCCATCTACCAGGTGCCCGTC 2541
            || ||||| |||||||||||| ||||| || || || ||||||||||||||||||||||||
Sbjct: 2442 GGCATCATTTGCAACATTGGCTCCGTTACCGGTTTTAATGCCATCTACCAGGTGCCCGTT 2501


Query: 2542 TACTCCGGCACCAAGGCCGCCGTGGTCAACTTCACCAGCTCCCTGGCGGTAAG 2594
            ||||| |||| |||||| || ||||| ||||||||||||||||||||||||||
Sbjct: 2502 TACTCTGGCAGCAAGGCGGCGGTGGTAAACTTCACCAGCTCCCTGGCGGTAAG 2554
```

Hence, the coordinates of the best local alignment are as follows:

Organism	Local alignment
D. melanogaster	2182–2594
D. guanche	2142–2554

As before, we look up the coordinates of the CDS for *Adh* and *Adh-dup* in *D. melanogaster*

```
grep CDS dmAdhAdhdup.gb
   CDS             join(2021..2119,2185..2589,2660..2926)
   CDS             join(3226..3321,3748..4152,4204..4521)
```

and similarly for *D. guanche*

```
grep CDS dgAdhAdhdup.gb
   CDS             join(1984..2076,2145..2549,2613..2879)
   CDS             join(3221..3316,3540..3944,4007..4345)
```

to find that the best alignment corresponds to the second exon of *Adh*.

Answer 158 Compute a single alignment:

```
time lal -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
```

This takes approximately 2.8 s, while computing two optimal local alignments

```
time lal -n 2 -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
```

takes approximately 22.5 s, eight times longer. The reason for this is that the top local alignment is surrounded by many cells with high scores. But the paths starting from these cells all intersect the optimal local alignment. It is therefore hard for the algorithm to find the starting point of the next best distinct alignment.

Answer 159 The coordinates of the second best alignment are as follows:

Organism	Local alignment
D. melanogaster	3829–4162
D. guanche	3621–3954

As before, look up the CDS coordinates

```
grep CDS *.gb
```

to find that the second best alignment corresponds to exon 2 of *Adh-dup*.

Answer 160 The score of the alignment is

```
gal -i dmAdhCds.fasta -j dmAdhdupCds.fasta |
grep Score
Score:              -1631.0
```

Answer 161 Compute the scores from random alignments

```
randomizeSeq -n 1000 dmAdhCds.fasta |
gal -i dmAdhdupCds.fasta            |
grep Score                          |
awk '{print $2}' > scores.dat
```

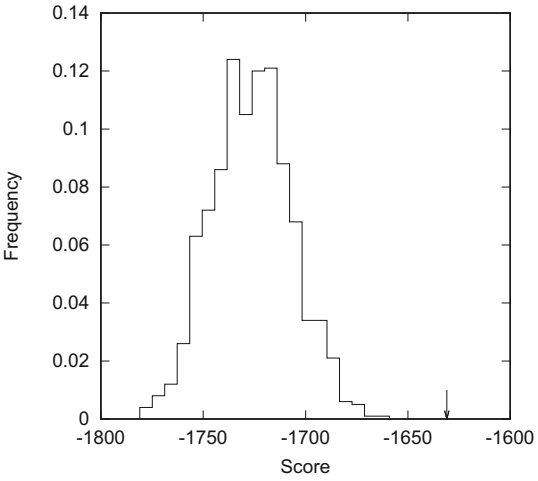
Plot the scores

```
histogram scores.dat | gnuplot -p scores.gp
```

where `scores.gp` is

```
set arrow from -1631,0.01 to -1631,0 # observed
set xl "Score"
set yl "Frequency"
plot[-1800:-1600][] "< cat" title "" with lines
```

to get



It is highly unlikely that the observed score (arrow) occurs between shuffled, nonhomologous versions of the input sequences.

Answer 162 First, remind yourself of the length of `dmAdhAdhdup.fasta` using `cchar`:

```
cchar dmAdhAdhdup.fasta
# Total number of input characters: 4761
```

Cut out the first and last kb:

```
cutSeq -r 1-1000    dmAdhAdhdup.fasta > f1.fasta
cutSeq -r 3762-4761 dmAdhAdhdup.fasta > f2.fasta
```

Compute the original score

```
gal -i f1.fasta -j f2.fasta | grep Score
Score:              -1385.0
```

Compute the random scores:

```
randomizeSeq -n 1000 f2.fasta |
gal -i f1.fasta               |
grep Score                    |
awk '{print $2}' > scoresR.dat
```

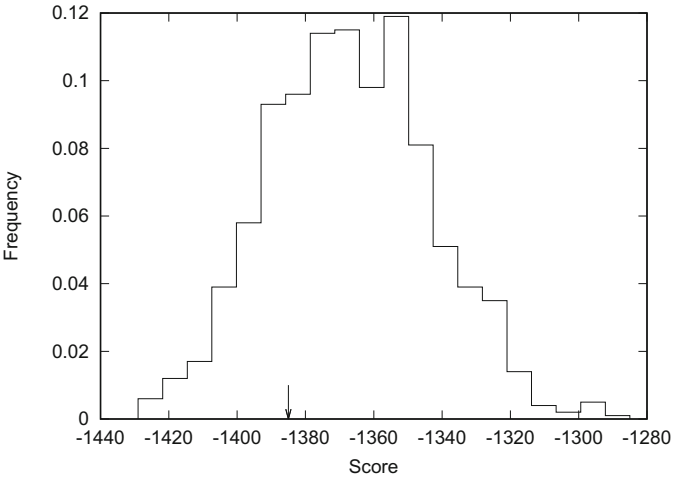
Plot the random scores

```
histogram scoresR.dat | gnuplot -p scoresR.gp
```

where `scoresR.gp` is

```
set arrow from -1385,0.01 to -1385,0 # observed
set xl "Score"
set yl "Frequency"
plot[][] "< cat" title "" with lines
```

to get



As expected, the score between the two arbitrary DNA segments (arrow) is not different from random. In other words, only scores to the right of the distribution of random scores are indicative of homology.

Answer 163 The substitution rates K_1 and K_2 are between orthologs, and thus refer to the split between *D. melanogaster* and *D. guanche*. The other four are between paralogs and refer to the *Adh* duplication.

Answer 164 Extract exon 2 from *Adh* and *Adh-dup* of *D. melanogaster*:

```
cutSeq -r 2185-2589 dmAdhAdhdup.fasta > dmAdhE2.fasta
cutSeq -r 3748-4152 dmAdhAdhdup.fasta > dmAdhdupE2.fasta
```

Repeat for *D. guanche*:

```
cutSeq -r 2145-2549 dgAdhAdhdup.fasta > dgAdhE2.fasta
cutSeq -r 3540-3944 dgAdhAdhdup.fasta > dgAdhdupE2.fasta
```

Answer 165

```
gal -i dmAdhE2.fasta -j dgAdhE2.fasta |
sed -n '/|/p'                         |
sed 's/ //g'                          |
awk '{s+=length($1)}END{print s}'
356
```

Answer 166 Exon 2 is 405 bp long and there were 356 matches. So the number of mismatches per site is

$$\pi = (405 - 356)/405 \approx 0.12.$$

Answer 167 Convert 0.12 mismatches per site to the number of substitutions per site:

$$K_1 = -\frac{3}{4}\log\left(1 - \frac{4}{3} \times 0.12\right) \approx 0.13.$$

Answer 168 Compute the number of matches

```
gal -i dmAdhdupE2.fasta -j dgAdhdupE2.fasta |
sed -n '/|/p'                               |
sed 's/ //g' | awk '{s+=length($1)}END{print s}'
324
```

So the number of mismatches per site is

$$\pi_2 = (405 - 324)/405 \approx 0.2,$$

and

$$K_2 = -\frac{3}{4}\log\left(1 - \frac{4}{3}\pi_2\right) \approx 0.23.$$

Answer 169 To compute M_3, we use the same commands as before:

```
gal -i dmAdhE2.fasta -j dmAdhdupE2.fasta |
sed -n '/|/p'                            |
sed 's/ //g'                             |
awk '{s+=length($1)}END{print s}'
```

We get

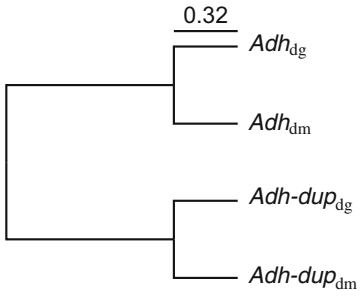
Comparison	Matches
M_3 : Adh_{dm} / $Adh\text{-}dup_{\text{dm}}$	228
M_4 : $Adh\text{-}dup_{\text{dg}}$ / Adh_{dg}	224
M_5 : Adh_{dm} / $Adh\text{-}dup_{\text{dg}}$	235
M_6 : $Adh\text{-}dup_{\text{dm}}$ / Adh_{dg}	226

That is, on average there are $M_{\text{dup}} = (228 + 224 + 235 + 226)/4 = 228.25$ matches. The number of mismatches per site is thus $\pi_{\text{dup}} = (405 - 228.25)/405 \approx 0.436$ and $K_{\text{dup}} \approx 0.65$.

Answer 170 The average between K_1 and K_2 is $(0.13 + 0.23)/2 = 0.18$. A rough estimate of the age of duplication is

$$0.65/0.18 \times 32 \approx 116$$

million years. When drawn as a tree, these divergence times look like this:



Answer 171 Take care of the preliminaries

```
mkdir KeywordTrees
cd KeywordTrees
```

and write the program `split.awk`:

```
BEGIN{
    n = split(t, ta, "")
    for(i=1; i<=n; i++)
        print ta[i]
}
```

Answer 172 Run

```
awk -v t=CACAGACACAT -v p=ACA -f naive.awk
```

where `naive.awk` is

```
BEGIN{
    n = split(t, ta, "")
    m = split(p, pa, "")
    for(i=1; i<=n-m; i++){
        for(j=1; j<=m; j++){
            if(ta[i+j-1] != pa[j])
              break
    }
    if(j == m + 1)
        print i
    }
}
```

Answer 173 Run

```
echo -e '>Seq\nACGTCG' |
awk -f naive2.awk -v file=mgGenome.fasta
122599
```

where `naive2.awk` is

```
BEGIN{
    cmd = "tail -n +2 " file
    while(cmd | getline)
       t = t $1
}
!/^>/{
    p = p $1
}
END{
    n = split(t, ta, "")
    m = split(p, pa, "")
    for(i=1; i<=n-m; i++){
        for(j=1; j<=m; j++){
            if(ta[i+j-1] != pa[j])
               break
        }
        if(j == m + 1)
            print i
    }
}
```

Answer 174 Reverse complement the sequence:

```
revComp mgGenome.fasta > mgGenomeR.fasta
```

and run

```
echo -e '>Seq\nCGGCCT' |
awk -f naive2.awk -v file=mgGenomeR.fasta
270306
```

to find one copy of the motif on the reverse strand.

Answer 175 Run

```
awk -f monoNuc.awk -v n=100 | fold
```

where `monoNuc.awk` is

```
BEGIN{
    print ">Mononuc"
    for(i=0; i<n; i++)
       printf("A")
    printf("\n")
}
```

Answer 176 Generate the text files

```
awk -f monoNuc.awk -v n=1000000 | fold > 1mb.fasta
awk -f monoNuc.awk -v n=2000000 | fold > 2mb.fasta
```

Measure the run times

```
awk -f monoNuc.awk -v n=10 |
  time awk -f naive2.awk -v file=1mb.fasta | tail
awk -f monoNuc.awk -v n=20 |
  time awk -f naive2.awk -v file=2mb.fasta | tail
```

where we got 2.65 s and 9.56 s, respectively. This roughly 3.6-fold increase in run time is somewhat smaller than the expected $2 \times 2 = 4$-fold increase.

Answer 177 The command

```
time naiveMatcher -p AAAAAAAAAAAAAAAAAAAA 2mb.fasta | tail
```

takes 0.282 s, compared to 9.56 s for the AWK script. This is a 34-fold speedup obtained just by switching from AWK to C.

Answer 178 Run

```
bash runNaive.sh > runNaive.dat
```

where `runNaive.sh` is

```
for a in 10 20 50 100 200 500 1000 2000 5000 10000
do
    echo -n ${a} ' '
    awk -f monoNuc.awk -v n=${a} |
        fold > pattern.fasta
```

```
    /usr/bin/time -p naiveMatcher -P pattern.fasta 2mb.fasta
         2>&1 |
        grep real |
        sed 's/real //'
done
```

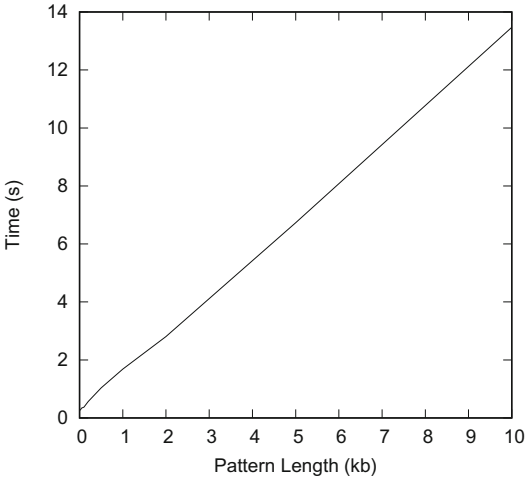
and plot

```
gnuplot -p plot1.gp
```

where `plot1.gp` contains

```
set xlabel "Pattern Length (kb)"
set ylabel "Time (s)"
plot "runNaive.dat" using ($1/1000):2 title "" with lines
```

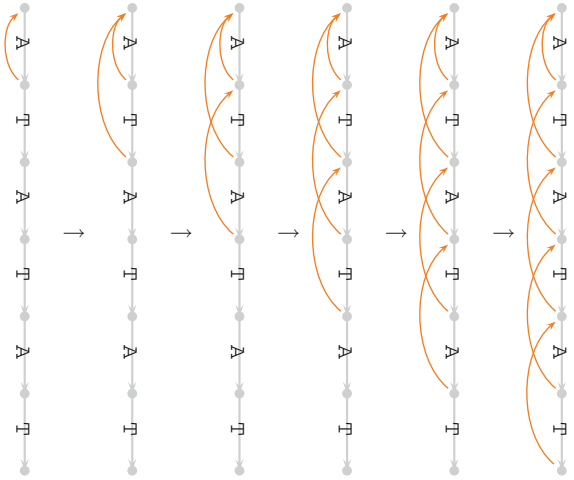
to get



In the artificial situation of an alphabet consisting of just a single character, the run time of `naiveMatcher` is linear in the pattern length.

Answer 179

Answer 180



Answer 181 The command

```
/usr/bin/time -p keywordMatcher -p AAAAAAAAAAAAAAAAAAAA 2mb.
     fasta 2>&1 |
grep real
```

runs in 0.40 s. This is actually slower than the naïve version, which took 0.26 s:

```
/usr/bin/time -p naiveMatcher -p AAAAAAAAAAAAAAAAAAAA 2mb.
    fasta 2>&1 |
grep real
```

Answer 182 Execute

```
bash runKeyword.sh > runKeyword.dat
```

where `runKeyword.sh` is almost identical to `runNaive.sh`, only the line

```
/usr/bin/time -p naiveMatcher -P pattern.fasta 2mb.fasta 2>&1 |
```

is replaced by

```
/usr/bin/time -p keywordMatcher -f pattern.fasta 2mb.fasta 2>&1|
```

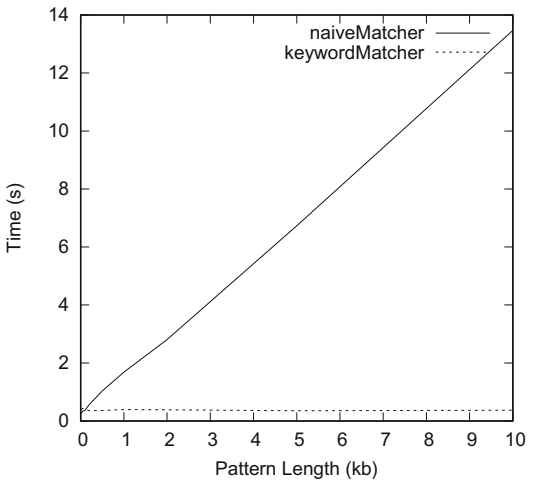
Plot `runKeyword.dat` together with `runNaive.dat` using

```
gnuplot -p plot2.gp
```

where `plot2.gp` is

```
set xlabel "Pattern Length (kb)"
set ylabel "Time (s)"
plot "runNaive.dat" using ($1/1000):2 title "naiveMatcher"
    with lines,\
"runKeyword.dat" using ($1/1000):2 title "keywordMatcher"
    with lines
```

to get



The run time of `keywordMatcher` is independent of the pattern length, while that of `naiveMatcher` is proportional to the pattern length. Notice that this observation only applies to our extreme case, where pattern and text consist of a single type of nucleotide.

Answer 183 We first measure the run times of `naiveMatcher`:

```
bash runNaiveLen.sh > runNaiveLen.dat
```

where `runNaiveLen.sh` is

```
for a in 1 2 5 10 20 50 100
do
    echo -n ${a} ' '
    ranseq -l ${a}000000 > tmp.fasta
    /usr/bin/time -p naiveMatcher -p TTTAACCTCCGGCGGAGTTT
        tmp.fasta 2>&1 |
        grep real |
        sed 's/real //'
done
```

Likewise, we execute

```
bash runKeywordLen.sh > runKeywordLen.dat
```

where `runKeywordLen.sh` is

```
for a in 1 2 5 10 20 50 100
do
    echo -n ${a} ' '
    ranseq -l ${a}000000 > tmp.fasta
    /usr/bin/time -p keywordMatcher -p TTTAACCTCCGGCGGAGTTT
        tmp.fasta 2>&1 |
        grep real |
        sed 's/real //'
done
```

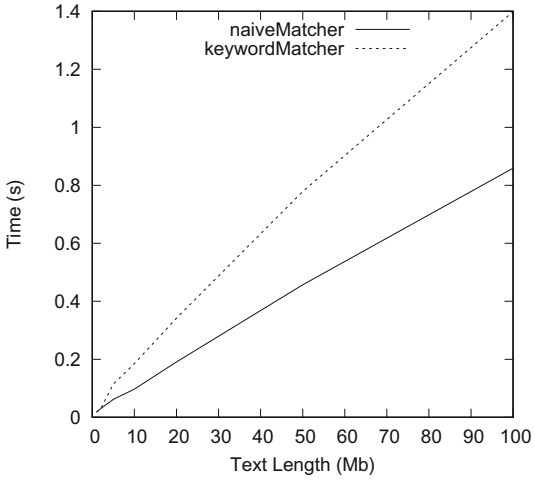
The two result files are plotted

```
gnuplot -p plot3.gp
```

where `plot3.gp` is

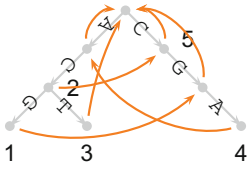
```
set xlabel "Text Length (Mb)"
set ylabel "Time (s)"
set key top center
plot "runNaiveLen.dat" title "naiveMatcher" with lines,\
"runKeywordLen.dat" title "keywordMatcher" with lines
```

to get



With random sequences, where on average only a small portion of the prefix is checked before a mismatch is found, the simple naïve method beats the more complex method based on pattern preprocessing.

Answer 184 Here is the keyword tree with all failure links included:



Answer 185 You could enter, for example,

```
keywordMatcher -t kt.tex -p 'ACG|AC|ACT|CGA|C' 2mb.fasta >
  /dev/null
```

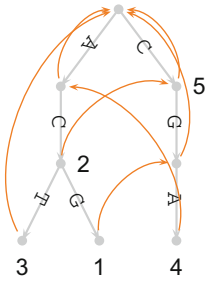
followed by

```
latex ktWrapper.tex; dvips ktWrapper.dvi
```

and

```
gv ktWrapper.ps &
```

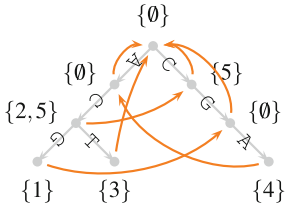
to get



This is not quite as nicely laid out as the manually generated keyword tree, but it does the job. Deleting `\date{}` results in an automatically generated date produced by `\maketitle`.

Answer 186 Walking into the tree recovers a match to $P_2 = \texttt{AC}$, followed by a match to $P_1 = \texttt{ACG}$. Then, follow the mismatch link twice to find $P_5 = \texttt{C}$. However, the first occurrence of `C` is missed that way. To prevent this, the matches at a particular point in the tree are not restricted to the pattern that may end at that point, but to any pattern that can be reached via the failure links. In practice, the preprocessing of a keyword tree includes a step where for every node the failure links are followed, resulting in an output set consisting of the node labels encountered along the way. Each output set may contain zero, one, or several patterns.

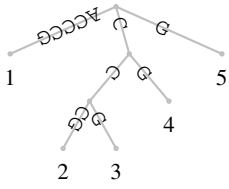
Answer 187 For most nodes of our keyword tree, the output set simply consists of the node label; the exception is node 2, where the output set now also includes P_5:



Answer 188

$S[1..]$ ACCCG
$S[2..]$ CCCG
$S[3..]$ CCG
$S[4..]$ CG
$S[5..]$ G

Answer 189 Here is our first suffix tree:

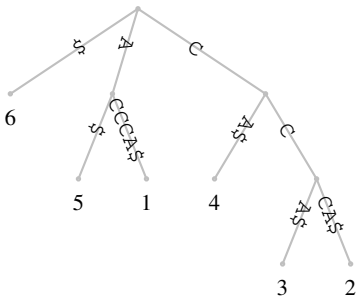


Answer 190 Start at the root and walk into the tree until CC has been matched. Then, look up the leaf labels of the subtree rooted on the node connected to the edge on which the search ended. These labels indicate the starting positions of CC in S, positions 3 and 2 in our example.

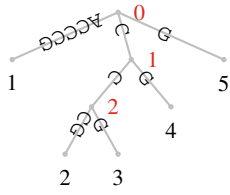
Answer 191 The last suffix, $S[5..] =$ A, does not generate a mismatch and hence no leaf in the suffix tree. This always happens when a suffix is a prefix of another suffix, like in our example where $S[5..] =$ A is a prefix of the suffix $S[1..] =$ ACCCA. To guarantee that each suffix generates a mismatch and hence a leaf when it is threaded into the tree, a so-called sentinel character is added at the end of S: $S =$ ACCCA$. This sentinel character, denoted by $, is not a nucleotide and hence cannot occur anywhere in the sequence.

Answer 192

1	2	3	4	5	6
A	C	C	C	A	$



Answer 193 The string depths are marked in red:



Answer 194 Look for the node with the greatest string depth. Its path label is the longest repeat in S, which is CC in our case.

Answer 195 Assuming your current directory is BiProblems, make the new directory, change into it, and copy the genome file:

```
mkdir SuffixTrees
cd SuffixTrees
cp ../Data/mgGenome.fasta .
```

Then execute

```
repeater -i mgGenome.fasta
#len|strId:pos_1|...|strId:pos_n|seq
243|389491|390403|TTTTTCAGCAGTTGGTTG...
```

to find that the genome of *M. genitalium* contains a repeat of 243 bp that occurs at positions 389,491 and 390,403.

Answer 196

```
# Total number of input characters: 580076
# Char  Count    Fraction
A       200544   0.345720
C       91515    0.157764
G       92306    0.159127
T       195711   0.337389
```

The probability of drawing AA is $0.35^2 \approx 0.12$.

Answer 197

```
cchar mgGenome.fasta  |
sed '/^#/d'           |
awk '{s+=$3^2}END{printf "P_m = %f\n", s}'
P_m = 0.283565
```

Answer 198 We need to solve

$$1 = P_{\mathrm{m}}^{l} \times L^2$$

for l. By rearranging and taking logarithms, we get

$$l = \frac{\log(1/L^2)}{\log(P_{\mathrm{m}})}.$$

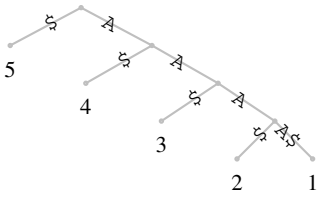
By substituting $P_{\mathrm{m}} = 0.284$ and $L = 580,076$, we find $l \approx 21.1$. This is much shorter than the observed longest repeat of length 243.

Answer 199 Our basic computation is

```
randomizeSeq mgGenome.fasta | repeater
```

which returns values like 23, 19, 21, 20, 19, 20, 21, 19, and so on. From 100 iterations, we found an average maximum length of 20.1, which is close to the expected 21.1, but still significantly smaller. One reason for this might be that in our model all starting points in the matrix are independent of each other, which is a simplification.

Answer 200 The suffix tree for AAAA looks like this:



During its construction, each suffix needed to be threaded from its beginning to its end into the intermediate tree. This means that for a sequence of length n,

$$n-1+n-2...+1 = n(n-1)/2$$

character comparisons are needed. The run time of naïve suffix tree construction is therefore proportional to $n(n-1)/2$, which scales as n^2 or $O(n^2)$. This is similar to the run time of optimal alignment, that is, too slow for genomics.

Answer 201 Use commands like

```
ranseq -l 1000000 | time repeater
```

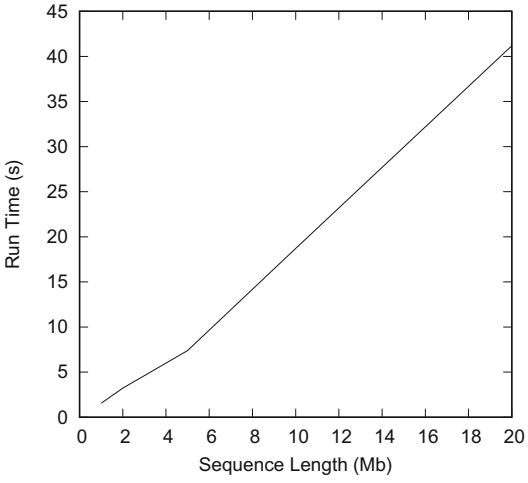
Collect the results in `time.dat` and plot them using

```
gnuplot -p stTime.gp
```

where `stTime.gp` is

```
set xlabel "Sequence Length (Mb)"
set ylabel "Run Time (s)"
plot "time.dat" title "" with lines
```

to get



For sequences of length n, the run time of `repeater` is $O(n)$. This linear run time behavior is optimal in the sense that it cannot be improved upon. The algorithm implemented in `repeater` was devised in 1995 [48] and is described in detail in a classic textbook of bioinformatics [21].

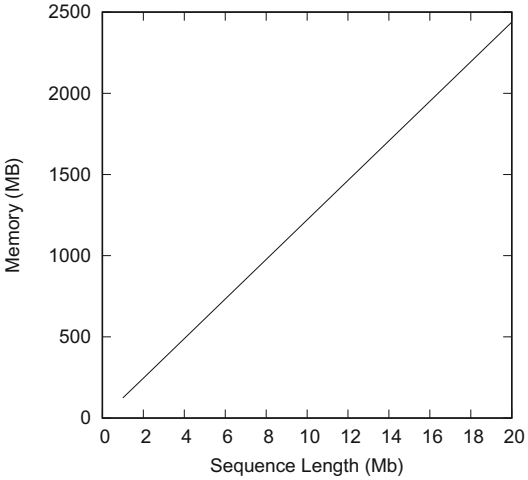
Answer 202 Say, `memory.dat` contains the memory measurements. Then

```
gnuplot -p stMemory.gp
```

where `stMemory.gp` is

```
set xlabel "Sequence Length (Mb)"
set ylabel "Memory (MB)"
plot "memory.dat" title "" with lines
```

returns



Like time, memory consumption is linear in sequence length. With almost 2.5 GB for 20 Mb of sequence, it is quite large, though.

Answer 203 The command

```
awk -f suf.awk s.fasta | sort -k 2 | cat -n
```

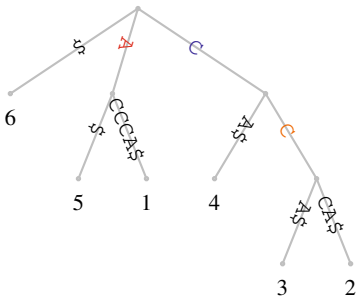
gives the same suffix array as in Fig. 3.6a, minus the header.

Answer 204 The root corresponds to sa[1..6], the node with path label `C` to sa[4..6], and the node with path label `CC` to sa[5..6].

Answer 205 Here are the common prefixes:

index	sa	suf	cp
1	6	$	nd
2	5	A$	–
3	1	ACCCA$	A
4	4	CA$	–
5	3	CCA$	C
6	2	CCCA$	CC

The color codes for the common prefixes are repeated in the suffix tree:



All three edges leading to an inner node of the suffix tree are labeled by a common prefix, because a suffix tree essentially summarizes the common prefixes of all suffixes.

Answer 206

index	sa	suf	cp	lcp
1	6	$	nd	-1
2	5	A$	–	0
3	1	ACCCA$	A	1
4	4	CA$	–	0
5	3	CCA$	C	1
6	2	CCCA$	CC	2

As the first entry in cp is undefined (there is no suffix "above" it), we give it a length less than the smallest entry in the lcp array; by convention -1 is used.

Answer 207 The remaining lcp intervals are as follows:

(e)

index	sa	lcp	2 1 0
1	6	-1	
2	5	0	
3	1	1	
4	4	0	
5	3	1	
6	2	2	
7	–	-1	

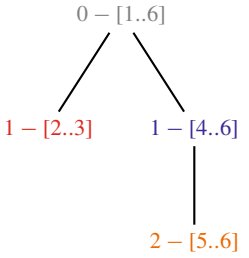
(f)

index	sa	lcp	2 1 0
1	6	-1	
2	5	0	
3	1	1	
4	4	0	
5	3	1	
6	2	2	
7	–	-1	

(g)

index	sa	lcp	2 1 0
1	6	-1	
2	5	0	
3	1	1	
4	4	0	
5	3	1	
6	2	2	
7	–	-1	

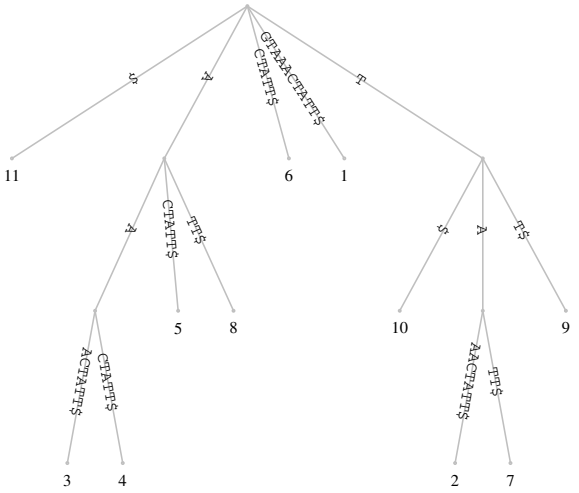
Answer 208



This lcp interval tree is the suffix tree in Fig. 3.6 stripped of its leaves.

Answer 209 On the left is the lcp interval tree, on the right the suffix tree.

i	suf	sa	lcp
1	$	11	-1
2	AAACTATT$	3	0
3	AACTATT$	4	2
4	ACTATT$	5	1
5	ATT$	8	1
6	CTATT$	6	0
7	GTAAACTATT$	1	0
8	T$	10	0
9	TAAACTATT$	2	1
10	TATT$	7	2
11	TT$	9	1
12	—	—	-1



Answer 210

isa	i	sa	suf	cp	lcp
3	1	6	$	-	-1
6	2	5	A$	-	0
5	3	1	ACCCA$	A	1
4	4	4	CA$	-	0
2	5	3	CCA$	C	1
1	6	2	CCCA$	CC	2

By following the arrows, convince yourself that sa[isa[1]] = 1, sa[isa[2]] = 2, and so on. Thus by traversing the inverse suffix array isa, the suffixes in the alphabetically sorted suffix array sa are visited in original order.

Answer 211

```
{
    n = n + 1       # position in the suffix array
    sa[n] = $1      # save the suffix array entries
    suf[n] = $2     # save the suffixes
    isa[sa[n]] = n # construct inverse suffix array
}END{
    printf "# i\tsa\tisa\tsuf\n" # print the output table
    for(i=1; i<=n; i++)
       printf "%d\t%d\t%d\t%s\n", i, sa[i], isa[i], suf[i]
}
```

Run it

```
awk -f suf.awk s.fasta | sort -k 2 | awk -f isa.awk
# i     sa      isa     suf
1       6       3       $
2       5       6       A$
3       1       5       ACCCA$
4       4       4       CA$
5       3       2       CCA$
6       2       1       CCCA$
```

Answer 212

```
{
    n++             # position in the suffix array
    sa[n] = $1      # save the suffix array
    suf[n] = $2     # save the suffixes
    if(sa[n] == 1) # find the input sequence
       s = $2
}END{
    # compute the isa
    for(i=1; i<=n; i++)
        isa[sa[i]] = i
    # compute the lcp-array
    lcp[1] = -1
    L = 0
    for(i=1; i<=n; i++){
        j = isa[i]
        if(j > 1){
           k = sa[j-1]
           while(substr(s, i+L, 1) == substr(s, k+L, 1))
              L++
           lcp[j] = L
           if(L > 1)
              L = L - 1
           else
              L = 0
        }
    }
    # print the output table
    printf "# i\tsa\tlcp\tsuf\n"
    for(i=1; i<=n; i++)
        printf "%d\t%d\t%d\t%s\n", i, sa[i], lcp[i], suf[i]
}
```

This can be run as

```
awk -f suf.awk s.fasta | sort -k 2 | awk -f esa.awk
# i     sa      lcp     suf
1       6       -1      $
2       5       0       A$
3       1       1       ACCCA$
4       4       0       CA$
5       3       1       CCA$
6       2       2       CCCA$
```

Answer 213

```
awk -f suf.awk dgAdhAdhdup.fasta |
sort -k 2                        |
awk -f esa.awk                   |
sed '/^#/d'                      |
sort -k 3 -n -r                  |
head -n 1
3325    988 12          TACATTACATTA...
```

The longest repeat has length 12, and it is found at positions 988 and 3325.

Answer 214

```
awk -f suf.awk dmAdhAdhdup.fasta |
sort -k 2                        |
awk -f esa.awk                   |
sed '/^#/d'                      |
sort -k 3 -n -r                  |
head -n 1
3890    3908    16        TCGATGTCCTGATCAA...
```

The longest repeat has length 16, and it is found at positions 3890 and 3908.

Answer 215

```
awk -f suf.awk dmDgAdhAdhdup.fasta  |
sort -k 2                           |
awk -f esa.awk                      |
sed '/^#/d'                         |
sort -k 3 -n -r                     |
head -n 1
1592    8048    37
      AGCAAGGTTCTCATGACCAAGAATATAGCGGTGAGTG...
```

The longest repeat for the concatenated sequences must be at least as long as the longer of the two repeats seen in the individual sequences, that is, it has to be at least 16 bp long. However, the two dehydrogenase sequences were taken from two *Drosophila* species; the relatedness between these species means there is much more sequence similarity between than within the alcohol sequences, leading to the much longer repeat of 37 bp.

Answer 216

Position	Shustring
1	AC
2	CCC
3	CCA
4	CA
5	nonexistent

The shortest shustrings are AC and CA with length 2. In the context of real sequences, shustrings could be used for the design of PCR primers, or in DNA-based identification of pathogens.

Answer 217 The enhanced suffix array for ACCCA is shown in the Solution to Problem 206—or you could generate it using `suf.awk` and `esa.awk`. To find the shustring for a given position, i, look at lcp[i] and the following entry, lcp[$i+1$]. Whichever is greater is the length of the longest repeat starting at sa[i]. This value plus one gives the shustring length at sa[i].

Answer 218

```
shustring -i mgGenome.fasta
#>gi|84626123|gb|L43967.2| Mycoplasma genitalium G37,
     complete genome 580076 35 6
#num  pos      len  seq
1     11911    6    CGAGGC
2     37969    6    GAGACG
3     60188    6    TCGGAC
...
```

So there are 35 unique motifs of length 6.

Answer 219 Including the reverse strand gives

```
shustring -r -i mgGenome.fasta
#>gi|84626123|gb|L43967.2| Mycoplasma genitalium G37,
     complete genome 580076 2 6
#num   pos       len  seq
1      174222    6    GACGGC
2      567107    6    GCCGGG
```

In other words, of the 35 shustrings found on the forward strand, 33 also occurred on the reverse strand, leaving only two shustrings of length 6 when scanning the whole genome.

Answer 220

```
shustring -l . -M 7 -r -i mgGenome.fasta |
head -n 1
#>L43967 L43967.1 Mycoplasma genitalium G37 complete genome.
    580074 254 6<=l<=7
```

So there 254 shustrings ≤ 7 bp long.

Answer 221

Rotations	Sort
TACTA$	$TACTA
ACTA$T	A$TACT
CTA$TA	ACTA$T
TA$TAC	CTA$TA
A$TACT	TA$TAC
$TACTA	TACTA$

The BWT is the last column of the sorted rotations:

```
ATTAC$
```

Make the directory for this session and change into it

```
mkdir Bwt
cd Bwt
```

Check the transform

```
bwt seq.fasta
```

where `seq.fasta` contains

```
>Seq
TACTA
```

Answer 222 We find long runs of identical letters, for example

```
...
115 LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLNNNNNN
116 NNNNNNNNNNNRHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
117 HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
118 HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
119 HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
120 HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
121 HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
122 HRRRRRRRRRRMMMMMMRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
...
```

Answer 223 The BWT is the last column of the sorted rotations. For decoding, we also need the first column of the rotation, which we obtain by sorting the transform:

Sort		Count		Reconstruct	
$\mathscr{F}$	$\mathscr{L}$	$\mathscr{F}$	$\mathscr{L}$	$\mathscr{F}$	$\mathscr{L}$
\$	C	$\$_1$	C_1	$\$_1$	C_1
C	C	C_1	C_2	C_1	C_2
C	C	C_2	C_3	C_2	C_3
C	T	C_3	T_1	C_3	T_1
G	\$	G_1	$\$_1$	G_1	$\$_1$
T	G	T_1	G_1	T_1	G_1

For the decoding, we trace back from $\$_1$ in $\mathscr{L}$ to $\$_1$ in $\mathscr{F}$ and note the nucleotides in $\mathscr{F}$ along that path, which returns `GTCCC`. To check this result, enter

```
bwt -d bwt.fasta
```

where `bwt.fasta` contains `CCCT$G`.

Answer 224 The key insight is that the last character in the rotation is just to the left of the start of a suffix. Hence, the BWT of a string is found by looking up $T[\text{sa}[i] - 1]$ for $i = 1, 2, ...$ The only exception comes when $\text{sa}[i]$ is 1: The character to the left of $T[1]$ by definition is the sentinel.

Answer 225 We write down our text with indexes:

1	2	3	4	5	6
T	A	C	T	A	$

Then we construct its suffix array:

i	sa	suf
1	6	$
2	5	A$
3	2	ACTA$
4	3	CTA$
5	4	TA$
6	1	TACTA$

Finally, write down $T[sa[i]-1]$ for $i = 1, ..., 6$ to get $\text{bwt}(T) =$ ATTAC$. Check the result

```
bwt seq.fasta
ATTAC$
```

where `seq.fasta` contains TACTA.

Answer 226

	0	1	2	3
Encoding	A	C	G	T
2	G	A	C	T
2,3	T	G	A	C
2,3,0	T	G	A	C
2,3,0,2	A	T	G	C
2,3,0,2,2	G	A	T	C

So the solution is 2,3,0,2,2. To check, enter

```
mtf mtf.fasta
```

where `mtf.fasta` contains GTTAG.

Answer 227

	0	1	2	3
Decoding	A	C	G	T
C	C	A	G	T
CC	C	A	G	T
CCA	A	C	G	T
CCAT	T	A	C	G
CCATG	G	T	A	C

So the solution is CCATG.

Answer 228 The decomposition is

```
T.A.C.TA
```

which we can verify using

```
lzd seq.fasta
T.A.C.TA
```

Answer 229 This microsatellite decomposes into two factors:

```
A.AAAA
```

In fact, all sequences of a single kind of nucleotide decompose into just two factors, regardless of their length.

Answer 230 The sequence `TACTA` contains 4 LZ factors, and hence its complexity is $4/5 = 0.8$, while `AAAA` decomposes into two factors, so its complexity is $2/5 = 0.4$.

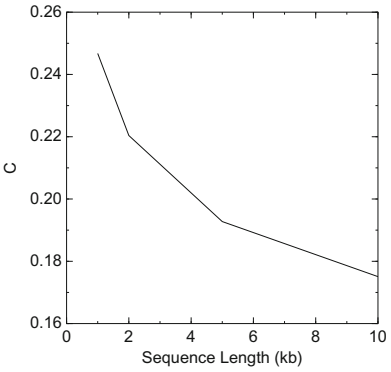
Answer 231 The smallest number of factors is 2, and hence $C \geq 2/|S|$. The largest number of factors is equal to the sequence length, so

$$2/|S| \leq C \leq 1.$$

Answer 232 We carry out the computations

```
for a in 1 2 5 10; do ranseq -l ${a}000 | lzd -n; done
# n   n/site
247   0.246753
# n   n/site
441   0.22039
# n   n/site
964   0.192761
# n   n/site
1751  0.175082
```

to get



We see that the maximum complexity, C, depends on the sequence length. So we can only sensibly compare C between sequences of the same length.

Answer 233 We use commands like

```
bwt mgGenome.fasta | lzd -n
```

to get

Operation	Number of Factors	C
None	60049	0.104
BWT	62688	0.108
BWT | MTF	45408	0.060
MTF	63495	0.109
MTF | BWT	65261	0.113

There is only one combination that results in any complexity reduction, BWT followed by MTF.

Answer 234 Measure the size of the original file

```
du -h mgGenome.fasta
576K
```

Then, measure the sizes of the transformed and compressed files to get

Operation	`gzip`	`bzip2`
nothing	172	160
`randomizeSeq`	176	164
BWT	176	164
BWT | MTF	124	116

As before, the combination of BWT and MTF leads to the greatest compressibility.

Answer 235 We begin as usual by creating a working directory:

```
mkdir KerrorAlignment
cd KerrorAlignment
```

Then copy the sequence

```
cp ../Data/dmAdhAdhdup.fasta .
```

and cut out the desired region

```
cutSeq -r 2301-2400 dmAdhAdhdup.fasta > dmAdhFrag.fasta
```

Check the result by confirming it matches its original position:

```
keywordMatcher -f dmAdhFrag.fasta dmAdhAdhdup.fasta
>DMADH X78384.1 ... :2301
```

Answer 236 Mutate the sequence:

```
mutator -p 10 dmAdhFrag.fasta > dmAdhFrag2.fasta
```

Check the mutation:

```
gal -i dmAdhFrag.fasta -j dmAdhFrag2.fasta
```

Confirm the exact match is gone:

```
keywordMatcher -f dmAdhFrag2.fasta dmAdhAdhdup.fasta
```

However,

```
lal -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

works as expected.

Answer 237 Locate the error-free copy

```
kerror -i dmAdhFrag.fasta -j dmAdhAdhdup.fasta
```

and the mutated copy with the fragments printed

```
kerror -L -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

Now search for the fragments; the first one is not found,

```
keywordMatcher -p
    GACCACCAACCTGCTGAAGACCATCTTCGCCCAGCTGAAGACCGTCGATG
    dmAdhAdhdup.fasta
```

but the second one is

```
keywordMatcher -p
    TCCTGATCAACGGAGCTGGTATCCTGGACGATCACCAGATCGAGCGCACC
    dmAdhAdhdup.fasta
```

Answer 238 The alignment produced by `kerror` is global in the query and local in the subject. Such "glocal" alignments are used to align sequencing reads to genomes.

Answer 239 Copy the chromosome files

```
cp ../Data/dmChr*.fasta .
```

Then list their sizes:

```
cchar -s dmChr*.fasta | grep '^>'
```

to get

```
>NT_033779.5 Drosophila melanogaster chromosome 2L 23513712
>NT_033778.4 Drosophila melanogaster chromosome 2R 25286936
>NT_037436.4 Drosophila melanogaster chromosome 3L 28110227
>NT_033777.3 Drosophila melanogaster chromosome 3R 32079331
>NC_004353.4 Drosophila melanogaster chromosome 4 1348131
>NC_024511.2 Drosophila melanogaster mitochondrion, complete
     genome 19524
>NC_004354.4 Drosophila melanogaster chromosome X 23542271
>NC_024512.1 Drosophila melanogaster chromosome Y 3667352
```

Compute the complete genome length:

```
cchar -s dmChr*.fasta   |
grep '^>'               |
cut -f 2                |
awk '{s+=$1}END{print s}'
137567484
```

The genome is approximately 138 Mb long.

Answer 240 Copy *Hamlet*

```
cp ../Data/hamlet.fasta .
```

and take a look at it

```
less hamlet.fasta
```

before counting its characters

```
cchar hamlet.fasta | head -n 1
# Total number of input characters: 136033
```

This means that the genome of *D. melanogaster* is approximately 1000 times longer than the text of *Hamlet*.

Answer 241 Compare the links and the original files using commands like

```
diff dmChr2L.fasta ../Data/dmChr2L.fasta
```

to find that content-wise they are identical. However, their sizes are vastly different: The command `ls -l` returns entries like

```
ls -l
lrwxrwxrwx 1 haubold haubold   27 Apr 17 16:01 dmChr2L.fasta
    -> ../Data/dmChr2L.fasta
```

which indicates the target of the symbolic link, `->`, and its size, 27 bytes. The original file, on the other hand, is almost a million times larger:

```
ls -l ../Data/dmChr2L.fasta
-rw-rw-r-- 1 haubold haubold 23807685 Apr 15 09:59
   ../Data/dmChr2L.fasta
```

Answer 242 Run a script like

```
for k in 1 2 5 10 20 50 100 200 500
do
    echo $k
    for a in 2L 2R 3L 3R 4 X Y Mt
    do
        kerror -k $k -i dmAdhAdhdup.fasta -j dmChr${a}.fasta
    done
done
```

It prints an alignment where *Adh*/*Adh-dup* is located on the left arm of chromosome 2 at positions 14,614,315–14,619,086; the alignment contains 161 errors, which is a surprisingly large number given that both sequences were sampled from the same species.

Answer 243 The number of errors per site is

$$\pi = \frac{161}{14,619,086 - 14,614,315 + 1} \approx 3.4\%.$$

Answer 244 Most errors are located toward the end of the alignment.

Answer 245 Cut out the *Adh*/*Adh-dup* locus:

```
cutSeq -r 14614315-14619086 dmChr2L.fasta > dmGenomic.fasta
```

Align the sequences

```
lal -i dmGenomic.fasta -j dmAdhAdhdup.fasta > aln.lal
```

This matches positions 1–4597 in the query and 1–4589 in the subject. Count the matches

```
grep '|' aln.lal |                # Extract match lines
sed 's/ *//g'    |                # Remove blanks
awk '{s+=length($1)}END{print s}' # Count match symbols
4541
```

This means that the mismatches per site is

$$\pi = \frac{4589 - 4541}{4589} \approx 1\%.$$

Answer 246 The command

```
kerror -k 161  -i dmAdhAdhdup.fasta -j dmChr2L.fasta
```

gives as time measurements on our computer

```
# Total time:        3.42s
# Data manipulation: 0.14s
# Matching:          0.49s
# Checking:          2.80s
```

This means that matching the $k + 1 = 162$ fragments is five times faster than checking whether or not they are part of a full alignment.

Answer 247 The *Adh*/*Adh-dup* region is already contained in `dmGenomic.fasta`. Align it to the *D. guanche* version:

```
gal -i dgAdhAdhdup.fasta -j dmGenomic.fasta > aln.gal
```

Count the matches

```
grep '|' aln.gal |
sed 's/ *//g'    |
awk '{s+=length($1)}END{print s}'
2960
```

Since `dmGenomic.fasta` is 4772 bp long, `kerror` needs to be run with $k = 4772 - 2960 = 1812$. In other words, `dgAdhAdhdup.fasta`, which is 4433 bp long, is divided into fragments of length $4433/(1812+1) \approx 2.45$. Since matches of this length are ubiquitous, the checking phase would take a very long time and make the search unfeasible in practice.

Answer 248 Create the directory for this session, change into it, and copy the input sequence:

```
mkdir FastLocalAlignment
cd FastLocalAlignment
cp ../Data/dmAdhAdhdup.fasta .
```

Cut out the fragment:

```
cutSeq -r 3101-3200 dmAdhAdhdup.fasta > dmAdhFrag.fasta
```

Align it to the original sequence:

```
sblast -i dmAdhFrag.fasta -j dmAdhAdhdup.fasta
# reading input data...done
# step1: generating word list from query...done
# step2: searching for exact matches of words in subject...done
# step3: extending exact matches...done
# qs   qe   ss   se   score
1      100  3101 3200 100.0
```

The result is exactly at the expected position. Repeat the alignment with the word list printed out:

```
sblast -L -i dmAdhFrag.fasta -j dmAdhAdhdup.fasta
```

There are 90 words, each 11 bp long.

Answer 249 Copy the fragment

```
cp dmAdhFrag.fasta dmAdhFrag2.fasta
```

Mutate it

```
bash mutate.sh
```

where `mutate.sh` is

```
for i in $(seq 1 11 100)
do
    mutator -p ${i} dmAdhFrag2.fasta > tmp
    mv tmp dmAdhFrag2.fasta
done
```

Then align it using `sblast`:

```
sblast -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

where no hit is found. This is because there is no word without a mismatch and hence no starting point for the extension step (Fig. 4.2c).

Answer 250 Any word length less than 11 will return an alignment, for example,

```
sblast -w 10 -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

Answer 251 The command

```
lal -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

returns the expected alignment. In other words, `lal` is more sensitive than `sblast` with default parameters.

Answer 252 Run

```
bash sensitivitySblast.sh > sensitivity1.dat
gnuplot -p sensitivity1.gp
```

where `sensitivitySblast.sh` is

```
for m in 0.01 0.02 0.05 0.1 0.2 0.5
do
    echo -n ${m} ' '
    for a in $(seq 100)
    do
        mutator -m ${m} dmAdhFrag.fasta > dmAdhFrag3.fasta
        sblast -i dmAdhFrag3.fasta -j dmAdhAdhdup.fasta
    done  |
        grep -A 1 qs | sed '/^#/d;/^-/d' | wc -l
done
```

Instead of the line

```
grep -A 1 qs | sed '/^#/d;/^-/d' | wc -l
```

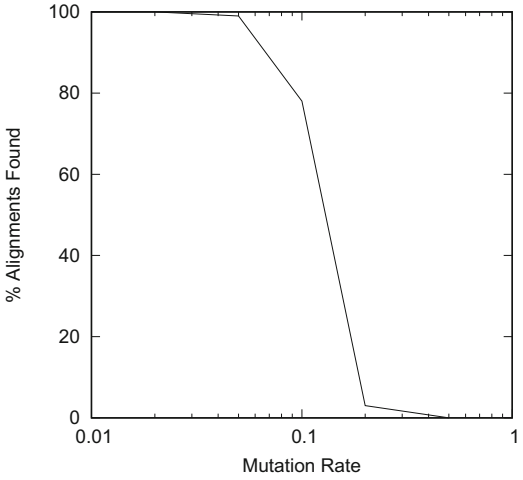
you might have used

```
grep -v ^# | wc -l
```

which is simpler and gives almost the same result. However, the more complex filter ensures that no more than one result is counted per `sblast` run. Compare the two solutions without `wc -l` to see the difference. The `gnuplot` script `sensitivity1.gp` contains

```
set logscale x
set xlabel "Mutation Rate"
set ylabel "% Alignments Found"
plot "sensitivity1.dat" title "" with lines
```

The resulting plot is



There is a sharp drop in sensitivity for queries mutated at more than 20% of their positions.

Answer 253 In `sensitivitySblast.sh` change the line

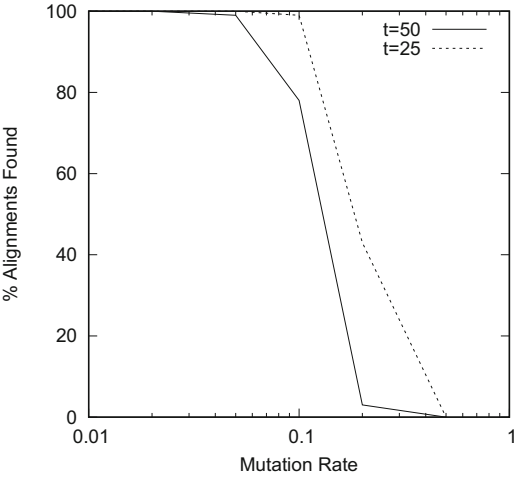
```
sblast -i dmAdhFrag3.fasta -j dmAdhAdhdup.fasta
```

to

```
sblast -t 25 -i dmAdhFrag3.fasta -j dmAdhAdhdup.fasta
```

Save the simulation results in `sensitivity2.dat` and plot

```
gnuplot -p sensitivity2.gp
```



where `sensitivity2.gp` is

```
set xlabel "Mutation Rate"
set ylabel "% Alignments Found"
set logscale x
plot "sensitivity1.dat" title "t=50" with lines,\
"sensitivity2.dat" title "t=25" with lines
```

to see that the sensitivity of `sblast` is increased if the minimum score is halved from 50 to 25.

Answer 254

```
sblast -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta |
sed '/^#/d'                                      |
sort -k 5 -n -r
2299    2594    2259    2554    164.0
3865    4028    3657    3820    76.0
3225    3328    3220    3323    68.0
2182    2285    2142    2245    64.0
```

The optimal local alignment is found by

```
lal -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
```

It has a score of 217, which is much larger than the score of the best alignment found by `sblast` (164).

Answer 255 The command we are looking for is

```
sblast -s 40 -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
# reading input data...done
# step1: generating word list from query...done
# step2: searching for exact matches of words in subject...done
# step3: extending exact matches...done
# qs    qe      ss      se      score
3829    4028    3621    3820    80.0
3225    3328    3220    3323    68.0
2182    2594    2142    2554    217.0
```

Now the best alignment has the same coordinates and the same score as the best alignment returned by `lal`. Heuristics like the extension parameter can influence the result.

Answer 256 Run

```
bash driveSblastDm.sh dmChr*.fasta
```

where `driveSblastDm.sh` contains

```
for a in $@
do
    echo Searching ${a}
    sblast -i dmAdhAdhdup.fasta -j $a |
        sed '/^#/d'
done
```

to see—as we have already done with `kerror`—that in *D. melanogaster* the *Adh*/*Adh-dup* region is located on the left arm of chromosome 2. To find the exact interval, run

```
bash driveSblastDm.sh dmChr*.fasta |
grep ^[0-9]                        |
sort -n -k 3
```

to locate *Adh*/*Adh-dup* in the interval 14,614,315–14,619,393.

Answer 257

```
time kerror -k 161 -i dmAdhAdhdup.fasta -j dmChr2L.fasta
```

takes 3.79 s, while

```
time sblast -i dmAdhAdhdup.fasta -j dmChr2L.fasta
```

takes only 1.37 s. The program `sblast` is faster than `kerror`, because `kerror` uses dynamic programming for finding the alignment (Fig. 4.1), while `sblast` uses the simpler extension strategy (Fig. 4.2). Apart from speed, `sblast` has the advantage that there is no need for guessing a suitable number of errors, k.

Answer 258 Run

```
sblast -i dgAdhAdhdup.fasta -j dmChr2L.fasta |
grep ^[0-9]                                  |
sort -n -k 3
```

to find homology in the interval 14,616,500–14,618,356. This shows that even ungapped alignment can be a highly effective tool: it is faster and more sensitive than `kerror`.

Answer 259 The command

```
blastn -query dmAdhFrag2.fasta -subject dmAdhAdhdup.fasta
```

returns no hit. However, with the appropriate `-word_size` the expected hit is found, plus three more of low significance:

```
blastn -query dmAdhFrag2.fasta -subject dmAdhAdhdup.fasta -
    word_size 10
```

Answer 260 Our manual binary search resulted in the following left and right borders (l & r), where the middle, $m = (l + r)/2$:

Step	l	r	m
1	1	100	51
2	1	51	26
3	26	51	39
4	26	39	33
5	26	33	30
5	26	30	28
6	28	30	29

Since we got a hit with a distance of 29 between mutations, but not with a distance of 28, the default value of `-word_size` is 28.

Answer 261 Here are the steps of the binary search:

Step	l	r	m
1	1	100	51
2	1	51	26
3	1	26	14
4	1	14	8
5	8	14	11
5	11	14	13
6	11	13	12

Since we got a hit with a distance of 12 between mutations, but not with a distance of 11, the default `-word_size` is now 11.

Answer 262 To explore the sensitivity in blastn mode, we simulate

```
bash sensitivityBlastn.sh > sensitivityN.dat
```

where `sensitivityBlastn.sh` is

```
for m in 0.01 0.02 0.05 0.1 0.2 0.5
 do
    echo -n ${m} ' '
    for i in $(seq 100)
    do
        mutator -m ${m} dmAdhFrag.fasta > dmAdhFrag3.fasta
        blastn -outfmt 7 -task blastn -query dmAdhFrag3.
           fasta -subject dmAdhAdhdup.fasta
    done  |
        grep -A 1 hits | sed '/^#/d;/^-/d' | wc -l
done
```

For the megablast mode, just leave out

```
-task blastn
```

from the `blastn` command and run

```
bash sensitivityMega.sh > sensitivityM.dat
```

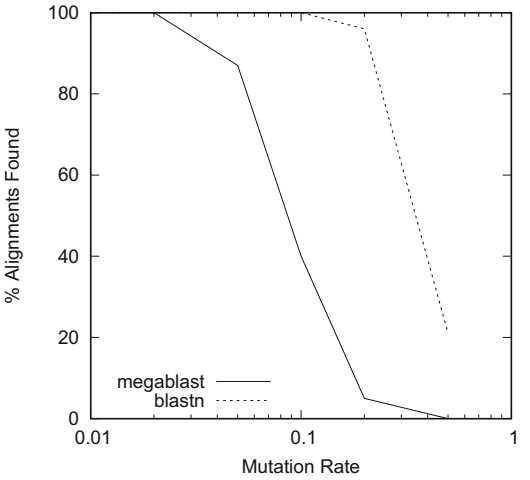
Plot the two result sets

```
gnuplot -p sensitivity3.gp
```

where `sensitivity3.gp` is

```
set xlabel "Mutation Rate"
set ylabel "% Alignments Found"
set logscale x
set key bottom left
plot "sensitivityM.dat" title "megablast" with lines,\
"sensitivityN.dat" title "blastn" with lines
```

We moved the key from the top right to the bottom left—otherwise it would intersect the graph in our printout; the blastn mode is much more sensitive than the megablast mode.



Answer 263 All that needs to be changed is the match score in `lal`:

```
lal -A 2 -i dmAdhFrag2.fasta -j dmAdhAdhdup.fasta
```

This returns the same alignment as `blastn` with the same score of 156.

Answer 264 The command

```
blastn -reward 1 -task blastn -query dmAdhFrag2.fasta
   -subject dmAdhAdhdup.fasta
```

returns the match line

```
Score = 131 bits (66),  Expect = 1e-34
```

in which the bit score has changed only from previously 141 to 131, but the raw score has been more than halved from 156 to 66.

Answer 265 The best alignment with the BLAST command

```
blastn -task blastn -subject dmAdhAdhdup.fasta -query
   dgAdhAdhdup.fasta
```

has a raw score of 842. In contrast, the best alignment with `lal`,

```
lal -A 2 -i dmAdhAdhdup.fasta -j dgAdhAdhdup.fasta
```

has the much higher score of 1088.

Answer 266 A bit of trial and error gave 333 as the smallest xdrop value compatible with the alignment returned by `lal`:

```
blastn -task blastn -query dmAdhAdhdup.fasta -subject
   dgAdhAdhdup.fasta -xdrop_gap_final 333
```

Answer 267 `Blastn` cannot be used to find a better alignment than `lal`. Remember, optimal alignment methods are guaranteed to return the best result given a score scheme.

Answer 268 Cut out the desired region

```
cutSeq -r 3101-3200 dgAdhAdhdup.fasta > dgAdhFrag.fasta

blastn -task blastn -query dgAdhFrag.fasta -subject
   dmAdhAdhdup.fasta
```

gives an alignment with raw score 22. However, the corresponding `lal` command

```
lal -A 2 -i dgAdhFrag.fasta -j dmAdhAdhdup.fasta
```

gives a better alignment with score 26. With a bit of trial and error, we found 7 is the largest word size compatible with the optimal alignment:

```
blastn -word_size 7 -task blastn -query dgAdhFrag.fasta
   -subject dmAdhAdhdup.fasta
```

Answer 269

$$P = 1 - e^{-0.015} \approx 0.015,$$

which illustrates that $P \leq E$, and also that for small values of E the two statistics are quite similar. Beware, however, P is a probability and hence bounded by 0 and 1, while E is an expectation value with lower bound 0 but no obvious upper bound.

Answer 270 The desired parameters are as follows:

Name	Value
s	413076
λ	0.625
K	0.410

We can use AWK to calculate

```
BEGIN{
    s = 413076
    l = 0.625
    K = 0.41
    x = 26
    y = K*s*exp(-l*x)
    print 1-exp(-y)
}
```

and get $P \approx 0.015$, which is exactly the value implied by the E-value returned.

Answer 271 The script

```
time bash simPval.sh > simPval.dat
```

where `simPval.sh` contains

```
for a in $(seq 1000)
do
    randomizeSeq dmAdhAdhdup.fasta  |    # randomize subject
       lal -A 2 -i dgAdhFrag.fasta  |    # alignment
       grep '^Sc'                   |    # extract score line
       sed 's/Score: *//'                # extract score
done
```

takes 98 s to execute. So if we decided to run the simulation for 10^4 iterations, we would have to wait approximately 17 min.

Answer 272 Count the results

```
awk -f count.awk simPval.dat
```

where `count.awk` is

```
{
    arr[$1]++
    c++
}END{
    for(a in arr)
        print a "\t" arr[a]/c
}
```

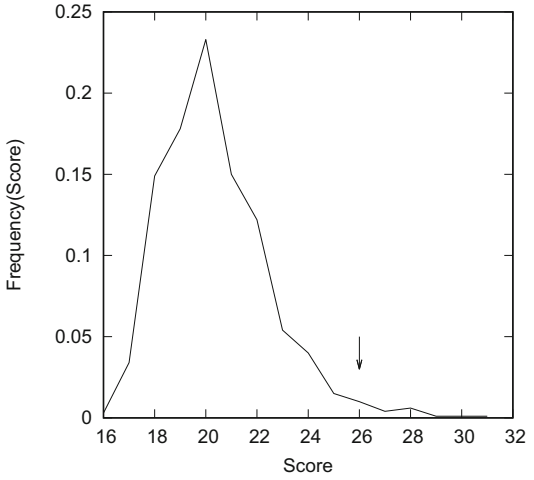
and plot them

```
awk -f count.awk simPval.dat |
sort -n                      |
gnuplot -p simPval1.gp
```

where `simPval1.gp` is

```
set xlabel "Score"
set ylabel "Frequency(Score)"
set arrow from 26,0.05 to 26,0.03
plot "< cat" title "" with lines
```

to get the distribution of random scores



where the arrow points to the observed score.

Answer 273 The P value is

```
awk '{if($1>=26)s++;c++}END{print s/c}' simPval.dat
0.022
```

Your result is bound to differ slightly. However, the simulated P value should always be at least similar to the theoretical $P = 0.015$.

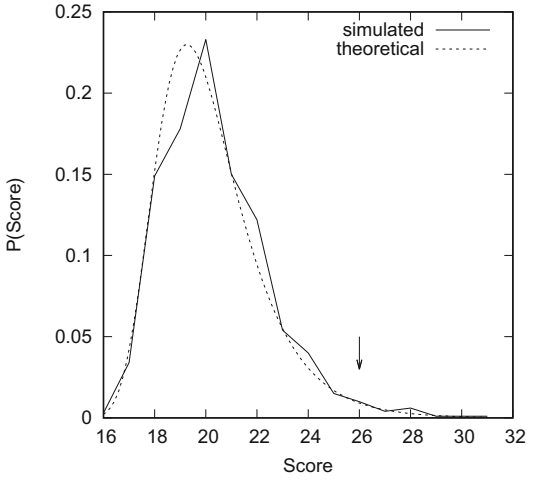
Answer 274 Plot

```
awk -f count.awk simPval.dat |
sort -n                      |
gnuplot -p simPval2.gp
```

where `simPval2.gp` is

```
s=413076
l=0.625
K=0.410
mu=log(K*s)/l
f(x)=l*exp((mu-x)*l-exp((mu-x)*l))
set xlabel "Score"
set ylabel "P(Score)"
set arrow from 26,0.05 to 26,0.03
plot "< cat" title "simulated" with lines,\
f(x) title "theoretical" with lines
```

to get



Again, the arrow points to the observed score. Notice that we now interpret the frequency of a score as its probability, and hence the label $P(\text{Score})$ along the y-axis. The significance of the observed score of 26 (its P-value) is the proportion of random scores ≥ 26 obtained either from the theoretical or the simulated curve. Since the two curves agree well, and the simulations are time-consuming, calculating the E-value directly is by far the better option.

Answer 275 The search

```
time bash blast.sh dmChr*.fasta
```

where `blast.sh` is

```
for a in $@
do
    echo $a
    blastn -task blastn -query dmAdhAdhdup.fasta -subject $a
         -evalue 1e-20 -outfmt 7
done
```

takes approximately 5 s and returns two alignments covering positions 14,614,315–14,618,911 and 14,619,298–14,619,405.

Answer 276 Database construction takes approximately 2 s. The search based on this database

```
time blastn -task blastn -outfmt 7 -query dmAdhAdhdup.fasta
    -db dmDb -evalue 1e-20
```

takes 0.7 s. So the combined run time of database construction plus search is 2.7 s, which is less than the 5 s it took to search the subject files in plain FASTA format.

Answer 277 The run

```
time blastn -outfmt 7 -query dmAdhAdhdup.fasta -db dmDb
    -evalue 1e-20
```

takes 0.15 s, so it is approximately five times faster than the blastn mode. The two alignments found have the same start and end coordinates as those found in blastn mode. This is because we are carrying out a sequence comparison within a species, that is, between very similar sequences, for which megablast is optimized. However, the alignments themselves do differ slightly; for example, in the first alignment we found:

Parameter	blastn	megablast
alignment length	4601	4603
mismatches	44	39
gap opens	10	15
bit score	7997	8155

Answer 278 Now the results do differ. In blastn mode, we get three alignments with a combined length of $169 + 173 + 37 = 379$ bp. The 169 bp alignment is the left-most hit, starting at position 14,616,354; the 173 bp alignment is the right-most hit, ending at position 14,618,839. Taken together, these three alignments cover the 2485 bp interval 14,616,354–14,618,839. In contrast, the megablast mode returns a single alignment of length 134 located at 14,617,517–14,617,650. Notice also the difference in E-values: In blastn mode, these are 0, 3×10^{-159}, and 10^{-37}; in megablast mode, the single alignment has $E = 3 \times 10^{-33}$.

Answer 279 Here is the overlap between the two example sequences:

```
ACCGTTC----
---GTTCAGTA
```

Answer 280 Create the directory, change into it, and generate the sequence files:

```
mkdir Shotgun
cd Shotgun
echo -e '>S1\nACCGTTC'  > s1.fasta
echo -e '>S2\nGTTCAGTA' > s2.fasta
```

Now compute the overlap alignment

```
oal -i s1.fasta -j s2.fasta

Query:              >s1
   Length:          7
Subject:            >s2
   Length:          8
Score:              4.0
Strand:             Plus / Plus


Query: 1 ACCGTTC---- 7
           ||||
Sbjct: 1 ---GTTCAGTA 8
//
```

Answer 281 You might be tempted to think that four comparisons are necessary: forward/forward, forward/reverse, reverse/forward, and reverse/reverse. But forward/forward is equivalent to reverse/reverse and forward/reverse is equivalent to reverse/forward, so only two comparisons are necessary.

Answer 282 Create the forward files

```
for a in $(seq 3); do cp f${a}.fasta f${a}f.fasta; done
```

Create the reverse files

```
for a in $(seq 3); do revComp f${a}.fasta > f${a}r.fasta; done
```

Then carry out the comparisons like

```
oal -i f1f.fasta -j f2f.fasta | grep Score
```

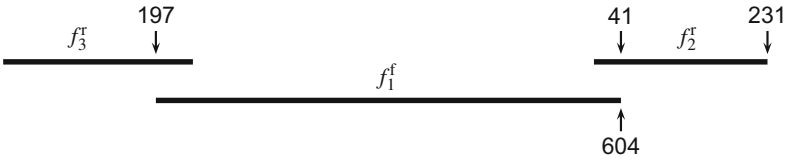
to find the following scores

	f_1^{f}	f_1^{r}	f_2^{f}	f_2^{r}	f_3^{f}	f_3^{r}
f_1^{f}			2	16	0	30
f_1^{r}						
f_2^{f}					1	0
f_2^{r}						
f_3^{f}						
f_3^{r}						

The two most substantial overlaps are between f1 and f2, and between f1 and f3. This means f1 bridges f2 and f3.

Answer 283 There are several ways of solving this, here is ours: Begin with `f1f.fasta` and `f2r.fasta` and get an overlap alignment like this

where 41 is the last overlapping position in f_2^{r}. Then, compare `f1f.fasta` and `f3r.fasta`, and add the next overlap, where 197 is the first overlapping position in f_3^{r}:



Now we can calculate the length of the underlying genomic region as

$$196 + 604 + 190 = 1000.$$

The reads were drawn from a 1 kb fragment.

Answer 284 Compute the genome size of *M. genitalium*:

```
cchar mgGenome.fasta

# Total number of input characters: 580076
# Char  Count   Fraction
A       200544  0.345720
C       91515   0.157764
G       92306   0.159127
T       195711  0.337389
```

The genome size of *M. genitalium* is 580076 and its GC content $0.157764 + 0.159127 \approx 0.317$.

Answer 285 Generate a random genome:

```
ranseq -s 35 -l 580076 -g 0.317 > ranGenome.fasta
```

and check the result:

```
cchar ranGenome.fasta
# Total number of input characters: 580076
# Char  Count   Fraction
A       197692  0.340804
C       91711   0.158102
G       92097   0.158767
T       198576  0.342328
```

Answer 286 The program `randomizeSeq` shuffles an existing sequence, which keeps its composition unchanged. In contrast, `ranseq` generates a new sequence, the composition of which is bound to vary between runs. You can verify this by piping repeated runs of `randomizeSeq` through `cchar` to find that the result is always the same, while the composition of each run on `ranseq` differs slightly (unless you fix the seed for the random number generator, of course).

Answer 287

$$10 \times 580076 = 5,800,760$$

Answer 288 We need the probability that a nucleotide is not sequenced times the length of the template:

$$580076 \times e^{-10} \approx 26.$$

That is, the expected combined length of all gaps in the assembly is 26.

Answer 289 By solving

$$L \times e^{-c} = 1$$

for c we find

$$c = -\ln\left(\frac{1}{L}\right).$$

In our case, the theoretical coverage $c \approx 13.3$.

Answer 290 Again, we write for the desired coverage

$$c = -\ln\left(\frac{0}{L}\right),$$

but since $\ln(0) = -\infty$, $c = \infty$ in this case. In other words, a combined gap length of 0 cannot be achieved. That is one reason why shotgun sequencing projects usually end with a few gaps that need to be closed by other laboratory methods.

Answer 291 Run `sequencer`:

```
sequencer -s 35 -c 13.3 ranGenome.fasta > reads.fasta
```

Again, we set a seed (`-s`) for the random number generator of `sequencer` to be able to exactly reproduce our result. The number of reads we have just generated is

```
grep -c '^>' reads.fasta

77161
```

and the number of nucleotides

```
cchar reads.fasta

# Total number of input characters: 7715050
# Char  Count   Fraction
A       2633810 0.341386
C       1223688 0.158611
G       1223629 0.158603
T       2633923 0.341401
```

Answer 292 Run the hashing program `velveth`:

```
velveth Assem/ 21 -short -fasta reads.fasta
```

Answer 293 First run the assembly on the hashed reads stored in the directory `Assem`:

```
velvetg Assem/ -exp_cov 13.3
```

Then count the contigs

```
grep -c '^>' Assem/contigs.fa

105
```

Ideally, we would get a single contig, but ending up with multiple contigs is the usual outcome of shotgun sequencing. Next, we count the nucleotides in the contigs:

```
cchar Assem/contigs.fa

# Total number of input characters: 583465
# Char  Count   Fraction
A       198953  0.340985
C       92269   0.158140
G       92672   0.158830
T       199567  0.342038
c       1       0.000002
g       1       0.000002
t       2       0.000003
```

We get an assembly that is $583,465 - 580,076 = 3,389$ nucleotides longer than the template, the length of which would of course be unknown in a real sequencing experiment. Four of those nucleotides are set in lower case to indicate inferior quality.

Answer 294

```
sequencer -P -s 35 -c 13.3 ranGenome.fasta > reads.fasta
velveth Assem/ 21 -shortPaired -fasta reads.fasta
velvetg Assem/ -exp_cov 13.3 -ins_length 500
grep -c '^>' Assem/contigs.fa
69
cchar Assem/contigs.fa
582931
# Char  Count   Fraction
A       198520  0.340555
C       92144   0.158070
G       92533   0.158737
N       236     0.000405
T       199454  0.342157
a       17      0.000029
c       13      0.000022
g       5       0.000009
t       9       0.000015
```

Paired-end sequencing gives fewer contigs than single-end sequencing even when applied to our idealized random genome. The best possible outcome would be to get a single contig that is identical to the input sequence.

Answer 295 Eliminate the sequencing error

```
sequencer -E 0 -P -s 35 -c 13.3 ranGenome.fasta > reads.fasta
velveth Assem/ 21 -shortPaired -fasta reads.fasta
velvetg Assem/ -exp_cov 13.3 -ins_length 500
grep -c '^>' Assem/contigs.fa
1
cchar Assem/contigs.fa
# Total number of input characters: 580135
# Char  Count   Fraction
A       198554  0.342255
C       92092   0.158742
G       91706   0.158077
N       109     0.000188
T       197672  0.340734
a       1       0.000002
t       1       0.000002
```

Without errors, we get a single contig with very few unknown nucleotides.

Answer 296 Simulate with 1% error:

```
sequencer -E 0.01 -P -s 35 -c 13.3 ranGenome.fasta > reads.
      fasta
velveth Assem/ 21 -shortPaired -fasta reads.fasta
velvetg Assem/ -exp_cov 13.3 -ins_length 500
grep -c '^>' Assem/contigs.fa
```

```
457
cchar Assem/contigs.fa
# Total number of input characters: 607802
# Char  Count   Fraction
A       207343  0.341136
C       96454   0.158693
G       96029   0.157994
N       1361    0.002239
T       206519  0.339780
a       31      0.000051
c       15      0.000025
g       19      0.000031
t       31      0.000051
```

This time there are 457 contigs. Notice also the many $(607,802 - 580,076 = 27,726)$ superfluous nucleotides in our assembly.

Answer 297 To *in silico* shotgun sequence the genome of *M. genitalium* and assemble it, run

```
sequencer -s 35 -c 13.3 mgGenome.fasta > reads.fasta
velveth Assem/ 21 -short -fasta reads.fasta
velvetg Assem/ -exp_cov 13.3
grep -c '^>' Assem/contigs.fa
344
cchar Assem/contigs.fa
# Total number of input characters: 573437
# Char  Count   Fraction
A       197561  0.344521
C       89775   0.156556
G       91005   0.158701
N       10      0.000017
T       195030  0.340107
a       21      0.000037
c       10      0.000017
g       8       0.000014
t       17      0.000030
```

There are 344 contigs.

Answer 298 The midpoint, or median, of $\{2, 2, 3, 4, 5\}$ is 3; the mean is $(2+2+3+4+5)/5 = 16/5 = 3.2$.

Answer 299 The total contig length is $2+2+3+4+5 = 16$. Now walk along $\mathscr{L}$ from right to left, until the cumulative length covered is at least 8; the element reached then is the N_{50}, in our case 4. The connection to the median is as follows: Rewrite $\mathscr{L}$ as $\mathscr{L}'$ such that each element x is repeated x times:

$$\mathscr{L}' = \{2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5\}$$

The N_{50} of $\mathscr{L}$ is the median of $\mathscr{L}'$, that is, the average of the 8-th and the 9-th element of $\mathscr{L}'$, which is 4, as expected.

Answer 300 The last line of `velvetg` output is

```
Final graph has 866 nodes and n50 of 25857, max 73815, total
   570552, using 77021/77157 reads
```

that is, $N_{50} = 25857$.

Answer 301 The reverse-sorted contigs are generated using

```
cchar -s Assem/contigs.fa   |
grep '^>'                   |
awk '{print $2}'            |
sort -r -n
```

The median contig length is found by extending the pipeline to look up the midpoint of these sorted lengths:

```
head -n 172 |
tail -n 1
47
```

This is quite different from $N_{50} = 25857$ and illustrates that the relationship between median and N_{50} is indirect, as explained in the Answer to Problem 299.

Answer 302 The program `n50.awk` is

```
{
    s[n++] = $1
    c += $1
}
END{
    while(sum < c/2)
        sum += s[i++]
    print "N_50: " s[i-1]
}
```

The complete pipeline now looks like this:

```
cchar -s Assem/contigs.fa |
grep '^>'                 |
awk '{print $2}'          |
sort -r -n                |
awk -f n50.awk
N_50: 25877
```

Our N_{50} differs from that returned by `velvetg` by $25877 - 25857 = 20$ nucleotides. We do not know why that is the case, but notice that the length quoted in the header of a contig is 20 less than the length of the actual contig:

```
cchar -s Assem/contigs.fa |
head -n 1

>NODE_1_length_18331_cov_10.140527: 18351
```

Answer 303

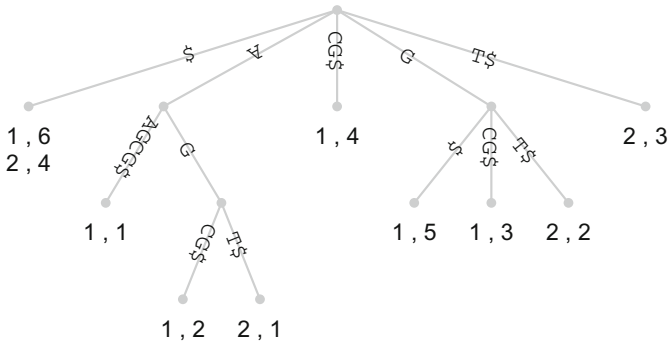
```
sequencer -P -s 35 -c 13.3 mgGenome.fasta > reads.fasta
velveth Assem/ 21 -shortPaired -fasta reads.fasta
velvetg Assem/ -exp_cov 13.3 -ins_length 500
cchar -s Assem/contigs.fa |
grep '^>'                 |
awk '{print $2}'          |
sort -r -n                |
awk -f n50.awk
N_50: 81536
```

Paired-end sequencing results in much longer contigs (81536) compared to single-end reads (25857). Do not forget to save this assembly

```
cp Assem/contigs.fa mgAssembly.fasta
```

Answer 304 Here is our generalized suffix tree:

	1	2	3	4	5	6
S_1	A	A	G	C	G	$
S_2	A	G	T	$		



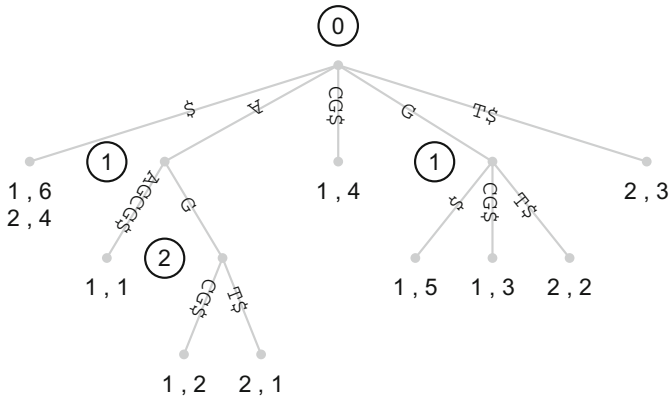
Answer 305 Suppose the sequence data is contained in `st.fasta`. Generate the corresponding generalized suffix tree:

```
drawStrees -i st.fasta -o st
latex st_fig.tex
dvips st_fig.dvi -o
gv st_fig.ps &
```

The result was already shown in the solution to Problem 304.

Answer 306 Mark the string depths (that of the root is always 0):

	1	2	3	4	5	6
S_1	A	A	G	C	G	$
S_2	A	G	T	$		



So the longest repeat in the input sequences is AG, which also happens to be the longest repeat *between* S_1 and S_2.

Answer 307

- Length of assembly:

```
cchar mgAssembly.fasta

# Total number of input characters: 577555
# Char  Count   Fraction
A       198323  0.343384
C       89287   0.154595
G       91923   0.159159
N       2774    0.004803
T       194480  0.336730
a       223     0.000386
c       184     0.000319
g       145     0.000251
t       216     0.000374
```

- Length of original sequence:

```
cchar mgGenome.fasta

# Total number of input characters: 580076
# Char  Count   Fraction
A       200544  0.345720
C       91515   0.157764
G       92306   0.159127
T       195711  0.337389
```

The assembly is 2521 nucleotides shorter than the input genome. It also contains 2774 unknown nucleotides (N) and $223 + 184 + 145 + 216 = 768$ low-quality nucleotides shown in lower case.

Answer 308 Generate the starting sequence

```
ranseq -l 1000 > s1.fasta
```

Cut out the first and last 100 bp

```
cutSeq -s -r 1-100,901-1000 s1.fasta > s2.fasta
```

Run `mummer`

```
mummer s1.fasta s2.fasta | sed '/^#/d'
...
> Rand_1;
       1         1       100
     900       100       101
```

Apart from the messages generated by `mummer`, the output consists of two lines with entries of the form (x, y, length). Notice that the second hit was extended by one nucleotide; however, solutions will differ as `ranseq` generates a new sequence every time it is run.

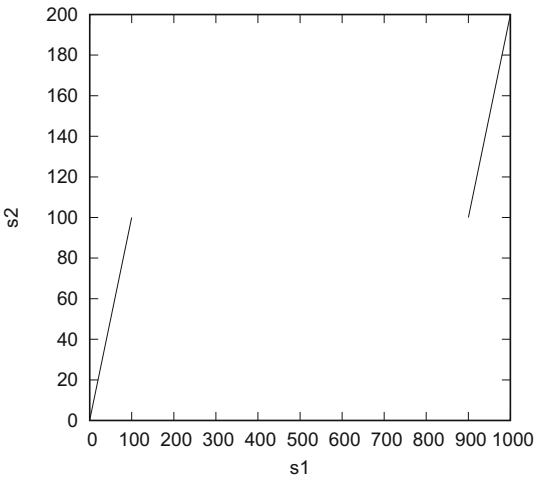
Answer 309 Align and plot

```
mummer s1.fasta s2.fasta |
awk -f mum2plot.awk      |
gnuplot -p mum.gp
```

where `mum.gp` is

```
set xlabel "s1"
set ylabel "s2"
plot "< cat" title "" with lines
```

to get



So the sequence in the first file—called reference file in the `mummer` interface—is written along the x-axis, the sequence in the second file—called query file—is written along the y-axis.

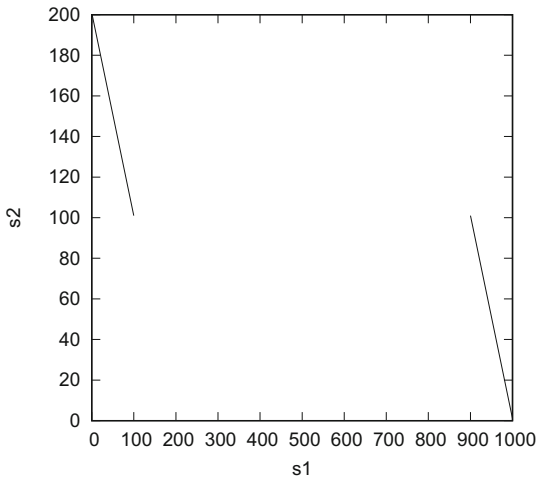
Answer 310 Reverse complement `s2.fasta`

```
revComp s2.fasta > s3.fasta
```

and pipe the results of `mummer` into a plot

```
mummer -b -c s1.fasta s3.fasta  |
awk -f mum2plot.awk             |
gnuplot -p mum.gp
```

and plot the `mummer` result



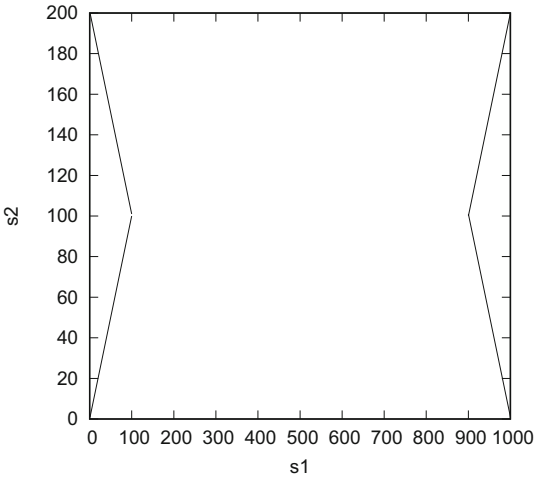
Answer 311 Concatenate `s2.fasta` and `s3.fasta`:

```
cat s2.fasta s3.fasta > s4.fasta
```

Compare the sequences

```
mummer -b -c s1.fasta s4.fasta |
awk -f mum2plot.awk            |
gnuplot -p mum.gp
```

to get



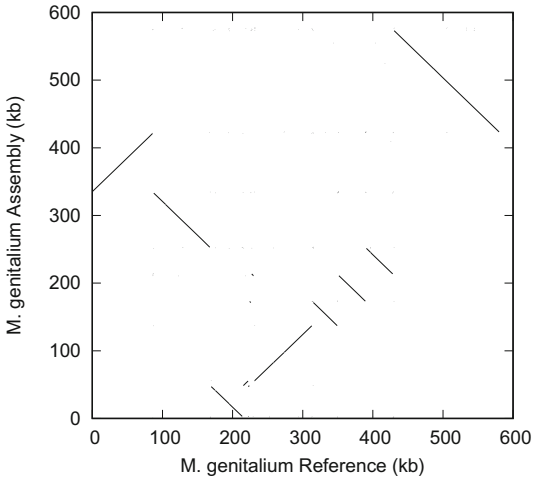
Answer 312 The pipeline

```
mummer -c -b mgGenome.fasta mgAssemblyS.fasta |
awk -f mum2plot.awk                           |
gnuplot -p mg.gp
```

where `mg.gp` is

```
set xlabel "M. genitalium Reference (kb)"
set ylabel "M. genitalium Assembly (kb)"
plot "< cat" using ($1/1000):($2/1000) title "" with lines
```

gives



Since we are comparing a known template to its assembly from simulated reads, one might have expected the assembly to be approximately identical to the template. This would have resulted in one line along the main diagonal of the dot plot matrix. However, due to the

stochastic nature of shotgun sequencing experiments, all we can expect is that the assembly covers most of the template, which is in fact what we observe. Notice also the transformation of the coordinates to kb, which was achieved with the `gnuplot` command

```
using ($1/1000):($2/1000)
```

Answer 313 Use the command `cchar` to find

Strain	Genome length
K12	4,639,675
O157H7	5,528,445

So the genomes of two strains from the same bacterial "species" can differ by $5.5/4.6 \times 100 \approx 20\%$ in length and hence in gene content.

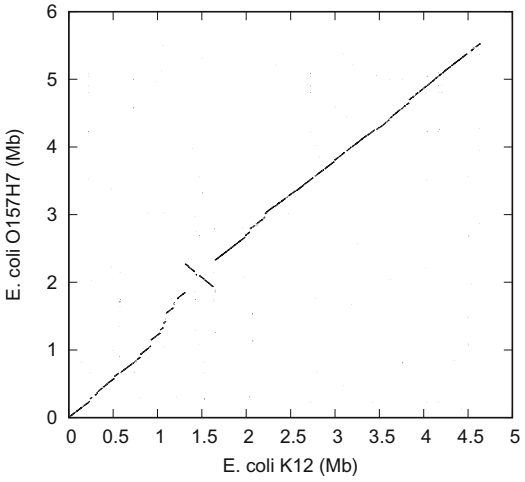
Answer 314 The pipeline

```
mummer -c -b ecoliK12.fasta ecoliO157H7.fasta |
awk -f mum2plot.awk                           |
gnuplot -p eco.gp
```

where `eco.gp` is

```
set xlabel "E. coli K12 (Mb)"
set ylabel "E. coli O157H7 (Mb)"
plot "< cat" using ($1/1000000):($2/1000000) title "" with
    lines
```

gives



This dot plot shows a large inversion around 1.5 Mb in K12 and 2 Mb in O157H7.

Answer 315 We count the SNPs by counting the entries in the SNP table:

```
tail -n +6 nucmer.snps | wc -l
86321
```

The genome length of K12 is 4,639,675 bp, so the number of pairwise differences per site is

$$\pi = \frac{86,321}{4,639,675} \approx 0.019$$

Answer 316 To compute the divergence time in generations, notice that the pairwise mismatches, π, have accumulated along two diverging lines of descent. Hence, the sought number of generations is

$$g = \frac{0.019/2}{2.2 \times 10^{-10}} \approx 43.2 \times 10^6$$

Answer 317 The lower bound is

$$\frac{g \times 30}{60 \times 24 \times 365.25} \approx 2463 \text{ years},$$

the upper

$$\frac{g \times 90}{60 \times 24 \times 365.25} \approx 7389 \text{ years}.$$

Answer 318 Sequence the template

```
sequencer -s 10 -c 15 dmChr2L.fasta > reads.fasta
```

Count the reads

```
grep -c '>' reads.fasta
3527065
```

Count the nucleotides sequenced

```
cchar reads.fasta
# Total number of input characters: 352705721
```

and the template length

```
cchar dmChr2L.fasta
# Total number of input characters: 23513712
```

Now calculate the exact coverage

```
bc -l
352705721 / 23513712
15.000001743663441995O3
```

This is very close to the expected coverage of 15.

Answer 319 Count the reads shorter than 100 bp:

```
sed '/^>/d' reads.fasta           |
awk '{if(length($1) < 100)print}' |
wc -l
12
```

Reads shorter than 100 bp are typically sampled from the edges of the template.

Answer 320 Run

```
bash runBlast.sh > blastTimes.dat
```

where `runBlast.sh` is

```
for a in 1 2 5 10 20 50 100 200 500 1000
do
    echo -n $a
    ((x=$a*2))
    head -n ${x} reads.fasta |
    /usr/bin/time -p blastn -task blastn-short -subject
          dmChr2L.fasta 2>&1 |
    grep real                |
    sed 's/real//'
done
```

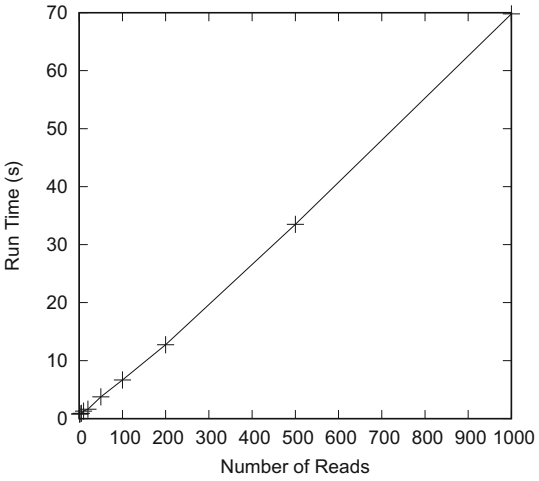
Plot `blastTimes.dat`

```
gnuplot -p blastTimes.gp
```

where `blastTimes.gp` is

```
set pointsize 2
set xlabel "Number of Reads"
set ylabel "Run Time (s)"
plot "blastTimes.dat" t "" w linespoints
```

This gives the run time of `blastn` in read mapping mode as a function of the number of reads:



The plot looks linear, so we can extrapolate to estimate the time needed to map all reads:

```
bc -l
3527065 / 1000 * 63.53 / 3600 / 24
2.59345416030092592592
```

In other words, approximately 2 days and 14 h would be needed to align the reads using BLAST in its short-read mode.

Answer 321 Indexing

```
bwa index -p dmChr2L dmChr2L.fasta
```

takes approximately 12 s to complete. The program reports computing the Burrows–Wheeler Transform (BWT) and the suffix array (SA) of the input sequence.

Answer 322 Run the command

```
bash runBwa.sh > bwaTimes.dat
```

where `runBwa.sh` is

```
for a in 1 2 5 10 20 50 100 200 500 1000
do
    echo -n $a
    ((x=$a*2*1000))
    head -n ${x} reads.fasta > tmpReads.fasta
    /usr/bin/time -p bwa mem dmChr2L tmpReads.fasta 2>&1 |
        grep '^real'                                     |
        sed 's/real//'
done
rm tmpReads.fasta
```

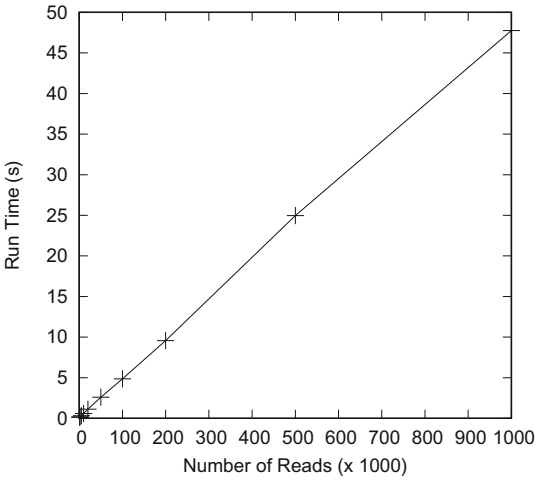
Plot `bwaTimes.dat`

```
gnuplot -p bwaTimes.gp
```

where `bwaTimes.gp` is

```
set pointsize 2
set xlabel "Number of Reads (x 1000)"
set ylabel "Run Time (s)"
plot "bwaTimes.dat" t "" w linespoints
```

to get



Again, the run time is linear in the number of reads, only roughly a thousand times faster than BLAST. Estimate the time required for mapping all reads

```
bc -l
3527065 / 1000000 * 48.274 / 60
2.837758930166666666
```

to find that all reads should be mapped in approximately 2 m 50 s.

Answer 323 The command

```
time bwa mem dmChr2L reads.fasta > reads.sam
```

takes 3 m 6 s, a bit more than the estimated 2:50.

Answer 324 Commands like

```
keywordMatcher -p  TGCC... dmChr2L.fasta
>NT_033779.5 Drosophila melanogaster chromosome 2L  TGCC...:
  12300064
```

show `reads.sam` lists every read in its forward orientation.

Answer 325 A read in the forward direction is denoted by dots, in the reverse direction by commas.

Answer 326 Pressing `g` opens a dialog for entering a position.

Answer 327 To get to the position, press `g` and enter

```
NT_033779.5:2000
```

As to position annotations, their first digits point to the actual position in the sequence. If you should ever be confused about this, look at the first position:

```
1
C
```

Answer 328 There are 10 proteins and

$$\binom{10}{2} = 45$$

protein pairs, each characterized by up to two edges, so the number of possible edges is 90. Of these 25, that is $25/90 \approx 28\%$, actually exist.

Answer 329 Set up the session

```
mkdir FastLocalAlignmentProt
cd FastLocalAlignmentProt
ln -s ../Data/mgProteome.fasta
```

Count the proteins:

```
grep -c '^>' mgProteome.fasta
476
```

The genome of *M. genitalium* encodes 476 proteins.

Answer 330 An all-against-all comparison is like filling in a square matrix, where query sequences are written along the first column and subject sequences along the top row:

	S_1	S_2	S_3	S_4	S_5
S_1					
S_2					
S_3					
S_4					
S_5					

Every empty cell in that matrix corresponds to a comparison, so there should be $5^2 = 25$ comparisons between our five example sequences, and $476^2 = 226,576$ comparisons between the proteins of *M. genitalium*. To test this prediction, generate five identical sequences

```
for a in $(seq 5); do ranseq -s 13 >> testSeq.fasta; done
```

Run the all-against-all search and count the comparisons

```
blastn -query testSeq.fasta -subject testSeq.fasta |
grep -c 'Score = '
25
```

The result is as predicted.

Answer 331

```
time blastp -max_hsps 1 -outfmt 6 -query mgProteome.fasta
    -subject mgProteome.fasta -evalue 1e-5 > mgProteome.blast
```

This takes 3.5 s. Look at the result:

```
head -n 2 mgProteome.blast
lcl|MG_002 lcl|MG_002 100.000 310  0 0 1 310 1 310 0.0      626
lcl|MG_002 lcl|MG_200  41.667  60 33 1 4  61 9  68 8.90e-09 50.4
```

Answer 332 Constructing the database,

```
time makeblastdb -in mgProteome.fasta -dbtype prot -out
   mgProteome
```

takes 0.02 s. Running the comparisons,

```
time blastp -max_hsps 1 -outfmt 6 -query mgProteome.fasta
    -db mgProteome -evalue 1e-5 > mgProteome.blast
```

takes 3.3 s, about the same as without the database. So, in this case, a precomputed database has no significant effect on the speed of `blastp`.

Answer 333 Carry out the edit and save the result to `tmp`:

```
sed 's/lcl|//g' mgProteome.blast > tmp
```

Make sure the substitution worked

```
head tmp
```

Replace the original

```
mv tmp mgProteome.blast
```

Answer 334 Generate the backup

```
cp mgProteome.blast backup.blast
```

and try

```
sed 's/MG_//g' mgProteome.blast > mgProteome.blast
```

The resulting file is empty. Do not forget to regenerate the original:

```
mv backup.blast mgProteome.blast
```

Answer 335 Print the first ten lines of output

```
head mgProteome.blast
```

to get the tabular BLAST result

q.	s.	% id.	al. len.	mism.	gaps	q. start	q. end	s. start	s. end	eval.	score
002	002	100.000	310	0	0	1	310	1	310	0.0	626
002	200	41.667	60	33	1	4	61	9	68	8.90e-09	50.4
002	019	37.879	66	33	1	4	61	9	74	2.34e-08	48.9
003	003	100.000	650	0	0	1	650	1	650	0.0	1344
003	203	45.426	645	329	9	10	647	5	633	0.0	543
004	004	100.000	836	0	0	1	836	1	836	0.0	1699
004	204	34.366	678	423	10	32	702	20	682	2.11e-125	390
005	005	100.000	417	0	0	1	417	1	417	0.0	851
006	006	100.000	210	0	0	1	210	1	210	1.52e-159	434
007	007	100.000	254	0	0	1	254	1	254	0.0	509

where we have removed the `MG_` in front of each accession to make the table fit the page. The first nonself pair is MG_002 and MG_200. Look at the header of MG_002

```
grep MG_002 mgProteome.fasta
>lcl|MG_002 DnaJ domain protein
```

and of `MG_200`

```
grep MG_200 mgProteome.fasta
>lcl|MG_200 DnaJ domain protein
```

to find they are both DnaJ domain proteins. These belong to the group of molecular chaperones involved in protein folding and cellular stress response.

Answer 336

```
wc -l mgProteome.blast
836 mgProteome.blast
```

Reading a table containing 836 entries might become rather tedious...

Answer 337 The pipeline we are looking for is

```
cut -f 1 mgProteome.blast |
sort                      |
uniq -c                   |
sort -n -r                |
head
```

which gives

```
     17 MG_410
     17 MG_180
     17 MG_179
     16 MG_526
     16 MG_467
     16 MG_303
     16 MG_290
     16 MG_187
     16 MG_065
     16 MG_042
```

In other words, MG_410, MG_180, and MG_179 each have 17 hits in the proteome. Since one of these is a self-hit, they all have at least 16 homologues.

Answer 338 Extract the seventeen proteins linked to MG_410:

```
grep MG_410 mgProteome.blast | # Extract hits to MG_410
cut -f 2                     | # Cut subject column
sort                         |
uniq > protFam.txt
```

and write the protein family to one line

```
tr '\n' ' ' < protFam.txt
```

to get

```
MG_014 MG_015 MG_042 MG_065 MG_079 MG_080 MG_119 MG_179 MG_180
MG_187 MG_290 MG_303 MG_304 MG_410 MG_421 MG_467 MG_526
```

The conversion from single column to single row makes printing easier.

Answer 339 Execute

```
neato -T x11 example2.dot
```

where `example2.dot` is

```
graph G {
        1 -- 2
        2 -- 3
        2 -- 5
        4 -- 5
}
```

to get the graphic depicted in the problem.

Answer 340 Here is the dot code for specifying the figure:

```
graph G {
        2 -- 1 [dir=forward]
        2 -- 5 [dir=forward]
        3 -- 4 [dir=forward]
        4 -- 1 [dir=both]
        5 -- 1 [dir=forward]
        5 -- 3 [dir=forward]
}
```

Answer 341 The expected number of edges is the maximal number of edges times the edge probability:

$$10 \times 9 \times 0.5 = 45.$$

Take a look at the output file

```
cat ranDot.dot
graph G {
        1 -- 3 [dir=forward]
        1 -- 5 [dir=forward]
        1 -- 6 [dir=both]
        1 -- 7 [dir=both]
...
```

We can thus compute the number of observed edges by filtering for the two types of edges, "forward" and "both", and counting "both" twice, "forward" once:

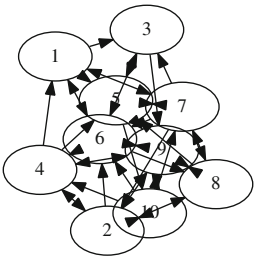
```
awk '/both/{s+=2}/forward/{s++}END{print s}' ranDot.dot
43
```

Your result may well differ from ours, but it should also be close to the expectation (45).

Answer 342 First apply `neato`

```
neato -T x11 ranDot.dot
```

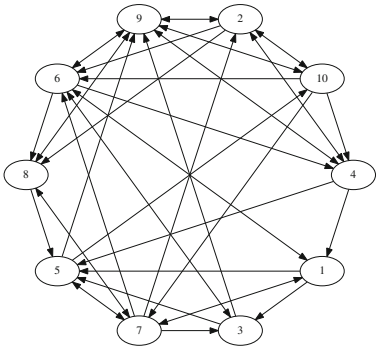
to get



This is rather messy. Try `circo`

```
circo -T x11 ranDot.dot
```

to get

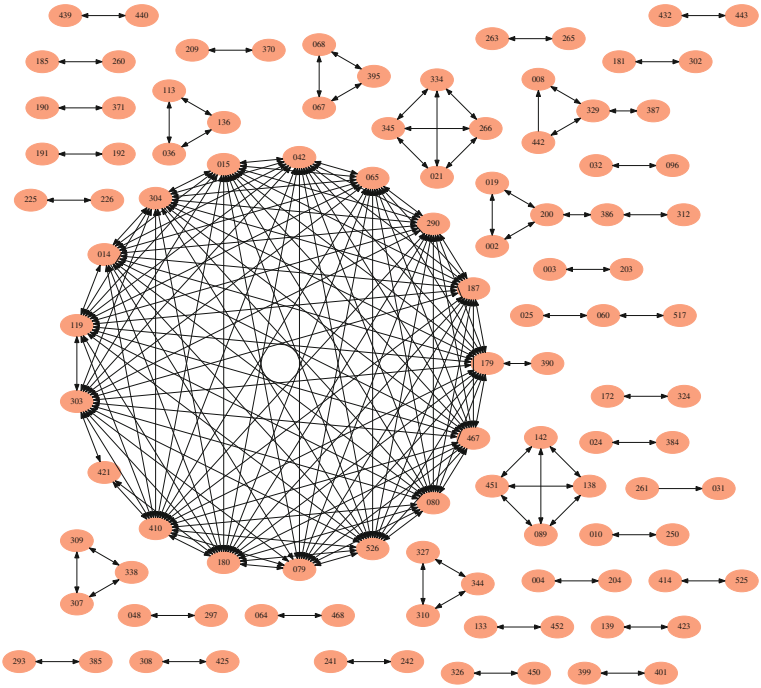


which is a nicer layout for a highly connected graph such as ours.

Answer 343 *M. genitalium* contains one highly connected protein family. It is much better resolved by `circo` than `neato`; hence, we used

```
circo -T x11 mgProteome.dot
```

to get



The protein family at the center contains the seventeen proteins already in `protFam.txt` plus MG_390. To record this, write

```
cp protFam.txt protFam2.txt
echo 'MG_390' >> protFam2.txt
```

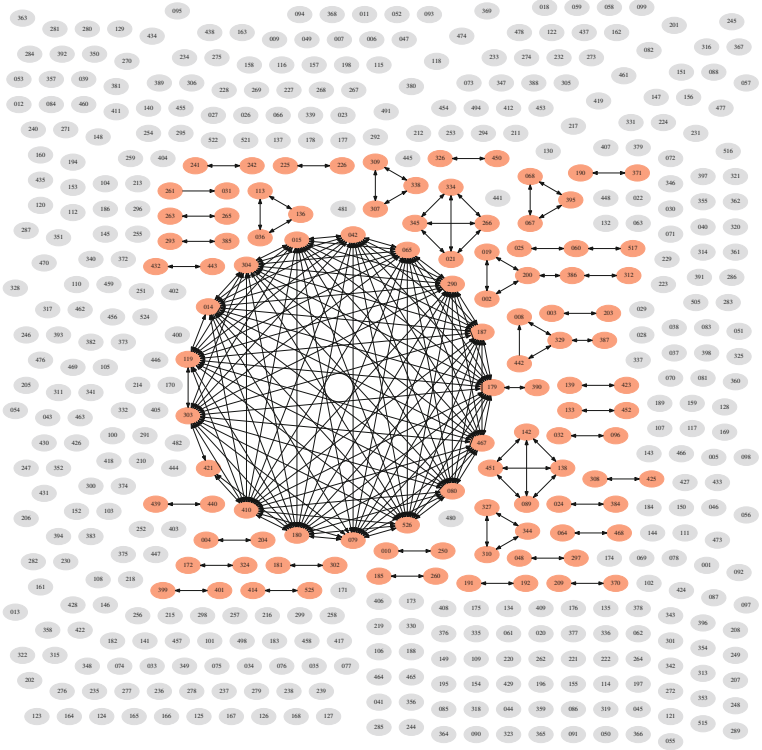
where `protFam2.txt` is now

```
tr '\n' ' ' < protFam2.txt
MG_014 MG_015 MG_042 MG_065 MG_079 MG_080 MG_119 MG_179 MG_180
MG_187 MG_290 MG_303 MG_304 MG_410 MG_421 MG_467 MG_526 MG_390
```

Answer 344 Calculate the new layout

```
circo -T x11 mgProteome2.dot
```

to get



Compute the number of singletons:

```
awk 'NF==1{if(length($1)==3)c++}END{print c}' mgProteome2.dot
374
```

That is, $374/476 \times 100 \approx 78.6\%$ of the proteome consists of singletons, given $E \leq 10^{-5}$.

Answer 345 Construct the list of alternatives matches

```
tr '\n' '|' < protFam2.txt
```

and then copy and paste them into the command

```
grep -E '(MG_014|MG_015|...|MG_390)' mgProteome.fasta
```

to get

```
>lcl|MG_014 ABC transporter, ATP-binding
>lcl|MG_015 ABC transporter, ATP-binding
>lcl|MG_042 spermidine
>lcl|MG_065 ABC transporter, ATP-binding protein
>lcl|MG_079 oligopeptide ABC transporter, ATP-binding protein
>lcl|MG_080 oligopeptide ABC transporter, ATP-binding protein
>lcl|MG_119 ABC transporter, ATP-binding protein
>lcl|MG_179 metal ion ABC transporter, ATP-binding protein, putative
>lcl|MG_180 metal ion ABC transporter ATP-binding protein, putative
>lcl|MG_187 ABC transporter, ATP-binding protein
>lcl|MG_290 phosphonate ABC transporter, ATP-binding protein, putative
>lcl|MG_303 metal ion ABC transporter, ATP-binding protein, putative
>lcl|MG_304 metal ion ABC transporter, ATP-binding protein, putative
>lcl|MG_390 ABC transporter, ATP-binding
>lcl|MG_410 phosphate ABC transporter, ATP-binding protein
>lcl|MG_421 excinuclease ABC, A subunit
>lcl|MG_467 ABC transporter, ATP-binding protein
>lcl|MG_526 ABC transporter, ATP-binding protein
```

Most of these are annotated as ABC transporters, so our protein family consists of ABC transporters.

Answer 346 Open `prosite.doc` in `emacs` or `less` and search for *ABC transporter*. Here is an edited version of the relevant entry:

```
{PDOC00185}
{PS00211; ABC_TRANSPORTER_1}
{PS50893; ABC_TRANSPORTER_2}
{BEGIN}
********************************************************************
* ATP-binding cassette, ABC transporter-type, signature and profile *
********************************************************************

ABC transporters belong to the ATP-Binding Cassette (ABC) superfamily
which uses the hydrolysis of ATP to energize diverse biological systems.
ABC transporters are minimally constituted of two conserved regions:
a highly conserved ATP binding cassette (ABC) and a less conserved
transmembrane domain (TMD). These regions can be found on the same
protein or on two different ones. Most ABC transporters function as a
dimer and therefore are constituted of four domains, two ABC modules and
two TMDs [1].

ABC transporters are involved in the export or import of a wide variety
of substrates ranging from small ions to macromolecules. The major
function of ABC import systems is to provide essential nutrients to
bacteria. They are found only in prokaryotes and their four constitutive
domains are usually encoded by independent polypeptides (two ABC
proteins and two TMD proteins). Prokaryotic importers require additional
extracytoplasmic binding proteins (one or more per systems) for function.
In contrast, export systems are involved in the extrusion of noxious
substances, the export of extracellular toxins and the targeting of
membrane components. They are found in all living organisms and in
general the TMD is fused to the ABC module in a variety of combinations.
Some eukaryotic exporters encode the four domains on the same
polypeptide chain [2,3].
...
{END}
```

Answer 347 A lot of energy-consuming transfer of various molecules across the bacterial cell membrane is conducted via ABC transport proteins.

Answer 348 Set up the working directory

```
mkdir PsiBlast
cd PsiBlast
ln ../Data/mgProteome.fasta
```

Convert the proteome of *M. genitalium* to a BLAST database

```
makeblastdb -dbtype prot -in mgProteome.fasta -out mgProteome
```

Get the sequence of protein M_410

```
getSeq -s 410 mgProteome.fasta > mg_410.fasta
```

and compare it to the proteome using `psiblast`:

```
psiblast -query mg_410.fasta -db mgProteome -outfmt 6 >
    mg_410.psi
```

The syntax should look familiar from our previous work with `blastn` and `blastp`. Count the number of unique hits with $E \leq 10^{-5}$:

```
awk '{if($11<=10^-5)print $1 "\t" $2}' mg_410.psi       |
tr '\t' '\n'                                            |
sort                                                    |
uniq                                                    |
wc -l
17
```

We found 17 members of this protein family.

Answer 349 Run `blastp`:

```
blastp -query mg_410.fasta -db mgProteome -outfmt 6 > mg_410.bp
```

Visual inspection of `mg_410.psi` and `mg_410.bp` shows that they are at least highly similar. To confirm identity, use

```
diff mg_410.psi mg_410.bp
```

Answer 350 Run `psiblast` iteratively

```
psiblast -num_iterations 0 -query mg_410.fasta -db mgProteome
    -outfmt 6 > mg_410b.psi
```

To count the number of rounds, look for the number of comparisons with MG_410 as query and subject

```
awk '{if($1~/410/ && $2~/410/)c++}END{print c}' mg_410b.psi
4
```

Now find the row numbers where each round starts

```
awk '{if($1~/410/ && $2~/410/)print NR}' mg_410b.psi
1
49
104
155
```

Finally, count the number of distinct hits with $E \leq 10^{-5}$ from line 155 onward:

```
tail -n +155 mg_410b.psi                  | # Print line >= 155
grep MG                                   | # Filter out footer
awk '{if($11<=10^-5)print $1 "\n" $2}'    | # Check E-value
sort                                      |
uniq                                      |
wc -l
21
```

Answer 351 Get `protFam2.txt`:

```
cp ../FastLocalAlignmentProt/protFam2.txt .
```

and look for the differences between the lists

```
diff psiBlastList.txt protFam2.txt
7,8d6
< MG_107
< MG_110
14d11
< MG_298
17d13
< MG_390
21a18
> MG_390
```

Three extra proteins MG_107, MG_110, and MG_298 were found by `psiblast`, while MG_390 is contained in both lists, but at different positions.

Answer 352 Use `grep` with extended notation to extract the header lines of the three extra proteins:

```
grep -E '(MG_107|MG_110|MG_298)' mgProteome.fasta
>lcl|MG_107 guanylate kinase
>lcl|MG_110 ribosome small subunit-dependent GTPase A
>lcl|MG_298 chromosome segregation protein SMC
```

These annotations are not in any obvious way connected to “ABC transporter”. However, the kinase binds ATP, the GTPase A binds GTP, which is similar to ATP, and SMC proteins belong to the ATPases. That is to say, ATP binding is the common feature of the 21 proteins we have identified.

Answer 353 Run `psiblast`

```
psiblast -out_ascii_pssm psiBlast.mat -num_iterations 0 \
-query mg_410.fasta -db mgProteome -outfmt 6 > mg_410b.psi
```

The position-specific score matrix is contained in the first 20 columns labeled `A` for alanine through `V` for valine. It consists of 329 rows, one for each amino acid in MG_410.

Answer 354 Identify the most frequent amino acid

```
cchar mg_410.fasta |
sed '/^#/d'        |
sort -k 2 -n -r    |
head -n 1
I   41  0.124620
```

Extract all positions occupied by isoleucine:

```
awk 'NR==3 || $2=="I"{print}' psiBlast.mat
```

which returns the position-specific score information

```
      A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V ...
 5 I  -1 -3 -3 -3 -1 -3 -3 -4 -3  4  1 -3  1  0 -3 -2 -1 -3 -1  2 ...
19 I  -1 -1 -1  0 -2  0  3 -3 -1  2  0 -1  0 -1 -2 -1 -1 -3 -2  1 ...
30 I  -2 -3 -4 -4 -1 -3 -3 -4 -3  3  4 -3  2  0 -3 -3 -1 -2 -1  2 ...
...
```

The match score is the score for isoleucine (`I`); to find its range, extend the previous command

```
awk 'NR==3 || $2=="I"{print}' psiBlast.mat      |
tail -n +2                                      |
awk '{print $12}'                               |
sort                                            |
uniq                                            |
tr '\n' ' '
0 1 2 3 4 5
```

The match score for isoleucine ranges between 0 and 5. The corresponding BLOSUM62 score is 4.

Answer 355 The first three accessions need to be extended to achieve uniqueness:

```
getSeq -s 'HBA_HUMAN H' uniprot_sprot.fasta > hbaHuman.fasta
getSeq -s 'HBA_HORSE H' uniprot_sprot.fasta > hbaHorse.fasta
getSeq -s 'HBB_HUMAN H' uniprot_sprot.fasta > hbbHuman.fasta
getSeq -s HBB_HORSE     uniprot_sprot.fasta > hbbHorse.fasta
```

Alternatively, `getSeq` could have been applied twice, for example

```
getSeq -s HBA_HUMAN uniprot_sprot.fasta | getSeq -s Hem
```

Answer 356 Construct the file containing the subject sequences:

```
cat hbbHuman.fasta hbaHorse.fasta hbbHorse.fasta >
  subject.fasta
```

Then run `blastp`

```
blastp -query hbaHuman.fasta -subject subject.fasta -outfmt 2
```

to get the slightly edited query-anchored alignment

```
Q_1  1    MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG  60
S_2  1    MVLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHFDLSHGSAQVKAHG  60
S_1  4      LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFDLSMGNPKVKAHG  65
                                                        \  \
                                                        |  |
                                                        G TPDAV
S_3  3      LSGEEKAAVLALWDKVNEE--EVGGEALGRLLVVYPWTQRFFDSFDLSNGNPKVKAHG  64
                                                        \   \
                                                        |   |
                                                        G PGAVM

Q_1  61   KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTP  120
S_2  61   KKVGDALTLAVGHLDDLPGALSNLSDLHAHKLRVDPVNFKLLSHCLLSTLAVHLPNDFTP  120
S_1  66   KKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTP  125
S_3  65   KKVLHSFGEGVHHLDNLKGTFAALSELHCDKLHVDPENFRLLGNVLVVVLARHFGKDFTP  124

Q_1  121  AVHASLDKFLASVSTVLTSKYR  142
S_2  121  AVHASLDKFLSSVSTVLTSKYR  142
S_1  126  PVQAAYQKVVAGVANALAHKY   146
S_3  125  ELQASYQKVVAGVANALAHKY   145
```

The amino acids printed below

```
\
|
```

in Subjects `S_1` and `S_3` are insertions.

Answer 357 Construct another set of subject sequences:

```
cat hbaHuman.fasta hbaHorse.fasta hbbHorse.fasta >
   subject2.fasta
```

and run `blastp`

```
blastp -query hbbHuman.fasta -subject subject2.fasta -outfmt 2
```

to get the query-anchored alignment

```
Q_1 2    VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKV 61
S_3 1    VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFGDLSNPGAVMGNPKV 60
S_1 3      LSPADKTNVKAAWGKVHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQV 56
                          \
                          |
                         GA
S_2 3      LSAADKTNVKAAWSKVHAGEYGAEALERMFLGFPTTKTYFPHF-DLS-----HGSAQV 56
                          \
                          |
                         GG

Q_1 62   KAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGK 121
S_3 61   KAHGKKVLHSFGEGVHHLDNLKGTFAALSELHCDKLHVDPENFRLLGNVLVVVLARHFGK 120
S_1 57   KGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPA 116
S_2 57   KAHGKKVGDALTLAVGHLDDLPGALSNLSDLHAHKLRVDPVNFKLLSHCLLSTLAVHLPN 116

Q_1 122  EFTPPVQAAYQKVVAGVANALAHKYH 147
S_3 121  DFTPELQASYQKVVAGVANALAHKYH 146
S_1 117  EFTPAVHASLDKFLASVSTVLTSKY  141
S_2 117  DFTPAVHASLDKFLSSVSTVLTSKY  141
```

Now, there is one insertion of two amino acids, `GA` into HBA_HUMAN and `GG` into HBA_HORSE.

Answer 358 Use commands like

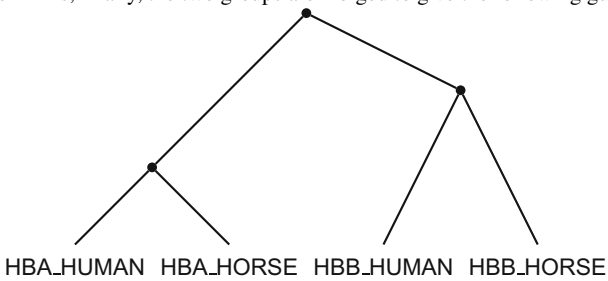
```
gal -p -i hbaHuman.fasta -j hbaHorse.fasta -m BLOSUM62 |
grep Score
```

to get

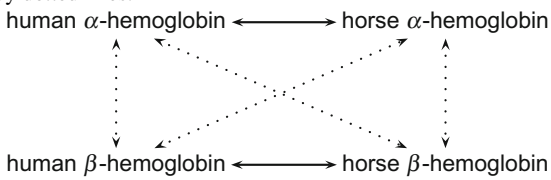
	HBA_HUMAN	HBA_HORSE	HBB_HUMAN	HBB_HORSE
HBA_HUMAN	—	648	282	268
HBA_HORSE	0.00	—	264	266
HBB_HUMAN	0.56	0.59	—	633
HBB_HORSE	0.59	0.59	0.02	—

where scores are in the top triangle and distances in the bottom triangle.

Answer 359 The most similar sequences, HBA from human and horse, are clustered first, followed by the HBBs; finally, the two groups are merged to give the following guide tree:



Answer 360 The pairs α/α and β/β are orthologs marked by solid lines, the α/β pairs are paralogs marked by dotted lines:



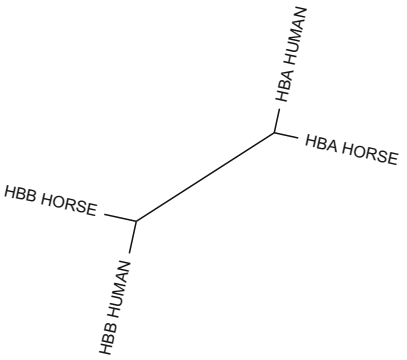
Answer 361 Run `clustalw`

```
clustalw hemoglobin.fasta
```

A slightly edited version of the guide tree in `hemoglobin.dnd` looks like this

```
(
(
HBA_HORSE:0.06,
HBA_HUMAN:0.06)
:0.4,
HBB_HORSE:0.08,
HBB_HUMAN:0.08);
```

Its graphical representation is



Answer 362 There are four gaps. The left-most has two origins,

```
-V
MV
```

was introduced when aligning the two β-hemoglobins, while

```
-M
-M
```

was introduced when the α- and β-pairs were aligned. The other three gaps each affect a pair of sequences, so they must have been introduced when the α- and β-pairs were aligned.

Answer 363 The `clustalw` alignment generates an end gap:

```
S1              ATG
S2              -AG
                  *
```

Answer 364 Values ≤ 5 give the alternative gap pattern, for example,

```
clustalw hemoglobin.fasta -GAPOPEN=5
```

Answer 365 Run the command

```
bash simTimes1.sh > simTimes1.dat
```

where `simTimes1.sh` is

```
for a in 100 200 500 1000 2000 5000 10000 20000
do
    echo -n ${a} ' '
    ranseq -n 2 -l ${a} > test.fasta
    /usr/bin/time -p clustalw test.fasta 2>&1 |
        grep real                             |
        sed 's/real //'
done
rm test.fasta
```

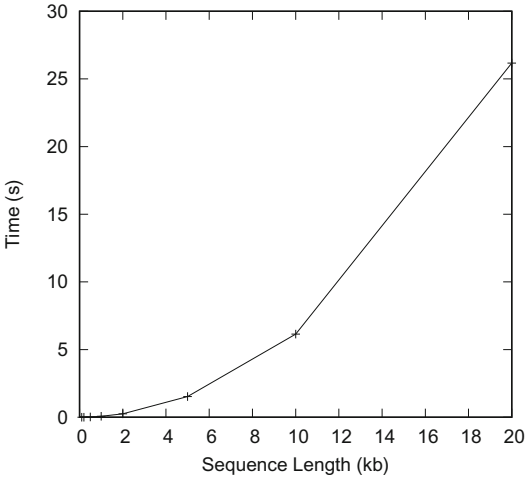
to collect the times. Plot them

```
gnuplot -p simTimes1.gp
```

where `simTimes1.gp` is

```
set xlabel "Sequence Length (kb)"
set ylabel "Time (s)"
plot "simTimes1.dat" using ($1/1000):2 title "" with
    linespoints
```

to get the run time of `clustalw` as a function of sequence length:



As the sequence length is doubled, the run time is roughly quadrupled.

Answer 366 Run

```
bash simTimes2.sh > simTimes2.dat
```

to collect run times, where `simTimes2.sh` is

```
for a in 2 5 10 20 50 100
do
    echo -n ${a} ' '
    ranseq -n ${a} -l 1000 > test.fasta
    /usr/bin/time -p clustalw test.fasta 2>&1        |
        grep real                                    |
        sed 's/real //'
done
```

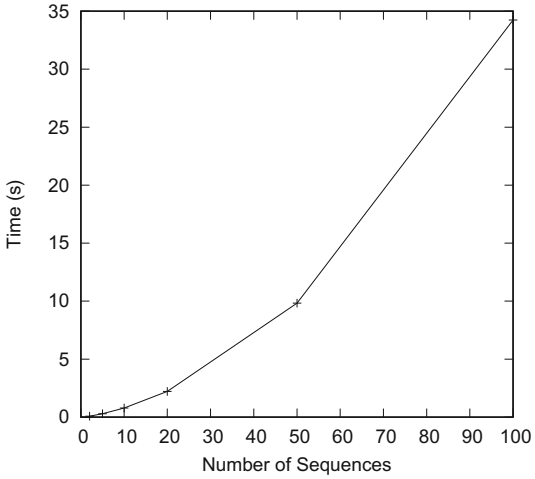
Plot them

```
gnuplot -p simTimes2.gp
```

where `simTimes2.gp` is

```
set xlabel "Number of Sequences"
set ylabel "Time (s)"
plot "simTimes2.dat" title "" with linespoints
```

to get



Again, as the number of sequences doubles, the run time roughly quadruples.

Answer 367 Get the sample

```
getSeq -s Hemoglobin uniprot_sprot.fasta |
awk -f fasta2tab.awk                     |
awk -f shuffle.awk                       |
head -n 100                              |
tr '\t' '\n'                             |
fold > hb100.fasta
```

Make sure 100 sequences were obtained

```
grep -c '^>' hb100.fasta
100
```

Measure the run time of `clustalw`

```
/usr/bin/time -p clustalw hb100.fasta 2>&1 |
grep real
```

which takes 2.30 s. Aligning 200 hemoglobin sequences takes 8.92 s, roughly four times longer.

Answer 368 There are

```
getSeq -s Hemoglobin uniprot_sprot.fasta | grep -c '^>'
833
```

hemoglobin sequences in UniProt. Aligning all of them with `clustalw` would roughly take $2.3 \times 4^3 = 147s$.

Answer 369 Get the hemoglobin sequence:

```
getSeq -s Hemoglobin uniprot_sprot.fasta > hemoglobinAll.fasta
```

Count them, just to make sure:

```
grep -c '^>' hemoglobinAll.fasta
833
```

Align them

```
time clustalw hemoglobinAll.fasta > /dev/null
```

which takes 150.1 s. This is close to the predicted run time of 147 s.

Answer 370 Set up the session:

```
mkdir TreesOfLife
cd TreesOfLife
```

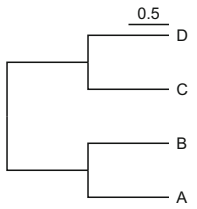
Without a scale bar the absolute branch lengths are meaningless, but since the branches all look the same, we can write

```
((A:1,B:1):1,(C:1,D:1):1);
```

Answer 371 The command

```
new2view first.tree
```

yields

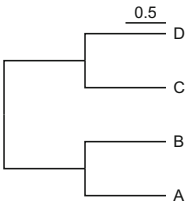


Your scaling might differ from ours—this can be adjusted with the `-d` and `-s` options. Moreover, the tree is now drawn from left to right rather than top to bottom. Left to right is often used when drawing phylogenies, as this makes it easier to place the taxon labels. However, when talking about the children of a node, we shall continue to refer to the right and the left child, rather than the top and the bottom child.

Answer 372 `second.tree` contains

```
((A,B),(C,D));
```

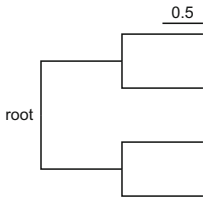
which looks identical to the tree with unit branch lengths—but that is just the drawing convention adopted by `new2view`:



Answer 373 The Newick tree is now

```
((,),(,))root;
```

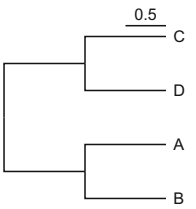
and looks like



Answer 374 In Newick notation, we have

```
((B,A),(D,C));
```

which is rendered with switched leaf labels compared to `second.tree`:

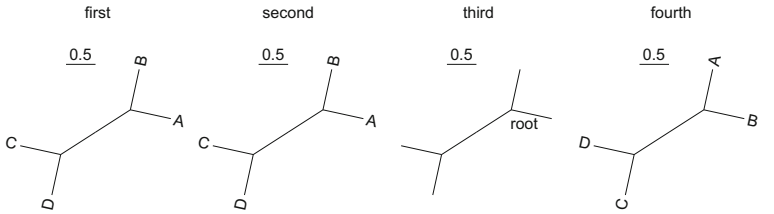


However, in phylogenies only the branching pattern matters, and hence a phylogeny remains unchanged by rotating branches around nodes.

Answer 375 Visualizing all tree files at once using

```
new2view -u *.tree
```

yields



An unrooted tree does not have a unique starting point. It still has direction, though, going from internal to leaf nodes. The unrooted layout is also known as "radial".

Answer 376 Save the four trees

```
cat first.tree second.tree third.tree fourth.tree >
   fourTreesU.tree
```

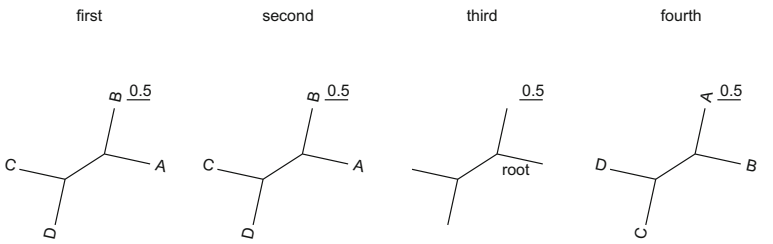
edit them to obtain

```
(A:1,B:1,(C:1,D:1):1);
(A,B,(C,D));
(,,(,))root;
(B,A,(D,C));
```

and draw them

```
new2view fourTreesU.tree
```

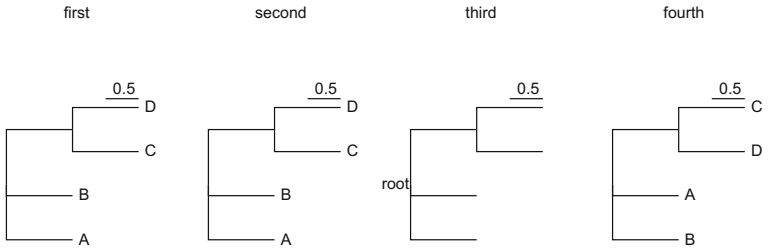
to get the expected radial layout—notice the root position in the third tree. This radial layout is the standard representation of unrooted trees in biology.



When enforcing the rooted layout

```
new2view -r fourTreesU.tree
```

we see the trifurcating root:



Answer 377 The preorder sequence is 2, 1, 6, 4, 3, 5, 8, 7, and 9.

Answer 378 The inorder sequence is 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Answer 379 The postorder sequence is 1, 3, 5, 4, 7, 9, 8, 6, and 2.

Answer 380 The commands for inorder, preorder, and postorder traversal are

```
traverseTree -t inorder traverse.tree
traverseTree -t preorder traverse.tree
traverseTree -t postorder traverse.tree
```

where `traverse.tree` contains

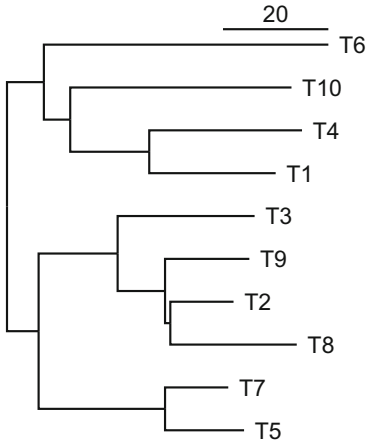
```
(,((,),(,)));
```

All three methods are centered on visiting child nodes rather than neighbor nodes. Hence, the name “depth first” traversal.

Answer 381 The command

```
genTree | new2view
```

gives, for example



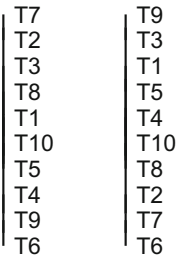
Your tree is bound to differ, but its leaves will also not be perfectly lined up. The leaves are labeled T1,..., T*n* for *taxon* 1,...,*n*. With default settings, `genTree` produces branch

lengths proportional to the number of mutations along a given branch. Since this is a random variable, branches rarely end at the same point along the horizontal axis. This interpretation of branch lengths as mutations seems to conflict with our intuition that time is marked along the x-axis: present on the right, past on the left. The conflict is resolved when we realize that the number of mutations is drawn from a Poisson distribution with mean proportional to the time that has elapsed between a given node and its parent. If we knew the time, the leaves would all align.

Answer 382 Repeat

```
genTree -t 0 | new2view
```

twice to get, for example



The trees differ in the order of taxa.

Answer 383 Compute $n!$ as a function of n:

```
awk -f factorial.awk -v n=100 > factorial.dat
```

where `factorial.awk` is

```
BEGIN{
    if(!n)  # n set via -v?
        n = 100
    f = 1
    for(i=1; i<=n; i++){
        f *= i
        print i, f
    }
}
```

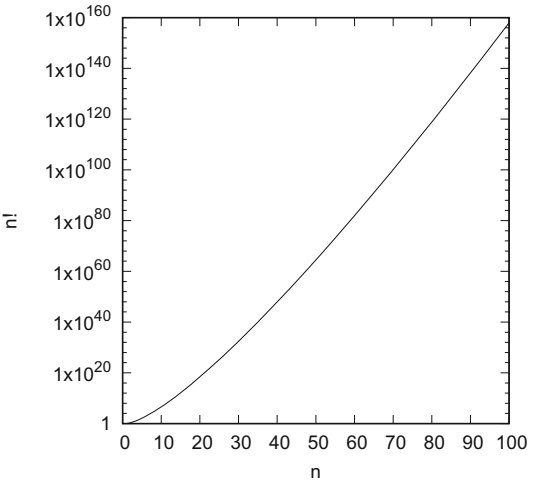
Plot the results

```
gnuplot -p plot.gp
```

where `plot.gp` is

```
set xlabel "n"
set ylabel "n!"
set logscale y
plot[][] "factorial.dat" t "" w l
```

to get



This gives a first impression of how the number of phylogenies scale with n.

Answer 384 Compute the number of trees

```
awk -f numTrees.awk -v n=100 > numTrees.dat
```

where `numTrees.awk` is

```
BEGIN{
    if(!n)  # n set via -v?
        n = 100
    f = 1
    if(n>0)
        print 1, f
    if(n>1)
        print 2, f
    for(i=3; i<=n; i++){
        f = 1
        for(j=3; j<=(2*i-3); j+=2)
            f *= j
        print i, f
    }
}
```

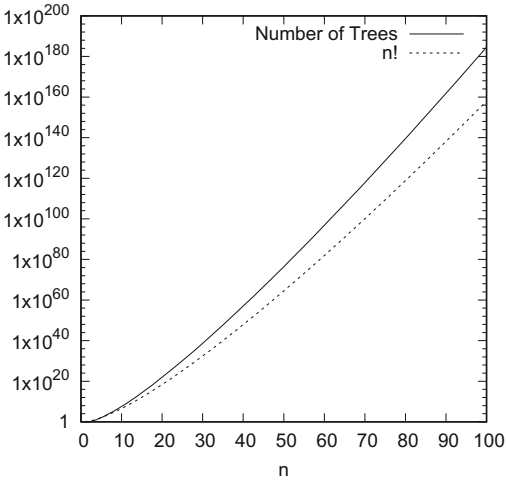
Plot the result together with n!

```
gnuplot -p plot2.gp
```

where `plot2.gp` is

```
set xlabel "n"
set logscale y
plot[][] "numTrees.dat" t "Number of Trees" w l,\
"factorial.dat" t "n!" w l
```

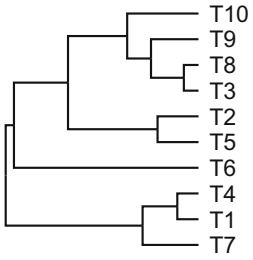
to get the exact number of phylogenies as a function of sample size, n, compared to the estimation via $n!$. The factorial estimation is roughly 20 orders of magnitude too small when $n = 100$:



Answer 385 Use the command

```
genTree -s -t 0 | new2view
```

to get something similar to



This time all the leaves line up because rather than representing mutations, the branches go back to the simulated times of species divergence.

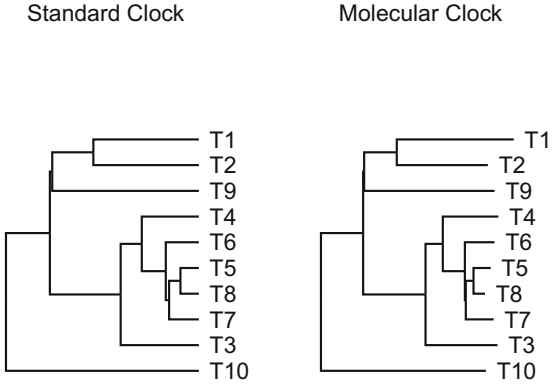
Answer 386 Use, for example,

```
genTree -s -S 13
```

and

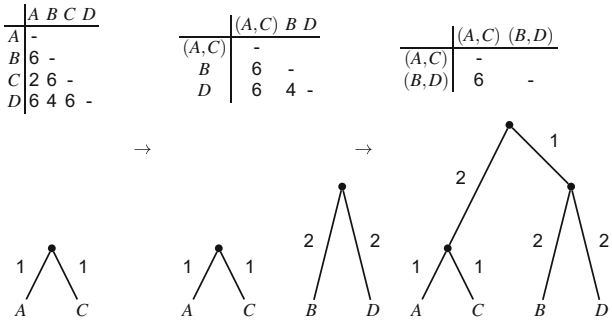
```
genTree -S 13
```

to get



This illustrates that the molecular clock is a stochastic clock that behaves only approximately like its standard version.

Answer 387 Here is the corresponding trace of the UPGMA algorithm:



Answer 388 The distances

- fulfill the three point criterion,
- fit the UPGMA tree,
- and, by implication, for n taxa there are no more than $n-1$ distinct entries in the distance matrix;

in other words, they are ultrametric.

Answer 389 Get the headers of the *Hominidae* sequences:

```
grep '^>' hominidae.fasta

>Pongo
>Gorilla
>Homo
>Pan
```

Answer 390 Here are the full scientific names and the trivial names of the *Hominidae* in `hominidae.fasta`:

Name in File	Full Name	Trivial Name
Homo	*Homo sapiens*	human
Pan	*Pan troglodytes/paniscus*	chimp
Gorilla	*Gorilla gorilla*	gorilla
Pongo	*Pongo pygmaeus*	orangutan

Answer 391 Extract the first ten polymorphic positions:

```
gd -P hominidae.fasta      |  # get polymorphisms
getSeq -c -s Pos           |  # exclude positions
cutSeq -r 1-10

>Homo 1..10
GTCATTCACC
>Pan 1..10
ATTATCCACC
>Gorilla 1..10
GTTGTTTATC
>Pongo 1..10
ACCACCCGTT
```

to compute the distance matrix

	Homo	*Pan*	*Gorilla*	*Pongo*
Homo	-			
Pan	3	-		
Gorilla	4	5	-	
Pongo	7	6	9	-

Answer 392 The file `test.dist` should look like this:

```
cat test.dist

4
Ho 0 3 4 7
Pa 3 0 5 6
Go 4 5 0 9
Po 7 6 9 0
```

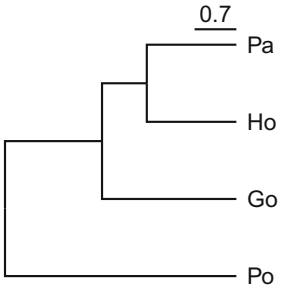
We cluster the taxa in `test.dist` by tracing the UPGMA algorithm:

```
clustDist -u -m test.dist
****** Round 1 ******
Ho       0   3.00   4.00   7.00
Pa    3.00      0   5.00   6.00
Go    4.00   5.00      0   9.00
Po    7.00   6.00   9.00      0
****** Round 2 ******
Go          0   9.00   4.50
Po       9.00      0   6.50
Ho,Pa    4.50   6.50      0
****** Round 3 ******
Po             0   7.75
Go,Ho,Pa    7.75      0
(Po:3.875,(Go:2.250,(Ho:1.500,Pa:1.500):0.750):1.625);
```

To visualize the tree, run

```
clustDist -u test.dist | new2view -d 3 -s 0.7
```

where `-d 3` restricts the smallest dimension of the tree to 3 cm, and `-s 0.7` sets the length of the scale bar, to get



Answer 393 Compute the distances

```
dnaDist hominidae.fasta > hominidae.dist
```

and print them to the screen:

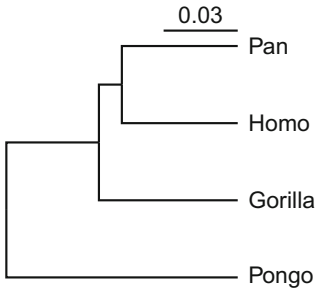
```
cat hominidae.dist
4
Homo      0.000000 0.093798 0.111717 0.180872
Pan       0.093798 0.000000 0.113013 0.192322
Gorilla   0.111717 0.113013 0.000000 0.188008
Pongo     0.180872 0.192322 0.188008 0.000000
```

Since this matrix has more than three distinct entries, the distances are not ultrametric.

Answer 394 The command

```
clustDist -u hominidae.dist |
new2view
```

gives the *Hominidae* phylogeny



Answer 395 Count the primate taxa contained in `primates.fasta`

```
grep -c '^>' primates.fasta
27
```

Compute the sequence lengths of their mitochondrial genomes and sort them

```
cchar -s primates.fasta   |
grep '^>'                 |
sort -k 2
```

to find they range between 15467 and 17036 bp.

Answer 396 Our measurements of the "real" time were as follows:

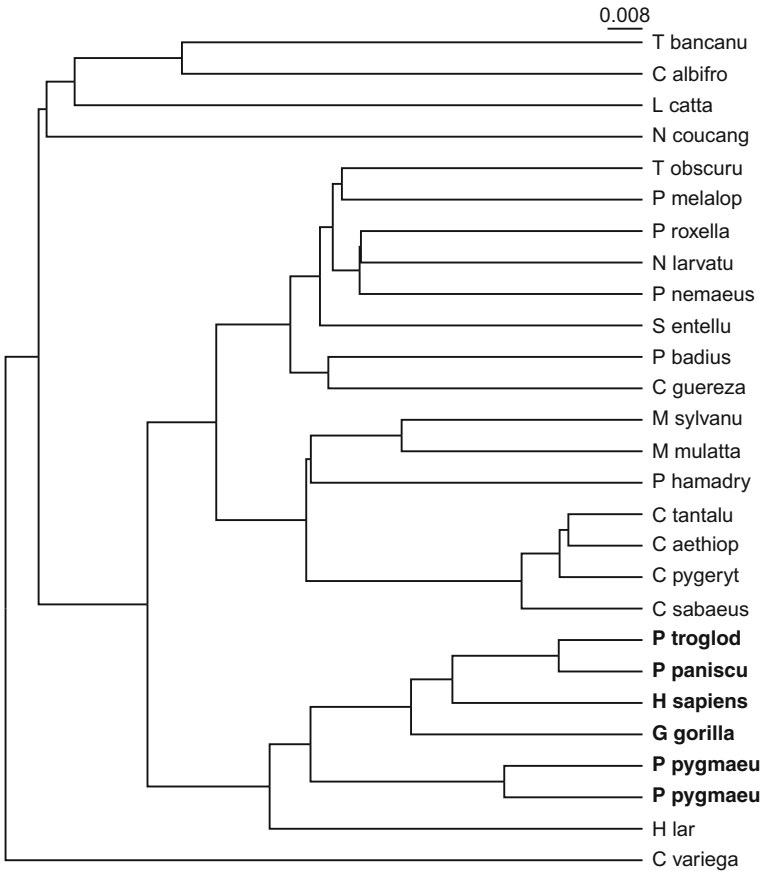
Threads	Time (s)
1	2.087
2	1.093
4	0.611
8	0.407

Your exact measurements are bound to differ slightly from ours, but the trend should be similar. If a lot of computing is going on while you are making these measurements, eight threads would not result in as much speedup as we observed when making these runs on an otherwise idle machine.

Answer 397 Cluster the distances and draw the primate phylogeny:

```
andi -t 8 primates.fasta  |
clustDist -u              |
new2view -d 12
```

which gives



The branching order for the *Hominidae* in **bold** is the same as with the small data set.

Answer 398 By plugging the distances into the equation describing the four point criterion, we get

$$7 + 7 = 10 + 4 \geq 5 + 5,$$

which is true.

Answer 399 In Fig. 5.4a, we have $d_{AB} = 5$, $d_{AC} = 7$, and $d_{BC} = 4$. Since these are three distinct numbers, the three point criterion does not hold.

Answer 400 Our trace of neighbor-joining begins with the row sums:

	A	B	C	D	r_i
A	-	5	7	10	**22**
B		-	4	7	**16**
C			-	5	**16**
D				-	**22**

Answer 401

	A	B	C	D	r_i
A	-	5	7	10	22
B	**-14**	-	4	7	16
C	**-12**	**-12**	-	5	16
D	**-12**	**-12**	**-14**	-	22

Answer 402

	C	D	(A, B)
C	-	5	3
D		-	6
(A, B)			-

Answer 403

$$d_{A(AB)} = (2 \times 5 + 22 - 16)/4 = 4,$$

and

$$d_{B(AB)} = (2 \times 5 + 16 - 22)/4 = 1.$$

Answer 404

$$\begin{aligned} d_{rC} &= (5 + 3 - 6)/2 = 1 \\ d_{rD} &= (5 + 6 - 3)/2 = 4 \\ d_{r(AB)} &= (3 + 6 - 5)/2 = 2 \end{aligned}$$

Answer 405 Set up the session

```
mkdir NeighborJoining
cd NeighborJoining
```

First, trace the clustering procedure

```
clustDist -m test.dist
****** Round 1 ******
A       0   5.00   7.00  10.00 22.00
B    0.00      0   4.00   7.00 16.00
C    0.00   0.00      0   5.00 16.00
D    0.00   0.00   0.00      0 22.00
****** Round 2 ******
C         0   5.00   3.00
D      5.00      0   6.00
A,B    3.00   6.00      0
(C:1.000000,D:4.000000,(A:4.000000,B:1.000000):2.000000);
```

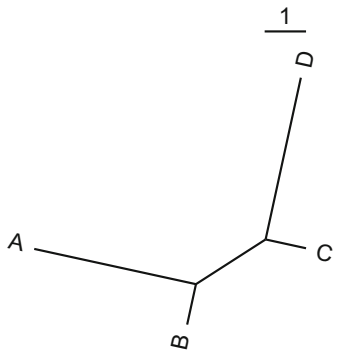
Then save the unrooted tree

```
clustDist test.dist > testU.tree
```

and draw it

```
new2view testU.tree
```

to get



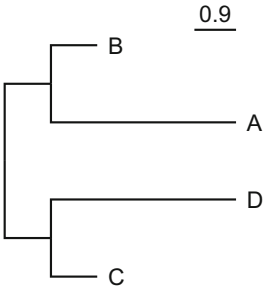
Answer 406 Execute

```
new2view -d 3 testR.tree
```

where `testR.tree` is the midpoint-rooted phylogeny

```
((C:1,D:4):1,(A:4,B:1):1);
```

to get



Answer 407 Use `testU.tree` as input to `retree`. This is started by entering

```
phylip retree
```

or just

```
retree
```

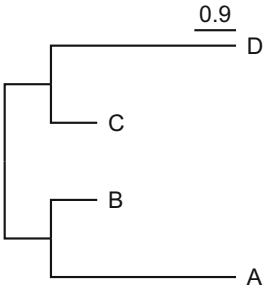
depending on your setup. Then follow the menu. PHYLIP writes the new tree to the file `outtree`. After quitting PHYLIP, execute

```
new2view -d 3 outtree
```

where `outtree` is

```
cat outtree
((A:4.0,B:1.0):1.0,(C:1.0,D:4.0):1.0);
```

to get



which has the same topology as the result computed by hand.

Answer 408 Compute the neighbor-joining tree

```
andi -t 8 primates.fasta |
clustDist > primates.tree
```

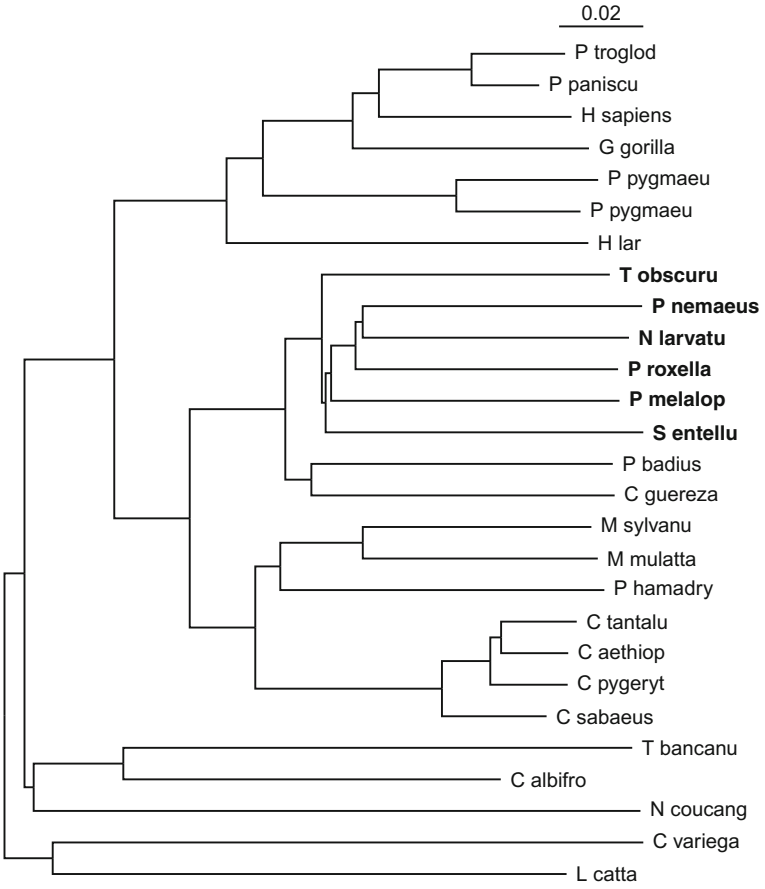
and carry out midpoint rooting

```
phylip retree
```

before drawing the primate phylogeny

```
new2view outtree
```

to get



This is quite similar to the primate phylogeny recovered with UPGMA, though there are differences in the branching order. Most of these are in the clade shown in bold, where branch lengths are short. If branches are short, clustering is based on little information and hence subject to reinterpretation by different algorithms.

Answer 409 The number of ancestors, a, as a function of the number of generations, g, in a bi-parental genealogy is

$$a = 2^g.$$

Therefore, the (theoretical) number of ancestors 30 generations back is

$$2^{30} = \left(2^{10}\right)^3 \approx \left(10^3\right)^3 = 10^9.$$

This is much larger than the human population at that time: If a generation is 25 years, 30 generations take us back to 1267. The world population at that time was approximately 400 million; it took until the early nineteenth century for the human population to exceed 10^9.

Answer 410 We expect individual i_4 to have eight great grandparents in generation b_3, which is impossible with population size 7. So in small populations there is more sharing of ancestors than in large populations.

Answer 411 Blue individuals have left no descendants in the present. Red individuals are ancestors of all extant individuals. In contrast to these universal ancestors, black individuals have left some descendants in the present. Notice that as you go back in time, partial ancestors go extinct leaving only universal ancestors and non-ancestors [41]. From the point of view of the present, this means that eventually we all have the same ancestors. So next time someone tells you she is a descendant of X, who lived a long time ago, you know there are only two possibilities: Either this is true, then it is true for everyone, or, alas, it is false.

Answer 412 Set up the directory

```
mkdir Descent
cd Descent
```

Run the simulation using

```
  drawGenealogy -D 0 -i 4 -p 7 -C -g 7 -t testFig.tex
```

Typeset the figure and view it

```
latex testFig
dvips testFig -o -q
gv testFig.ps &
```

Answer 413 Simulate the times to the most recent universal ancestor

```
bash firstUnivAnc.sh > firstUnivAnc.dat
```

where `firstUnivAnc.sh` is

```
for a in 10 20 50 100 200 500 1000
do
    echo -n $a ' '
    for b in $(seq 100);
    do
        drawGenealogy -g 50 -c -p $a
    done |
        grep comm |
        awk '{s+=$8;c++}END{print s/c}'
done
```

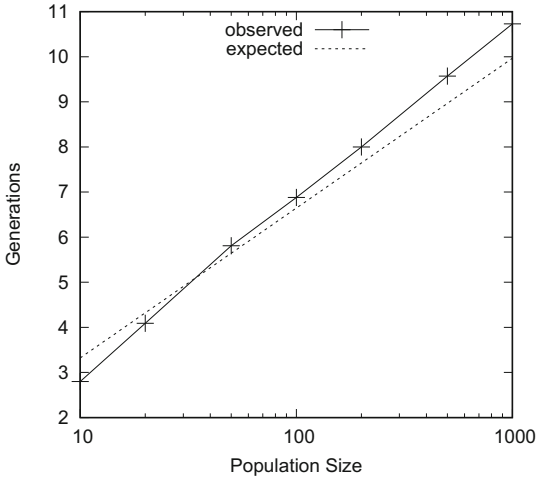
Plot the simulation results

```
gnuplot -p plot1.gp
```

where `plot1.gp` is

```
set xlabel "Population Size"
set ylabel "Generations"
set pointsize 2
set log x
set key top center
f(x) = log(x) / log(2)
plot [][] "firstUnivAnc.dat" title "observed" w linesp,\
f(x) title "expected" wi li
```

The resulting graph shows the number of generations until the appearance of the first universal ancestor as a function of population size.



The expectation fits the simulation quite well.

Answer 414 Compute times until all present-day individuals have identical ancestors:

```
bash allUnivAnc.sh > allUnivAnc.dat
```

where `allUnivAnc.sh` is

```
for a in 10 20 50 100 200 500 1000
do
    echo -n $a ' '
    for b in $(seq 100);
    do
        drawGenealogy -g 50 -c -p $a
    done |
        grep iden |
        awk '{s+=$5;c++}END{print s/c}'
done
```

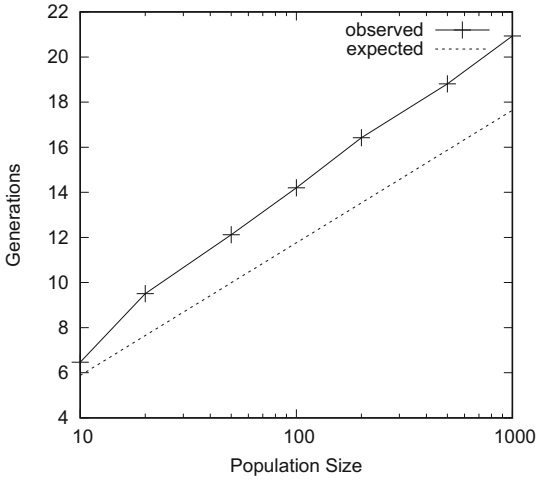
Plot the simulation together with the expectation

```
gnuplot -p plot2.gp
```

where `plot2.gp` is

```
set xlabel "Population Size"
set ylabel "Generations"
set pointsize 2
set log x
f(x) = 1.77 * log(x) / log(2)
plot [][] "allUnivAnc.dat" title "observed" wi linespoints,\
f(x) title "expected" wi li
```

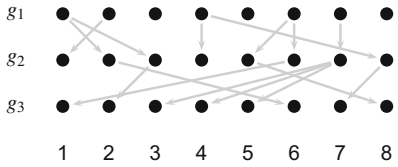
to get the number of generations until all present-day individuals have identical ancestors as a function of population size.



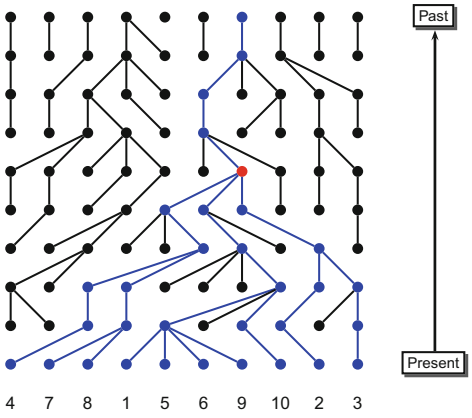
In this case, the expectation is distinct from the simulation, but at least similar.

Answer 415 The common ancestor of the two genes in i_4 lies beyond b_6. So in this uni-parental genealogy of genes, it takes much longer to reach the first common ancestors of genes than in the bi-parental genealogy of individuals.

Answer 416 Our extension of the Wright–Fisher model in Figure 6.4 looked like this; yours is bound to look different:



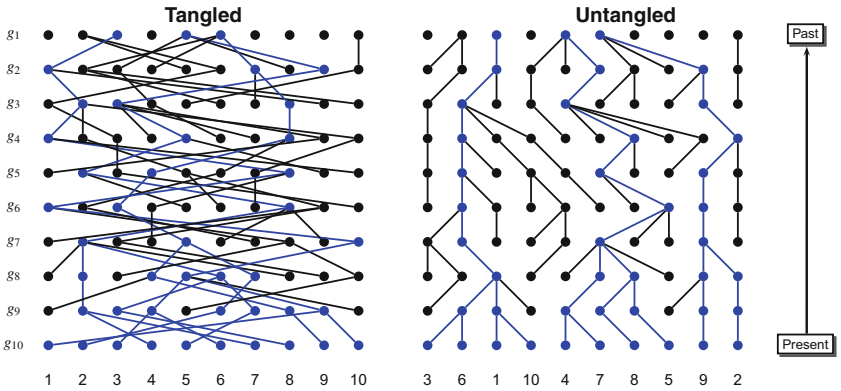
Answer 417 Yes, the Wright–Fisher simulation contains common ancestors, the most recent of which is marked in red:



Answer 418 The commands

```
drawWrightFisher -p 10 -t wrapWf.tex -a -1
latex wrapWf
dvips wrapWf -o -q
```

generate Wright–Fisher simulations like



Among ten iterations, we found four instances with a time to the most recent common ancestor of ten generations or less.

Answer 419 The probability of two genes picking a common ancestor is $1/N \times 1/N = 1/N^2$. Since there are N opportunities for picking the same ancestor (each gene has one ancestor), the probability of any two genes picking the same ancestor is $1/N$.

Answer 420 In ten iterations of

```
awk -f trace1.awk -v seed=$RANDOM -v N=10       |
sort                                            |
uniq                                            |
wc -l
```

we found no instance where all ten lineages remained.

Answer 421 Compute P_n for $N = 10$:

```
BEGIN{
    n = 10
    pn = 1
    for(i=1; i<n; i++)
        pn *= 1 - i / n
    print pn
}
```

which returns $Pn \approx 4 \times 10^{-4}$. So we carried out 10^4 iterations of the simulation in Problem 420

```
for a in $(seq 10000)
do
    awk -f trace1.awk -v seed=$RANDOM -v N=10  |
        sort                                   |
        uniq                                   |
        wc -l
done                                           |
    awk 'BEGIN{c=0}{if($1==10)c++}END{print c}'
```

and found $P_{\mathrm{n}} = 2 \times 10^{-4}$, which is close to the expected value.

Answer 422 We ran

```
awk -f trace2.awk -v seed=$RANDOM -v N=100 -v n=10
```

ten times and found three occasions where the number of lineages was reduced, that is, genes had picked common ancestors.

Answer 423 $P_{\mathrm{a}} = 90/2000 = 0.045$. The simulation might look like this

```
for a in $(seq 1000)
do
    awk -f trace2.awk -v seed=$RANDOM -v N=1000 -v n=10
done |
    awk '{if($1 < 10)s++;c++}END{print s/c}'
```

which gave us $P_{\mathrm{a}} = 0.048$, quite close to expected probability of an ancestor event.

Answer 424 Generate the number of lineages per generation

```
awk -f trace3.awk -v N=100 -v n=100 > trace3.dat
```

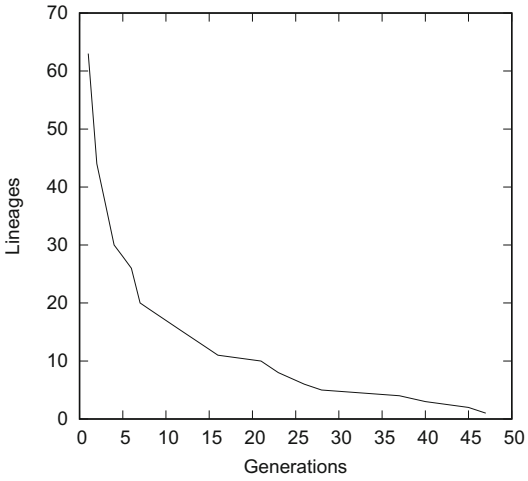
Plot the results

```
gnuplot -p plot3.gp
```

where `plot3.gp` is

```
set xlabel "Generations"
set ylabel "Lineages"
plot [][] "trace3.dat" t "" w l
```

to get



Answer 425 Commands like

```
awk -f trace3.awk -v seed=$RANDOM -v N=100 -v n=100
```

give widely varying times to the most recent common ancestor. Still, one might expect that it takes much longer for 100 lineages to find their common ancestor than for two. However, for individual runs of the simulation, times obtained with $n = 2$ are often quite similar as those with $n = 100$.

Answer 426 $n = 2$: N generations; $n \to \infty$: $2N$ generations, that is to say, the expected time to the most recent common ancestor, $E\{T_{\mathrm{MRCA}}\}$, varies by no more than a factor of 2.

Answer 427 Run

```
bash simTmrca.sh > simTmrca.dat
```

where `simTmrca.sh` is

```
for i in 2 5 10 20
do
    echo -n ${i} ' '
    for j in $(seq 100)
    do
        awk -f trace3.awk -v seed=$RANDOM -v N=100 -v n=$i |
              tail -n 1
    done |
        awk '{s+=$1;c++}END{print s/c}'
done
```

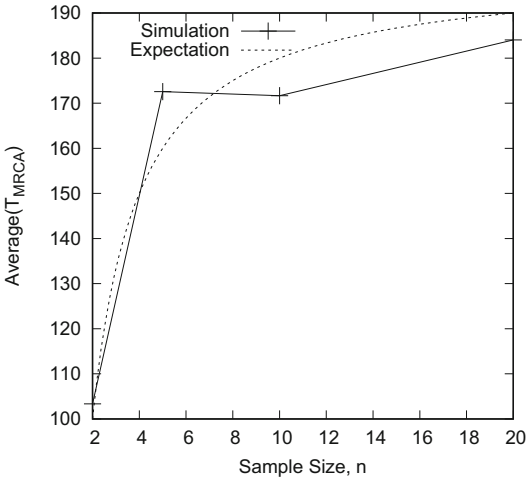
and plot the result

```
plot -p plot4.gp
```

where `plot4.gp` is

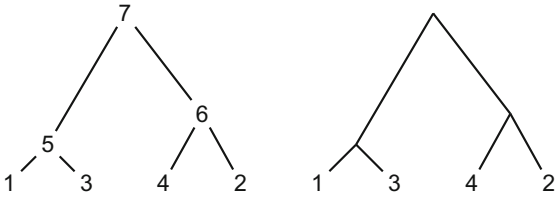
```
set xlabel "Sample Size, n"
set ylabel "Average(T_{MRCA})"
f(x) = 2*100*(1-1/x)
set pointsize 2
set key top left
plot [][] "simTmrca.dat" t "Simulation" w linespoints,\
f(x) t "Expectation" w l
```

to get



The simulated average time to the most recent common ancestor is quite similar to its expectation.

Answer 428 On the left is a example coalescent with all nodes labeled. It has four leaves, so it describes the genealogy of a sample of $n = 4$ genes:



By convention, only leaves are labeled in a coalescent, as shown on the right.

Answer 429 T_1 would begin with the most recent common ancestor, the root of the coalescent, and go on forever. It is not shown, because the coalescent is bounded by the most recent common ancestor.

Answer 430 By running this code with `i=4`, `i=3`, and `i=2`, we got $T_4 = 0.10$, $T_3 = 0.03$, $T_2 = 0.29$; so the coalescence times were as follows:

Index	1	2	3	4	5	6	7
Node	1	2	3	4	5	6	7
Time	0.00	0.00	0.00	0.00	0.10	0.13	0.42

Time to the most recent common ancestor is the time of the root node, $0.42 \times 2N$ generations.

Answer 431 Code like

```
for a in $(seq 100)
do
    awk -v seed=$RANDOM -f genCoalTimes.awk -v n=1000 |
        tail -n 1
done |
    awk '{s+=$2;c++}END{print s/c}'
```

gave average times to the most recent common ancestor of $T_{\mathrm{MRCA}} = 0.90$ for $n = 2$ and $T_{\mathrm{MRCA}} = 1.98$ for $n = 1000$. Recall from Problem 426 that when measured in $2N$ generations, we expect

$$T_{\mathrm{MRCA}} = \left(1 - \frac{1}{n}\right).$$

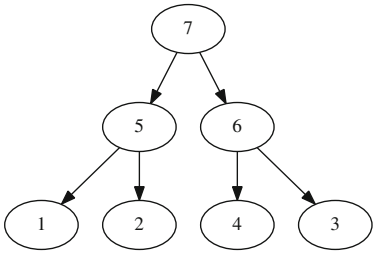
Answer 432 Here is a step-by-step depiction of shuffling, or put more formally, sampling without replacement; the values to be swapped are shown in bold. A particular position in the array might be picked repeatedly.

$r = 1, n = 5$	$r = 3, n = 4$	$r = 1, n = 3$	$r = 2, n = 2$	Result
1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5
1 2 3 4 **5**	5 2 **3** **4** 1	**5** 2 **4** 3 1	4 2 5 3 1	4 2 5 3 1

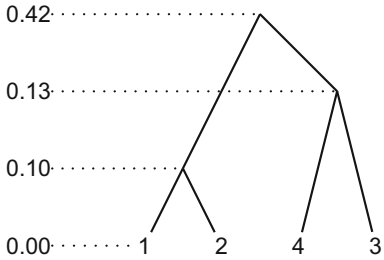
Answer 433 When continuing the construction of the coalescent, for parent 6 we got as first child indexes 1 and 1. So the first child of node 6 is the node at position 1, which is 4. Then, node 3 is placed at position 1, and is thus drawn as the second child of 6. Finally, the node at position 1 is replaced by node 6:

Index	1 2 3 4 5 6 7
Node	1 2 3 4 5 6 7
	4 5
	3
	6
Child1	1 4
Child2	2 3
Time 0	0 0 0

This leaves only two children for 7, 5, and 6; so the final topology is as follows:



Answer 434 Solutions are bound to differ; ours is



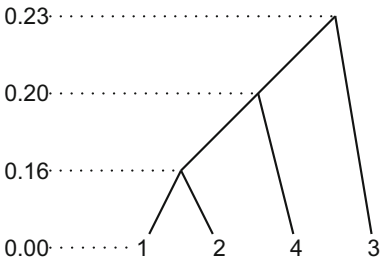
Answer 435 For a coalescent of sample size 4, the children of three internal nodes need to be picked:

```
awk -v seed=$RANDOM -v n=4 -f pickChildren.awk
# Pa C1   C2
5    1    2
6    1    2
7    1    1
```

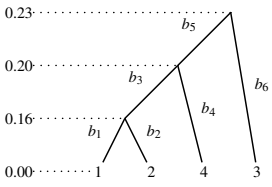
So we can fill in our table:

Index	1	2	3	4	5	6	7
Node	1	2	3	4	5	6	7
	4	5					
	3	6					
	6						
Child1					1	4	3
Child2					2	5	6
Time	0.00	0.00	0.00	0.00	0.16	0.20	0.23

This gives the tree



Answer 436 We label the branches on our coalescent and then draw the mutations:



Branch	Length	Mutations
b_1	0.16	0
b_2	0.16	1
b_3	0.04	0
b_4	0.20	0
b_5	0.03	0
b_6	0.23	3

Our tree has a total of four mutations. This is far fewer than the corresponding expectation value:

```
watterson -n 4 -t 10
S = 18.333333
```

Answer 437 Here is an example run to generate two samples:

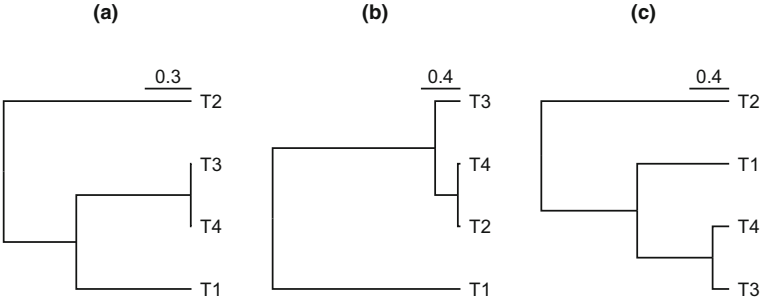
```
awk -f coalescent.awk -v seed=$RANDOM -v theta=10 -v
     sampleSize=4 -v numSamples=2
S= 28
S= 9
```

S is the number of mutations (segregating sites) found in a simulated sample.

Answer 438 The command

```
awk -f coalescent.awk -v seed=$RANDOM -v theta=10 -v
     sampleSize=4 -v numSamples=1 -v printTree=1 |
grep T          |
new2view
```

returns, for example



The coalescent time is measured in units of $2N$ generations. Accordingly, in coalescent (**a**) the scale refers to $0.3 \times 2N$ generations in a haploid population.

Answer 439 Our coalescent simulation looks like this:

```
ms 4 10000 -t 10  |
grep '^s'         |
awk '{s+=$2;c++}END{print s/c}'

18.32
```

The expectation was already computed in Problem 436, 18.33; this is very close to the simulated value.

Answer 440

```
ms 4 10000 -t 10  |
grep '^s'         |
awk '{if($2<=1)s++;c++}END{print "P="s/c}'
```

```
P=0.0102
```

Our test of the Wright–Fisher model rejects the null hypothesis with an error probability of just 1%. So the difference between observation and expectation is significant if we apply the common threshold of $\alpha = 0.05$.

Answer 441 First position:

```
tabix http://guanine.evolbio.mpg.de/problemsBook/chr19.mgp.
    vcf.gz 19:1-5000000 |
head -n 1

19  3078554
```

Last position:

```
tabix http://guanine.evolbio.mpg.de/problemsBook/chr19.mgp.
    vcf.gz 19:50000000-70000000 |
tail -n 1

19  61331223
```

Answer 442

```
tabix http://guanine.evolbio.mpg.de/problemsBook/chr19.mgp.
    vcf.gz 19:3078554-61331223 |
wc -l

1617408
```

That is, there are

$$\frac{1,617,408}{61,331,223 - 30,78,554 + 1} \approx 0.0278$$

SNPs per position. 17 mice have 34 chromosomes; we can compute the corresponding harmonic number $\sum_{i=1}^{33} 1/i$ using

```
watterson -t 1 -n 34

S = 4.088798
```

So the per nucleotide population mutation rate

$$\theta = \frac{0.0278}{4.0888} = 0.0068.$$

Answer 443 Count SNPs on mouse chromosome 19:

```
tabix http://guanine.evolbio.mpg.de/problemsBook/chr19.mgp.
    vcf.gz 19:5,189,001-5,190,000 |
wc -l

40
```

We get the θ for 1 kb from the per nucleotide value of 0.0068 and compute the expected number of SNPs:

```
watterson -t 6.8 -n 34

S = 27.803828
```

Significance

```
ms 34 10000 -t 6.8  |
grep '^s'           |
awk '{c++;if($2>=40)s++}END{print "P="s/c}'


P=0.1227
```

So the difference between theory and observation is not significant and we do not reject the null hypothesis, the Wright–Fisher model.

Answer 444 We search for *Plin5* in both data sets:

```
grep Plin5 all_a.txt > plin5_a.txt
grep Plin5 all_b.txt > plin5_b.txt
```

Answer 445 This can be solved in various ways, including mental arithmetic, here is our solution:

```
awk '{for(i=2;i<=NF;i++){s+=$i;c++}}END{print s/c}' plin5_a.txt
11.9303
```

and

```
awk '{for(i=2;i<=NF;i++){s+=$i;c++}}END{print s/c}' plin5_b.txt
12.947
```

Answer 446 We run the program

```
testMeans plin5_a.txt plin5_b.txt
Plin5 1.193e+01 1.295e+01 7.871e-03
```

The first two numbers are the averages we already computed, the third number is the P-value, that is, the error probability when rejecting the null hypothesis that the difference between the two samples is negligible. If we use the customary cutoff value of $\alpha = 0.05$, the difference is significant.

Answer 447 We aimed for very high precision and ran the Monte Carlo test with 10^6 iterations:

```
testMeans -t m -i 1000000 plin5_a.txt plin5_b.txt
Plin5 1.193e+01 1.295e+01 7.934e-03
```

This still varies between different runs, but it is quite close to the 7.871×10^{-3} computed previously.

Answer 448 We simulate the data

```
simNorm -m 12 -i 100 > experiment1.txt
simNorm -m 12 -i 100 > experiment2.txt
```

Then we compute the P values and count the cases where $P \leq 0.05$:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4<=0.05)print}'                  |
wc -l
6
```

In this case, the observed false-positive rate is $6/100 = 0.06$. This is close to the expected $\alpha = 0.05$, but is bound to vary between runs.

Answer 449 We ran

```
simNorm -m 12 -i 10000 > experiment1.txt
simNorm -m 12 -i 10000 > experiment2.txt
testMeans experiment1.txt experiment2.txt |
awk '{if($4<=0.05)print}' | wc -l
462
```

So the observed false-positive rate is $462/10000 = 0.0462$, which again is close to the expected 0.05.

Answer 450 We compute

$$1 - (1 - 0.05)^{100} = 0.994;$$

that is to say, there is a 99.4% chance of getting at least one false-positive when carrying out 100 hypothesis tests with $\alpha = 0.05$.

Answer 451 We use the same computation of the false-positive rate as before, except that this time we divide α by 10^4:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4<=0.05/10000)print}'            |
wc -l
0
```

In other words, after Bonferroni correction, we found not a single false-positive result, the type I error was all but eliminated (again, your result may differ slightly). We know that all samples were drawn from the same population, so this is the correct result.

Answer 452 Simulate the data

```
simNorm -m 6 -d 2.5 -i 10000 > experiment1.txt
simNorm -m 8 -d 2.5 -i 10000 > experiment2.txt
```

and determine the frequency with which the null hypothesis is *not* rejected:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4>0.05)print}'                   |
wc -l
6866
```

Since the samples were drawn from populations with different μ, every acceptance of the null hypothesis is a false-negative result: $\beta = 6866/10000 \approx 0.68$. In other words, the false-negative rate is large if the difference in means, also called the "effect size", is small.

Answer 453 Simulate the data

```
simNorm -m 6 -d 2.5 -i 10000 > experiment1.txt
simNorm -m 12 -d 2.5 -i 10000 > experiment2.txt
```

and again determine the frequency with which the null hypothesis is *not* rejected:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4 > 0.05)s++;c++}END{print "beta=" s/c}'
beta=0.0074
```

This means the false-negative rate drops dramatically when the effect size is increased.

Answer 454 Simulate the data

```
simNorm -m 6 -d 2.5 -i 10000 > experiment1.txt
simNorm -m 12 -d 3.5 -i 10000 > experiment2.txt
```

and compute the false-negative rate:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4 > 0.05)s++;c++}END{print "beta=" s/c}'
beta=0.0489
```

Answer 455 Apply the corrected α:

```
testMeans experiment1.txt experiment2.txt |
awk '{if($4 > 0.05/10000)s++;c++}END{print "beta=" s/c}'
beta=0.9811
```

This means, the Bonferroni correction leads to a large type II error rate and thereby obscures almost entirely the true difference between the two sets of experiments.

Answer 456 As before we count the nonsignificant P values:

```
testMeans experiment1.txt experiment2.txt |
sort -g -k 4                              |
awk '{if($4>c*0.05/10000)s++; c++}END{print "beta=" s/c}'
beta=0.0536
```

Here, the β is very close to $\delta = 0.05$, as it should be.

Answer 457 Simulate the data

```
simNorm -i 10000 -m 6 -d 2.5 > experiment1.txt
simNorm -i 10000 -m 6 -d 2.5 > experiment2.txt
```

and analyze them as before

```
testMeans experiment1.txt experiment2.txt           |
cut -f 4                                            |
sort -g                                             |
awk 'BEGIN{m=10000;d=0.05}{if($1<=NR*d/m)print $1}' |
wc -l
0
```

The type I error seems to have been removed as thoroughly as with the Bonferroni correction, but without the concomitant increase in type II error.

Answer 458 To test the effect of sample size, you could execute

```
bash sim.sh |
gnuplot -p plot.gp
```

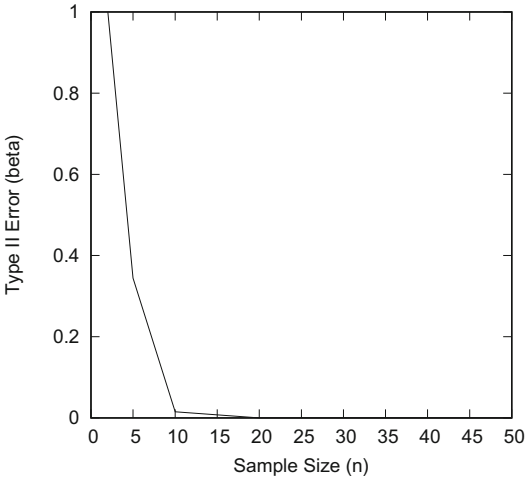
where `sim.sh` is

```
for a in 2 5 10 20 50 100 200 500
do
    echo -n $a ' '
    simNorm -i 10000 -m 6 -d 2.5 -n $a > experiment1.txt
    simNorm -i 10000 -m 7 -d 2.5 -n $a > experiment2.txt
    testMeans experiment1.txt experiment2.txt        |
    cut -f 4                                         |
    sort -g                                          |
    awk '{if($1<=NR*0.05/10000)c++}END{print 1-c/10000}'
done
```

and `plot.gp`

```
set xlabel "Sample Size (n)"
set ylabel "Type II Error (beta)"
plot "< cat" t "" w l
```

to get



The type II error decreases dramatically with increasing sample size. We say the power of the test grows. However, increasing the sample size beyond 20 does not improve its power substantially any more.

Answer 459 To obtain the type II error as a function of sample size, execute

```
bash sim2.sh |
gnuplot -p plot2.gp
```

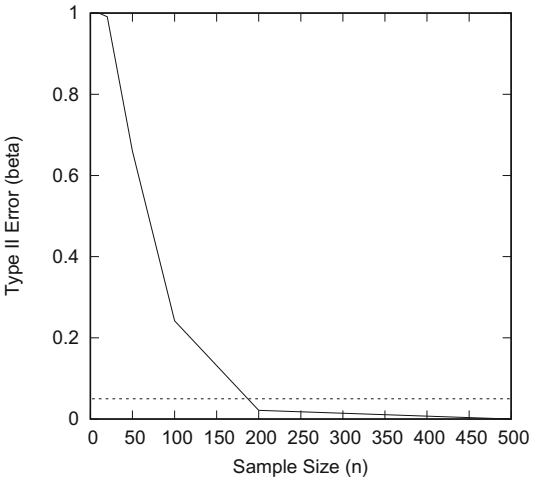
where `sim2.sh` is

```
for a in 2 5 10 20 50 100 200 500
do
    echo -n $a ' '
    simNorm -i 10000 -m 6 -d 2.5 -n $a > experiment1.txt
    simNorm -i 10000 -m 7 -d 2.5 -n $a > experiment2.txt
    testMeans experiment1.txt experiment2.txt                |
    cut -f 4                                                 |
    sort -g                                                  |
    awk '{if($1<=NR*0.05/10000)c++}END{print 1-c/10000}'
done
```

and `plot2.gp`

```
set xlabel "Sample Size (n)"
set ylabel "Type II Error (beta)"
f(x)=0.05
plot "< cat" t "" w l, f(x) t "" w l
```

to get



A sample size of $n \approx 200$ is now needed to obtain $\beta < 0.05$.

Answer 460 The number of experiments is the number of rows in `all_a.txt` or `all_b.txt`:

```
wc -l all_a.txt
25789 all_a.txt
```

The number of distinct genes is

```
cut -f 1 all_a.txt | sort | uniq | wc -l
14779
```

Answer 461

```
testMeans all_a.txt all_b.txt                    |
sort -k 4 -g                                     |
awk 'BEGIN{m=25789;d=0.1}{if($4<=NR*d/m)print}' |
cut -f 1                                         |
sort                                             |
uniq > genes.txt
```

The file `genes.txt` contains 209 distinct gene identifiers.

Answer 462 The highest red node is "Metabolic Process". The underlying study is concerned with the effect of fatty food on mice. Apparently, a change in food leads to a change in metabolism, which makes sense.

Answer 463 Do not forget to construct the new directory `RelationalDb` to keep your work on relational databases separate from the rest. The file `fatty_food.sql` contains

```
create table fatty_food(
   Sym varchar(18),
   M1 float,
   M2 float,
   M3 float,
   M4 float,
   M5 float,
   M6 float,
   M7 float,
   M8 float,
   primary key(Sym)
);
```

Notice that capital and lower case letters are not distinguished in SQL.

Answer 464 The file `fatty_food.sql` now contains

```
create table fatty_food(
   Sym varchar(18),
   M1 float,
   M2 float,
   M3 float,
   M4 float,
   M5 float,
   M6 float,
   M7 float,
   M8 float,
   primary key(Sym),
   foreign key (Sym) references normal_food(Sym)
);
```

Answer 465 The following commands construct the database

```
sqlite3 mouseExpress.db
.read normal_food.sql
.read fatty_food.sql
.separator "\t"
```

```
.import normal_food.txt normal_food
.import fatty_food.txt fatty_food
```

Notice that `normal_food` needs to be filled before `fatty_food`; otherwise, the foreign key constraint is violated.

Answer 466 Here is the required combination of `insert`, `select`, and `delete` commands:

```
insert into normal_food
values('toy_gene2',17.1, 9.5, 27.7, 6.5, 24.1, 30.2,
30.6, 14.3);
select * from normal_food where sym like 'toy_gene2';
delete from normal_food where sym like 'toy_gene2';
select * from normal_food where sym like 'toy_gene2';
```

Answer 467 If we enter

```
insert into normal_food
values('Plin5',17.1, 9.5, 27.7, 6.5, 24.1, 30.2,
30.6, 14.3);
insert into fatty_food
values('Plin5',17.1, 9.5, 27.7, 6.5, 24.1, 30.2,
30.6, 14.3);
```

we get the error messages

```
Error: near line 1: UNIQUE constraint failed: normal_food.sym
Error: near line 4: UNIQUE constraint failed: fatty_food.sym
```

Answer 468 We type

```
insert into fatty_food
values ('toy_gene3',3.4, 8.0, 4.4, 26.7, 8.6, 26.6, 4.8, 20.5);
```

and get the error message

```
FOREIGN KEY constraint failed
```

as there is no entry for `toy_gene3` in `normal_food`. If you do not get this error, you probably did not enter

```
PRAGMA foreign_keys = ON;
```

in this database session.

Answer 469 We write

```
.read insert.sql
```

where `insert.sql` contains the same commands we entered interactively, and so we get the same result.
To enter one value too may or too few, we can use

```
-- Insert too few values
insert into normal_food
values('toy_gene3',17.1, 9.5, 27.7, 6.5, 24.1, 30.2, 30.6);
-- Insert too many values
insert into normal_food
values('toy_gene3',17.1, 9.5, 27.7, 6.5, 24.1, 30.2, 30.6,
     14.3, 16.7);
```

to trigger the error messages

```
Error: near line 2: table normal_food has 9 columns but 8
    values were supplied
Error: near line 6: table normal_food has 9 columns but 10
    values were supplied
```

Notice that comments are marked by `--` in SQL.

Answer 470 Counting in SQL:

```
select count(*) from normal_food;
14779
select count(*) from fatty_food;
14779
```

Answer 471

```
select sym, (m1+m2+m3+m4+m5+m6+m7+m8)/8 from normal_food
   limit 3;
Nppa    6.37625
Gm12689 6.791625
Hvcn1   6.951
```

Answer 472 Compute the maximum

```
select sym, max((m1+m2+m3+m4+m5+m6+m7+m8)/8) from normal_food;
Pigt    19.068
```

the minimum

```
select sym, min((m1+m2+m3+m4+m5+m6+m7+m8)/8) from normal_food;
Trpv3   5.817125
```

and, finally, the average

```
select avg((m1+m2+m3+m4+m5+m6+m7+m8)/8) from normal_food;
8.49976156201365
```

Answer 473 Compute the maximum

```
select sym, max((m1+m2+m3+m4+m5+m6+m7+m8)/8) from fatty_food;
Pigt    19.068
```

the minimum

```
select sym, min((m1+m2+m3+m4+m5+m6+m7+m8)/8) from fatty_food;
Trpv3   5.823
```

and the average

```
select avg((m1+m2+m3+m4+m5+m6+m7+m8)/8) from fatty_food;
8.50960544522628
```

Answer 474

```
select normal_food.sym,
       (normal_food.m1 + normal_food.m2 +
       normal_food.m3 + normal_food.m4 +
       normal_food.m5 + normal_food.m6 +
       normal_food.m7 + normal_food.m8) / 8,
       (fatty_food.m1 + fatty_food.m2 +
       fatty_food.m3 + fatty_food.m4 +
       fatty_food.m5 + fatty_food.m6 +
       fatty_food.m7 + fatty_food.m8) / 8
from normal_food join fatty_food using (sym);
```

During construction of this command, you might have restricted its output by adding

```
limit 10
```

at the end. However, for the next Problem we need the full output, so make sure `join.sql` is saved as shown above.

Answer 475

```
sqlite3 mouseExpress.db < join.sql |
tr '|' '\t'                        |
awk -f fc.awk
Hsd3b5 1.3583
```

Write the AWK program `fc.awk` to find the gene with the largest fold change. where `fc.awk` contains

```
BEGIN{
    max = -1
}
{
    if($2 > $3 && $3 > 0)
        fc = $2 / $3
    else if($2 > 0)
        fc = $3 / $2
    else
        fc = -1
    if(fc > max){
        max = fc
        sym = $1
    }
}
END {
    print sym, max
}
```

Answer 476

```
sqlite3 mouseExpress.db < join.sql | tr '|' '\t' > avg.txt
```

Answer 477 Here is our solution:

```
import java.sql.*;
public class MouseExpressDb2{
    public static void main( String args[] ){
       String query = "select * from normal_food join
           fatty_food using(sym)";
       double fc, a1, a2, max;
       String name = null;
       try{
           Class.forName("org.sqlite.JDBC");
           Connection c = DriverManager.getConnection("jdbc:
               sqlite:mouseExpress.db");
           Statement stmt = c.createStatement();
           ResultSet rs = stmt.executeQuery(query);
           max = -1;
           while(rs.next()){
               a1 = a2 = 0.0;
               for(int i=2;i<=9;i++){
                  a1 += rs.getFloat(i);
                  a2 += rs.getFloat(i+8);
               }
               a1 /= 8;
               a2 /= 8;
               if(a1 > a2 && a2 > 0)
                  fc = a1 / a2;
              else if(a1 > 0)
                  fc = a2 / a1;
              else
                  fc = -1;
              if(fc > max){
                  max = fc;
                  name = rs.getString(1); // Access column #1,
                       i.e.sym, a string
              }
           }
           System.out.printf("%s\t%.3f\n", name, max);
       }catch(Exception e){
             System.err.println(e.getClass().getName() + ": "
                  + e.getMessage());
             System.exit(0);
       }
    }
}
```

When we run this, we get the same result as with `fc.awk`

```
java -cp sqlite-jdbc-3.15.1.jar:. MouseExpressDb2
Hsd3b5  2.675
```

Answer 478 At the time of writing ENSEMBL consisted of

```
mysql -h ensembldb.ensembl.org -u anonymous -e "show databases" |
tail -n +2                                                      |
wc -l
6346
```

Answer 479 At the time of writing the latest mouse `core` database in ENSEMBL was

```
mysql -h ensembldb.ensembl.org -u anonymous -e "show databases" |
grep mus_musculus_core                                          |
tail -n 1
mus_musculus_core_88_38
```

Answer 480

```
mysql -h ensembldb.ensembl.org -u anonymous -D
    mus_musculus_core_88_38 -e "show tables" |
tail -n +2 |
wc -l
73
```

Answer 481 Enter

```
mysql ... -e "describe seq_region"
```

To get

```
+-----------------+------------------+------+-----+---------+----------------+
| Field           | Type             | Null | Key | Default | Extra          |
+-----------------+------------------+------+-----+---------+----------------+
| seq_region_id   | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name            | varchar(255)     | NO   | MUL | NULL    |                |
| coord_system_id | int(10) unsigned | NO   | MUL | NULL    |                |
| length          | int(10) unsigned | NO   |     | NULL    |                |
+-----------------+------------------+------+-----+---------+----------------+
```

In `mysql`, “attributes” are called “Field”, and hence the first column of this table is the one most relevant for us.

Answer 482 We pipe the command given in the problem through

```
awk '{s+=$2}END{print s}'
```

to get a genome length of 2,725,521,370 bp.

Answer 483 The command

```
mysql ... -e "describe exon"
```

gives

```
+------------------+----------------------+------+-----+---------+----------------+
| Field            | Type                 | Null | Key | Default | Extra          |
+------------------+----------------------+------+-----+---------+----------------+
| exon_id          | int(10) unsigned     | NO   | PRI | NULL    | auto_increment |
| seq_region_id    | int(10) unsigned     | NO   | MUL | NULL    |                |
| seq_region_start | int(10) unsigned     | NO   |     | NULL    |                |
| seq_region_end   | int(10) unsigned     | NO   |     | NULL    |                |
| seq_region_strand| tinyint(2)           | NO   |     | NULL    |                |
| phase            | tinyint(2)           | NO   |     | NULL    |                |
| end_phase        | tinyint(2)           | NO   |     | NULL    |                |
| is_current       | tinyint(1)           | NO   |     | 1       |                |
| is_constitutive  | tinyint(1)           | NO   |     | 0       |                |
| stable_id        | varchar(128)         | YES  | MUL | NULL    |                |
| version          | smallint(5) unsigned | YES  |     | NULL    |                |
| created_date     | datetime             | YES  |     | NULL    |                |
| modified_date    | datetime             | YES  |     | NULL    |                |
+------------------+----------------------+------+-----+---------+----------------+
```

where again the first column contains the information we are looking for.

Answer 484 Compute the number of nucleotides contained in exons:

```
mysql ... -e "select sum(seq_region_end - seq_region_start + 1)
   from exon"
+-----------------------------------------+
| sum(seq_region_end - seq_region_start + 1) |
+-----------------------------------------+
|                                 168950237   |
+-----------------------------------------+
```

Then, divide this number by the total genome length

$$\frac{168,950,237}{2,725,521,370} \times 100 \approx 6.2\%$$

6.2 % of the mouse genome is covered by exons.

Answer 485 For this, we need the attribute `is_constitutive`:

```
mysql ... -e "select sum(seq_region_end - seq_region_start + 1)
   from exon where is_constitutive=1"

+-----------------------------------------+
| sum(seq_region_end - seq_region_start + 1) |
+-----------------------------------------+
|                                  38483186 |
+-----------------------------------------+
```

Hence, $38,483,186/2,725,521,370 \times 100 \approx 1.4\%$ of the mouse genome is covered by constitutive exons.

Answer 486 We use

```
select display_label, description
from xref
where display_label like 'Hsd3b5'
```

to learn that *Hsd3b5* is a hydroxy-delta-5-steroid dehydrogenase, 3 beta- and steroid delta-isomerase 5.

Answer 487 To find the `stable_id` of *Hsd3b5*, we can use

```
select display_label, gene.stable_id
from xref join gene on display_xref_id = xref_id
where display_label like  'Hsd3b5'
```

and get the gene `ENSMUSG00000038092`. All mouse genes have a `stable_id` of the form `ENSMUSG...`

Answer 488 Join the `gene` and `transcript` tables via `gene_id` and filter for ENSMUSG00000038092:

```
select gene.stable_id, transcript.stable_id
from gene join transcript using(gene_id) where
     gene.stable_id like 'ENSMUSG00000038092'
```

which gives ENSMUST00000044094, that is one transcript. All mouse transcripts have a `stable_id` of the form `ENSMUST...` You might have been tempted to join the two tables via `stable_id`, which is another attribute common to `gene` and `transcript`. However, the `stable_id`s in `gene` have the format `ENSMUSG...`, while we already saw that the `stable_id`s of transcripts are `ENSMUST...` Connecting the two tables is thus only possible via `gene_id`.

8.2 Appendix: UNIX Guide

This is a brief summary of the most essential UNIX commands. It is meant to be read next to a computer running the UNIX operating system, so that readers can experiment. For further reading, we recommend the system's online documentation and [1].

8.2.1 File Editing

Of the many text editors available for UNIX systems, we recommend `emacs`, as it comes with a standard graphical user interface for casual use. At the same time, it is a powerful and versatile tool used by many IT professionals. As a result, it is available on many systems. It is started by typing

```
emacs &
```

This opens a window with standard menus running `emacs`. A new file is opened by clicking on the corresponding icon. It is then edited using standard keyboard input and mouse moves. In addition, `emacs` comes with a rich set of keyboard shortcuts, called "key bindings", making its use much more efficient than with these traditional techniques. Table 8.1 lists the key bindings we use regularly in our own work. The table also illustrates the principle of creating different versions of commands through alternate use of the control key and the meta key. Commands referring to sentences only work if a full stop is followed by at least two blanks. Two of these key combinations are special: `C-x` and `M-x`. `C-x` is a prefix for other key combinations, and we have listed the ones we find most useful in Table 8.2. `M-x` is also a prefix for further commands, but these are called by extended names like `calendar` rather than one or two characters. Again, Table 8.2 lists our favorites. The full list of key bindings can be accessed by `C-h b`. The best place to start using the key bindings is the "Emacs Tutorial" opened by `C-h t`.

8.2.2 Working with Files

The many things we need to do with text files on a regular basis include viewing them, measuring their size, finding patterns in them, and so on. Table 8.3 lists some of the commands to carry out these routine tasks. It is a good idea to learn at least some of them by heart, as they are used constantly when working with the UNIX command line.

UNIX commands tend to come with numerous options. These are documented in the manual pages, which are accessed using the program `man`; for example,

```
man ls
```

This invokes a text viewer responsive to some of the same key bindings for cursor movement as `emacs`, e.g., `C-v` to scroll down and `M-v` to scroll up. Inside `man`, `h` invokes help and `q` quits.

8.2.3 Entering Commands Interactively

Any command entered at a command prompt is interpreted by a piece of software called the "shell", which runs inside a terminal window. There are different kinds of shells and the command

```
echo $SHELL
```

returns the active shell. The following description applies to the `bash` shell. If it is not already running, it can be started by entering

```
bash
```

Table 8.1 Paired `emacs` key bindings (shortcuts)

Key	Binding	Key	Binding
`C-a`	Move to beginning of line	`M-a`	Move to beginning of sentence
`C-b`	Move backward one character	`M-b`	Move backward one word
`C-d`	Delete character	`M-d`	Delete word to the right
—	—	`M-BACKSP`	Delete word to the left
`C-e`	Move to end of line	`M-e`	Move to end of sentence
`C-f`	Forward one character	`M-f`	Forward one word
`C-g`	Keyboard quit	—	—
`C-h`	Help	`M-h`	Mark paragraph
`C-k`	Delete line	`M-k`	Delete sentence
`C-l`	Center buffer on current line	`M-l`	Lower case word
`C-n`	Next line	—	—
`C-p`	Previous line	—	—
—	—	`M-q`	Layout paragraph
`C-r`	Search backward	`M-r`	Move to top/bottom of window
`C-s`	Search forward	—	—
`C-t`	Transpose characters	`M-t`	Transpose words
—	—	`M-u`	Upper case word
`C-v`	Scroll up	`M-v`	Scroll down
`C-w`	Delete selection	`M-w`	Copy selection
`C-x`	Command prefix (Table 8.2)	`M-x`	Execute extended command (Table 8.2)
`C-y`	Paste	—	—
`C-z`	Suspend frame	`M-z`	Delete to character
`C-_`	Undo	—	—
`C-SPC`	Set mark	—	—
`C-+`	Increase font size	—	—
`C-`	Decrease font size	—	—
—	—	`M-<`	Move to beginning of buffer
—	—	`M->`	Move to end of buffer

The command line completes prefixes of commands and file names in response to pressing TAB once if the prefix is unique. Otherwise, by pressing TAB a second time, the list of possible completions is presented. The most effective way of interacting with the command line is to let the autocompletion do as much work as possible by carefully mixing typing and tabbing. With a little practice this becomes second nature.

Like `man`, the shell is responsive to the same basic key bindings as `emacs`, which is an added benefit from learning them.

Table 8.2 A selection of frequently used composite commands in `emacs`

Key	Binding
`C-x b`	Switch buffer
`C-x k`	Kill buffer
`C-x o`	Switch to other buffer
`C-x C-c`	Quit
`C-x C-f`	Find file
`C-x C-s`	Save buffer
`C-x C-w`	Write file
`C-M-\`	Indent region
`M-x g`	Go to line
`M-x help`	Start help menu
`M-x shell`	Run shell in `emacs` buffer
`M-x count-words`	Count lines, words, and characters
`M-x rename-buffer`	Rename the current buffer
`M-x calendar`	Start calendar

8.2.4 Combining Commands: Pipes

UNIX commands such as those listed in Table 8.3 can be combined into programs by using the output from one command as the input for another. To do this, individual commands are combined via *pipes* denoted by a vertical line, "|". For example, to count the number of files,

```
ls | wc -l
```

where the output of `ls` is the input of `wc`.

The shell can also expand file names. Hence,

```
ls *.txt | wc -l
```

returns the number of files in the working directory that end in `.txt`.

8.2.5 Redirecting Output

By default, the result of a command is printed to the screen standard output stream called stdout. This usually corresponds to the screen. Output can be redirected to a file by writing, for example,

```
ls > tmp
```

which creates the file `tmp` (check using `ls`) and writes the results of the command `ls` into it (check using `cat tmp`). The simple redirection command, `>`, overwrites

Table 8.3 Commands for working with files

Command	Explanation
`cat`	Print (conCATenate) to screen
`-n`	Print numbered lines
`-b`	Print non-blank lines
`-s`	Squeeze blank lines
`cp` *file1 file2*	Copy *file1* to *file2*, overwrite old *file2* if it exists
`cp` *file1 file2 toDir*	Copy *file1* and *file2* to directory *toDir*
`cut -f n`	Cut the *n*th field
`diff` *fromFile toFile*	Find differences between *fromFile* and *toFile*
`grep` *pattern*	Print lines matching *pattern*
`-v`	Print lines not matching
`-A n`	Print matching lines and *n* lines after
`-B n`	Print matching lines and *n* lines before
`head` *filename*	Print first 10 lines of file
`-n n`	Print first *n* lines
`join` *file1 file2*	Join two sorted files on 1st column
`less`	Pager for viewing text
`ls`	List names of all files in current directory
`-l`	Long listing for more information
`-r`	List in reverse order
`-t`	List in time order, most recent first
`-u`	List by time last used
`mv` *file1 file2*	Move *file1* to *file2*, overwrite old *file2* if it exists
`rm` *filenames*	Remove named files, irrevocably
`-r`	Remove recursively directories and their contents
`rmdir` *directory*	Remove named directory
`sort`	Sort files alphabetically by line
`-n`	Sort numerically
`-k n`	Sort by column *n*
`-r`	Reverse sort
`-R`	Randomize
`tac`	Reverse lines of named files
`tail`	Print last 10 lines of file
`-n`	Print last *n* lines
`+n n`	Start printing file at line *n*
`uniq` *filenames*	Filter out repeated lines in sorted input
`-c`	Count repeated lines
`-d`	Print duplicated lines
`-u`	Print unique lines
`wc`	Count lines, words, and characters
`-l`	Count lines
`-L`	Return length of the longest line

the original content of the target file. The variant >> appends to whatever is already in the file:

```
ls >> tmp
```

The redirection can also go the other way:

```
cat < tmp
```

This writes the contents of `tmp` to the standard input, from where they can be read by `cat`.

8.2.6 Shell Scripts

Any command entered on the command line can also be submitted to the system from a text file called a "shell script". Suppose the file `ls.sh` contains a single line,

```
ls
```

This can be executed by passing it to the `bash`:

```
bash ls.sh
```

Alternatively, `ls.sh` can be made executable

```
chmod +x ls.sh
```

and then run

```
./ls.sh
```

Shell scripts can contain do loops and conditional statements. Say, a set of sequence files with names of the form `fileName.txt` need changing to `fileName.fasta`. This script generates ten example files:

```
for f in 1 2 3 4 5 6 7 8 9 10
do
    echo '>s'${f} > s${f}.txt
    echo 'ACCGT' >> s${f}.txt
done
```

The variable `f` takes the values of the list to the right of `in`. The value of `f` is referenced by writing it in curly brackets and prefixing it with `$`. The command `echo` prints its argument. These files are in *fasta* format, a header line starting with > followed by one or more lines of sequence data.

Instead of explicitly specifying a sequence of numbers, `seq` can be used:

```
for f in $(seq 10)
```

To change the file extensions from `.txt` to `.fasta`, we construct the script `rename.sh`, which allows us to write

```
bash rename.sh *.txt
```

where `rename.sh` is

```
for f in $@
do
    mv ${f}  ${f%txt}fasta
done
```

Notice the `for`-construct:

```
for f in $@
```

which loops over every argument on the command line, in our case all files ending with `.txt`. Further, in the construct

```
${f%txt}fasta
```

the suffix `txt` of the file name is deleted with the `%` operator and then replaced by `fasta`.

8.2.7 Directories

Directories may contain files and other directories. The UNIX file system is thus hierarchical and can be depicted as a tree of directories. Figure 8.1 shows a portion of this tree. There are three special types of directories:

1. Root directory: The most basic directory situated at the root of the file system (Fig. 8.1). Its name is `/`
2. Home directory: The directory accessed after logging in. It is the only directory in which a user can create files and directories.
3. Working directory: The directory a user is currently working in.

The full name of a directory such as

```
/home/tom
```

is known as its *path*. A forward slash (`/`) can therefore either refer to the root directory or function as a delimiter of directory names. A variation of the name of the root directory, `~/`, refers to the user's home directory. Table 8.4 summarizes the essential commands for working with directories.

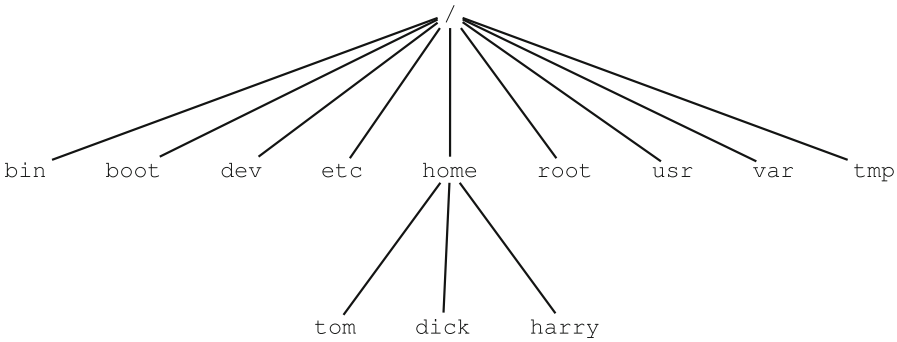


Fig. 8.1 A sample of the directories that make up a typical UNIX file system

Table 8.4 Commands for working with UNIX directories

Command	Explanation
`pwd`	Print working directory
`cd`	Return to home directory
`cd` *directoryname*	Change to the named directory
`cd ..`	Move up one level in the file system
`mkdir` *directoryname*	Make the named directory
`rmdir` *directoryname*	Remove the named directory

```
a 10
b 11
c 12
d 15
e 11
```

Fig. 8.2 Sample data contained in the file `data.dat`

8.2.8 Filters

Programs for filtering textual data are the bread and butter of practical bioinformatics. The following sections introduce four of the most popular filters: `grep`, `tr`, `sed`, and `awk`. All three make use of a common notation for specifying patterns in strings. Such patterns are called "regular expressions". We give examples of them when dealing with the individual filters and explain them in more detail afterward.

`grep`

Consider the data shown in Fig. 8.2, and say we wanted to extract all lines matching 11. The command

```
man ls
```

where `data.dat` contains the listing in Fig. 8.2 would return

```
b 11
e 11
```

With `-v` the lines *not* matching are printed:

```
grep -v 11 data.dat
a 10
c 12
d 15
```

`Grep` can also be used to search for more complex patterns specified as regular expressions. For example, `[25]` matches "2" or "5":

```
grep [25] data.dat
c 12
d 15
```

tr

The program `tr` is used to translate or delete characters. For example, the command

```
tr -d '\n' < data.dat
```

removes all line breaks. Notice that unlike many UNIX tools, `tr` does not take file names as arguments. Hence, the < notation for reading from `data.dat`. Alternatively, we could have written

```
cat data.dat | tr -d '\n'
```

To "translate" tabulators into newlines, we enter

```
tr '\t' '\n' < data.dat
```

Or to convert the line labels to upper case, we can write

```
tr a-z A-Z < data.dat
```

where `a-z` is a character range indicating any lower case letter. Number ranges are coded similarly, which would allow us to express numbers as characters:

```
tr 0-9 a-z < data.dat
```

A biologically more relevant translation would be to encode `A` or `G` as purine (`R`) and `C` or `T` as pyrimidine (`Y`):

```
echo ACGT | tr AG R | tr CT Y
```

sed

Emacs is an interactive editor. In contrast, sed is a noninteractive, "stream" editor. Perhaps, its simplest operation is to delete a single line, say the second line:

```
sed '2d' data.dat
```

Instead of deleting the second line, it can be printed

```
sed '2p' data.dat
```

By default sed applies its pattern to every line it encounters and prints all other lines unchanged. So in the previous example, the line

```
b 11
```

was printed twice. The option -n restricts the output to the matching lines:

```
sed -n '2p' data.dat
```

It is also possible to print line ranges:

```
sed -n '2,4p' data.dat
```

A replacement for head would be

```
sed -n '1,10p' data.dat
```

Like directories in paths, regular expressions are delineated by pairs of /

```
sed '/b/p' data.dat
```

So

```
sed -n '/b/p' data.dat
```

is an alternative to

```
grep b data.dat
```

The complement version of grep,

```
grep -v b data.dat
```

becomes in sed

```
sed '/b/d' data.dat
```

Apart from the print and delete operations, which in practice are often dealt with using grep, sed can carry out substitutions, which are beyond the capabilities of grep; for example,

```
sed 's/1/9/' data.dat
```

substitutes the first occurrence of a "1" by a "9" in each line. The *global* version of this command replaces all occurrences of "1" by "9",

```
sed 's/1/9/g' data.dat
```

The regular expression first encountered with `grep`, `[25]`, which specifies 2 *or* 5, can also be used in a substitution

```
sed 's/[25]/X/g' data.dat
```

AWK

AWK mixes a programming language with the line- and pattern-based paradigm of `grep` and `sed`. The following exposition is adapted from the original description of the language by its authors [4, Appendix A]. An AWK program is executed as

```
awk 'program' <file>
```

or

```
awk -f program.awk <file>
```

Each program has the structure

```
pattern {action}
```

The pattern is evaluated for each input line and if true, the statements in the action block are executed. Table 8.5 lists the most common patterns. Expressions and regular expressions can be combined with the logical operators `&&` (and), `||` (or), and `!` (not). Actions are specified through sequences of statements, some of which are listed in Table 8.6.

Statements are separated by newlines or semicolons; lines starting with `#` are comments. The most common input and output functions are listed in Table 8.7. One of these, `printf`, produces formatted output. Formatting is done via the format conversion commands listed in Table 8.8. Apart from `printf`, which prints to the screen (stdout), these are also recognized by `sprintf`, which prints to a string.

AWK has a number of built-in variables (Table 8.9). Of these, `NF`, the number of fields is particularly useful, as it allows traversal of all fields in a line, as in

```
for(i=1; i<=NF; i++)
   print $i
```

Table 8.5 Patterns in AWK

Pattern	Meaning
`BEGIN`	True before any input lines are processed
`END`	True after all input lines have been processed
Expression	Any expression in the AWK language
/regular expression/	True if matched
Pattern, pattern	Pattern range; true if in range

Table 8.6 AWK actions

Action	Meaning
`delete` *array element*	Delete specific entry from an array
`exit`	Terminate program
`if`(*expression*) *statement* [`else` *statement*]	Conditional execution
input–output statement	See Table 8.7
`for`(*expression*; *expression*; *expression*) *statement*	Repeat a fixed number of times
`for`(*variable* `in` *variable*) *statement*	Iterate over the keys of a hash
`while`(*expression*) *statement*	Execute while true
{*statement*}	Statements are grouped by curly brackets

Table 8.7 Input and output in AWK

Action	Meaning
`close`(*fileOrPipe*)	Close file or pipe
command `|` `getline`	Pipe into `getline`
`print`	Print current line
`printf` *fmt*, *expr-list*	Print formatted output
`system`(*cmd*)	Send *cmd* to the shell for execution

Table 8.8 Formatting for `printf` and `sprintf`

Command	Meaning
`%c`	Character
`%d`	Decimal number
`%e`	Engineering convention, `[-]d.dddddde[-+]dd`
`%f`	Floating point number, `[-]d.dddddd`
`%g`	General: `%d`, `%f`, or `%e`, whichever is shorter
`%s`	String

Table 8.9 AWK built-in variables

Variable	Meaning
`ARGC`	Number of arguments on command line
`ARGV`	Array of arguments on command line, `ARGV[0..ARGC-1]`
`FILENAME`	Name of current input file
`FS`	Field separator
`NF`	Number of fields in current line
`NR`	Number of records (= lines)

Table 8.10 AWK string manipulation functions; s and t: strings, r: regular expression, i and n: integers

Function	Meaning
`gsub(`r, s, t`)`	Globally substitute r by s in t
`index(`s, t`)`	Return first starting position of t in s or 0 for no match
`length(`s`)`	Return length of s
`match(`s, r`)`	Return first starting position of r in s or 0 for no match
`split(`s, a, f`)`	Split s on f into fields saved in a, return number of fields; if f is omitted, use `FS`
`sprintf(`*fmt*, *expr-list*`)`	Return *expr-list* as a string formatted according to *fmt*
`sub(`r, s, t`)`	Like `gsub`, except that only first occurrence of r in t is replaced by s
`substr(`s, i, n`)`	Return n-char substring starting at i; if n is omitted, return suffix starting at i

Table 8.11 The arithmetic functions of AWK

Function	Meaning
`cos(`x`)`	$\cos(x)$
`exp(`x`)`	e^x
`int(`x`)`	Truncate x to integer
`log(`x`)`	Natural logarithm
`rand()`	Random number, r, $0 \leq r < 1$
`sin(`x`)`	$\sin(x)$
`sqrt(`x`)`	$\sqrt{x}$
`srand(`x`)`	Seed the random number generator with x; otherwise, x = current second

AWK is designed for manipulating strings and Table 8.10 lists its built-in string functions. Of these, `sprintf` has already been mentioned as allowing formatting.

AWK provides a selection of arithmetic functions, which are listed in Table 8.11.

Expressions are combined using the operators in Table 8.12. Perhaps, the most obscure—but in practice very useful—is string concatenation, which takes no explicit operator. For example, the code

```
a = "bio"          # assignment
b = "informatics"  # assignment
c = a b            # concatenation & assignment
print c
```

would print `bioinformatics`.

To conclude this brief exposition of AWK, we give a few examples. Consider again our sample data in Fig. 8.2. Print the second column:

```
awk '{printf "%d\n", $2 }' data.dat
```

Table 8.12 AWK operators

Operators	Meaning
`= += -= *= /= %= ^=`	Assignment
`?:`	Conditional expression
`||`	OR
`&&`	AND
`in`	Key in hash
`~ !~`	Regular expression match and its negation
`< <= > >= != ==`	Comparisons
	String concatenation without explicit operator
`+ -`	Addition, subtraction
`* /%`	Multiplication, division, modulo
`!`	NOT
`^`	To the power of
`++ --`	Increment, decrement, can be used in prefix and postfix notation
`$`	field (column)

which replicates the command

```
cut -f 2 data.dat
```

However, in contrast to `cut`, AWK can also manipulate the input data. For instance, it can sum over the entries in the second column:

```
awk '{s += $2}END{printf "sum: %d\n", s}' data.dat
```

And their average is computed as

```
awk '{c++; s += $2}END{printf "avg: %g\n", s/c}' data.dat
```

AWK arrays behave like hash tables with strings or numbers as keys. In order to compute the occurrence of distinct numbers in the second column of the input data file, write

```
awk '{s[$2]++}END{for(a in s)printf "%d\t%d\n", a, s[a]}'
data.dat
```

As a final example, consider

```
awk '/[25]/{print}' data.dat
```

as a replacement for our first example of a regular expression in `grep`,

```
grep [25] data.dat
```

Like the rest of UNIX, `awk` is described in its `man` pages. In addition, we have learned the language from the introduction by its authors, which is a model of clarity and usefulness [4].

8.2.9 Regular Expressions

The expression `[25]` we just used is an example of a regular expression, a notation for sets of strings; in this case, the set comprises the members "2" and "5". Another example is the dot (.), which matches any character, and hence refers to the set of all strings length one. As a rule, everything is text in UNIX, and as a consequence, regular expressions are used in many UNIX programs, not just the three examples we saw above, `grep`, `sed`, and AWK, but also in `emacs` and the shell. Knowing about regular expressions is thus very useful when working on UNIX systems.

There are three variants of regular expressions: regular, extended, and PERL (another programming language). In the following, we refer to the extended syntax, which is invoked by the `-E` switch in `grep` and the `-r` switch in `sed`. AWK regular expressions are by default of the extended kind. The sections "REGULAR EXPRESSIONS" in the `man` page for `grep` and "Regular Expressions" in the AWK `man` page contain more detail.

Character Classes

Character classes are written in square brackets. For example, `[ab]` matches either an `a` or a `b`. The complement of a character class is `[^ab]`, which matches anything but `a` or `b`. Some character classes are used so frequently; there is a standardized notation for referring to them. We list six such classes in Table 8.13. To find out how, say, the character class "digit" works, try

```
sed 's/[[:digit:]]/x/g' data.dat
```

Quantifiers

There are four different types of quantifiers in regular expressions (Table 8.14). They are all known as *greedy*, which means they maximize the number of matches. For

Table 8.13 Regular expression character classes

Class	Meaning	Code
`[:alpha:]`	Letters	`[A-Za-z]`
`[:digit:]`	Digits	`[0-9]`
`[:space:]`	Whitespace characters	`[  t n r f v]`
`[:cntrl:]`	Control characters	—
`[:graph:]`	Graphic characters	`[^[:cntrl:]]`
`[:print:]`	Printing characters	`[[:graph] ]`

Table 8.14 Quantifiers in regular expressions

Number of matches (x)	Expression
$x \geq 0$	`*`
$x \geq 1$	`+`
$m \leq x \leq n$	`{m,n}`
$m \leq x$	`{m,}`

Table 8.15 Anchors in regular expressions

Expression	Explanation
$\backslash b$	Word boundary
`^`	Beginning of line (except when used inside a character class)
`$`	End of line

example, the expression `.*` would match the entire line of text rather than stopping at the beginning of the line upon encountering the first "match".

Anchors

Anchors allow a position within a string to be referenced, and Table 8.15 lists the three most important ones.

Backreferences

Assigning values to variables is one of the most important operations in traditional programming languages. In regular expressions, backreferences provide an analogous mechanism, of which there are two kinds, depending on where the referencing is done:

- Inside regular expressions: \n, where $1 \leq n \leq \infty$;
- Outside of regular expressions: `$n`, where $1 \leq n \leq \infty$.

For example, to substitute any pair of identical digits by just a single occurrence of that digit, `sed` could be used:

```
sed -E 's/([0-9])(\1)/\1/g' data.dat
```

426995_1_En_8_Chapter.toc