Exercises in Algorithm Design

Roland Backhouse[*]

January 25, 2012

Abstract

This is a collection of questions in algorithm design which might be adapted for use in an examination.

1 Verification Conditions

Exercise 1 The following program has been written to calculate the quotient $M \div 9$ and the remainder $M \bmod 9$ of natural number M by repeatedly dividing by 10. (Dividing by 10 is much easier than dividing by 9 when numbers are given in decimal form.)

```
{ $0 \le M$ }
$q,r := 0, M{+}1$ ;
{ **Invariant:** $M{+}1 = 9{\times}q + r$ }
do $10 < r \rightarrow r,q := (r \bmod 10) + (r \div 10)\ ,\ q + (r \div 10)$
od ;
$r := r{-}1$
{ $0 \le r < 9 \wedge M = 9{\times}q + r$ } .
```

a) There is a "bug" in the test for terminating the loop. Construct the verification condition relating to this test in order to identify the bug. Suggest a correction to the test.
b) In the process of constructing the verification condition, you will find that you need to add a (fairly obvious) conjunct to the invariant. State clearly what the addition is.

[*]School of Computer Science and Information Technology, University of Nottingham, Nottingham NG8 1BB, England

The remaining parts of this question relate to the corrected version of the program.

c) Test the algorithm on the cases $M=1367$ and $M=25679$. Give details of the successive values of q and r.

d) A bound function has not been supplied. Suggest a suitable bound function.

e) Construct verification conditions that demonstrate the correctness of the program.

□

Exercise 2 a) Construct verification conditions for the following program. Assume that M and m are integers. Variables x, X and y may be integers or reals. The symbol "$\div$" denotes integer division.

```
{ $0 \le M$ }
$m,x,y := M,X,0$ ;
{ **Invariant:** $0 \le m \wedge m \times x + y = M \times X$
  **Bound function:** $m$ }
do $m \ne 0$ →    if   even.m → $m,x := m \div 2 , 2 \times x$
                  □   odd.m → $m,x,y := (m-1) \div 2 , 2 \times x , y-x$
                  fi
od
{ $y = M \times X$ } .
```

b) There is an error in the program. Use the verification conditions to identify the error and how to correct it.

□

2 Saddleback Search

The following exercises are all based on saddleback search.

Exercise 3 Given are two functions f and g, both of which map natural numbers to natural numbers. Suppose both functions are ascending. (That is, for all natural numbers i and j, $i \le j \Rightarrow f.i \le f.j$. Similarly for g.) It is known that there are numbers m and n such that $f.m = g.n$. Develop a program to find one pair of such numbers.

□

Exercise 4 Given are two arrays f and g, of lengths M and N, respectively. Both arrays store ascending sequences of integers. That is, for all indices i and j to the array f,

$$i \leq j \;\Rightarrow\; f[i] \leq f[j] \quad .$$

Similarly for g.

This question is about developing a program that will calculate

$$\langle \Downarrow i,j \;:\; 0 \leq i < M \wedge 0 \leq j < N \;:\; |f[i] - g[j]| \rangle \quad .$$

(Note: $|x|$ denotes the absolute value of x and $\Downarrow$ is the minimum quantifier.) Parts (a) and (b) are hints towards identifying an appropriate invariant.

a) Suppose that m and n satisfy $0 \leq m < M$ and $0 \leq n < N$, respectively. Suppose $f[m] - g[n] \geq 0$. Show that

$$\begin{array}{ll} & \langle \Downarrow i,j \;:\; m \leq i < M \;\wedge\; n \leq j < N \;:\; |f[i] - g[j]| \rangle \\ = & (f[m] - g[n]) \downarrow \langle \Downarrow i,j \;:\; m \leq i < M \;\wedge\; n{+}1 \leq j < N \;:\; |f[i] - g[j]| \rangle \quad . \end{array}$$

(Hint: split off the case $j = n$ in the quantification.)

b) What is the symmetric property when $g[n] - f[m] \geq 0$?

c) Use these two properties to develop a program that will calculate

$$\langle \Downarrow i,j \;:\; 0 \leq i < M \wedge 0 \leq j < N \;:\; |f[i] - g[j]| \rangle \quad .$$

Your program should use variables m, n and d satisfying the invariant property:

$$\begin{array}{ll} & \langle \Downarrow i,j : 0 \leq i < M \wedge 0 \leq j < N : |f[i] - g[j]| \rangle \\ = & d \downarrow \langle \Downarrow i,j : m \leq i < M \wedge n \leq j < N : |f[i] - g[j]| \rangle \quad . \end{array}$$

Your program should have time complexity proportional to $M{+}N$ (rather than $M{\times}N$).

Explain all steps in your design in sufficient detail that it is possible for the reader to formally verify the correctness of your program. In particular, state explicitly any properties of minimum that your design exploits. You should assume that M and N are nonzero. Take care over the initialisation of d.

□

Exercise 5 Given are two integers M and N satisfying $0 \leq M$ and $0 \leq N$. Construct a program to count the number of integers k and l satisfying

$$0 \leq k \leq N \ \wedge \ 0 \leq l \leq M \ \wedge \ M \times k = N \times l \ .$$

The running time of your program should be linear in $M+N$. (Hint: the function mapping k and l to $M \times k - N \times l$ is strictly increasing in k and strictly decreasing in l. Develop a suitably modified implementation of saddleback search.)

State precisely the postcondition of your program as well as the invariant property and bound function on which any loop in your program is based. Justify clearly every test and assignment in your program.

How would you modify the program so that all multiplications are replaced by additions?

□

3 Periodic Functions

Exercise 6 A function f from natural numbers to natural numbers is said to be *ultimately periodic* if there are numbers i and j such that $i < j$ and $f^i.0 = f^j.0$. (f^i denotes i applications of f. So, $f^0.0 = 0$ and $f^{k+1}.0 = f.(f^k.0)$.) The *period* of f is the smallest strictly positive number p such that, for some i, $f^i.0 = f^{i+p}.0$. (Random-number generators are examples of ultimately periodic functions.)

Formally, the period of an ultimately periodic function f is a solution of

$$p:: \quad 1 \leq p \wedge \langle \exists m \ :: \ \langle \forall i,j \ : \ i \leq j \ : \ f^i.0 = f^j.0 \ \equiv \ m \leq i \wedge p \backslash j - i \rangle \rangle \ .$$

The notation "$m \backslash n$" means "m divides n" (or "n is a multiple of m", specifically, $\langle \exists k \ :: \ n = k \times m \rangle$). (In mathematical vernacular, the existential quantification over the variable m would be expressed as "for all sufficiently large i and j".)

As a consequence, we have (for ultimately periodic f)

$$\langle \forall k \ : \ 1 \leq k \ : \ f^k.0 = f^{2 \times k}.0 \Rightarrow p \backslash k \rangle$$

(Instantiate i and j to k and $2 \times k$, respectively.)

Use this property to construct a program that, given ultimately periodic f, computes a multiple of the period of f. Your program should use only simple variables (thus, no arrays or other complex data structures).

□

Exercise 7 For any number N greater than 0, the decimal representation of $\frac{1}{N}$ is ultimately periodic. That is, it consists of an initial sequence $d_0, d_1, \ldots, d_{m-1}$ of digits followed by a sequence $d_m, d_{m+1}, \ldots, d_{n-1}$ that is repeated over and over. For example, $\frac{1}{4} = 0.25000\ldots$ is the initial sequence 25 followed by the sequence 0 repeated over and over ($m=2$ and $n=3$), while $\frac{1}{7} = 0.142857142857\ldots$ is the sequence 142857 repeated over and over ($m=0$ and $n=6$). This question is about constructing a program to determine the period $n-m$ using only simple variables (thus, no arrays or other complex data structures).

Consider, for example, the case of finding the decimal representation of $\frac{1}{7}$. Long-division can be used to check that $\frac{1}{7} = 0.142857142857\ldots$. (Check the calculation yourself.) At each iteration, a new digit is calculated together with a new remainder value. (For $\frac{1}{7}$ the successive values of the remainder are 1, 3, 2, 6, ...) Observe the value of the remainder at each step; it is a number that is less than 7, and hence the sequence of remainders will eventually repeat. In general, using long-division to compute $\frac{1}{N}$, the remainder at each step will eventually repeat.

The calculation of the digits of $\frac{1}{N}$ initialises remainder r to 1 and subsequently iterates execution of the assignment

$$r,d := (10\times r) \bmod N \, , (10\times r) \div N \quad .$$

Let $r.i$ denotes the value of r after the ith iteration. In addition, let $d.i$ denotes the value of d after the ith iteration. (So, $r.0 = 1$ and $r.(i{+}1) = (10 \times r.i) \bmod N$.). Also, $d.0$ is undefined, and $d.(i{+}1) = (10 \times r.i) \div N$.)

a) It is a fact that the period of the digits of $\frac{1}{N}$ is the same as the period of the remainder r. Formulate and prove a precise statement of this property.

b) Show that $r.N$ will eventually repeat. That is, show that

$$\langle \exists k : 0 < k : r.N = r.(N{+}k) \rangle \quad .$$

c) Making use of parts (a) and (b), construct a program that computes the period of the decimal representation of $\frac{1}{N}$ using only simple variables (thus no arrays or other complex data structures).

□

4 Euclid's Algorithm

The oldest known algorithm, Euclid's algorithm for computing greatest common divisors, is an excellent source of examination questions.

Throughout, m and n are natural numbers. We use the notation "$m\backslash n$" to mean "m divides n" (or "n is a multiple of m", specifically, $\langle\exists k :: n{=}k{\times}m\rangle$). Note that the divides relation is a partial ordering on numbers. (That is, it is reflexive, anti-symmetric and transitive.) The number 0 is the largest number in this ordering. (That is, $m\backslash 0$ for all m.)

Exercise 8 The greatest common divisor of numbers m and n, denoted $m \,\mathrm{gcd}\, n$, is defined by, for all numbers p,

(9) $\quad p\backslash(m \,\mathrm{gcd}\, n) \;\equiv\; p\backslash m \wedge p\backslash n$.

(We use an infix notation for the gcd operator because it is associative and symmetric; note that its unit is 0.)

The following program computes the greatest common divisor of positive natural numbers M and N using Euclid's algorithm.

```
{ $0<M \wedge 0<N$ }
$m,n := M,N$ ;
{ Invariant: $0<m \wedge 0<n \wedge M \,\mathrm{gcd}\, N = m \,\mathrm{gcd}\, n$ }
do $m<n \rightarrow n := n-m$
□ $n<m \rightarrow m := m-n$
od
{ $M \,\mathrm{gcd}\, N = m$ } .
```

a) Suggest a bound function for the program.

b) Construct verification conditions that demonstrate that the loop body maintains the invariant, and that the postcondition is satisfied on termination of the loop body.

c) Prove the validity of the verification conditions you have constructed using (9).

d) Show how to prove that multiplication distributes over gcd by identifying a suitable invariant of the algorithm. State clearly the definition of gcd that you assume, and what properties of subtraction are needed to verify the invariant.

Suppose the program is augmented with assignments to variables p and q, as shown below.

```
{ $0<M \wedge 0<N$ }
$m,n,p,q := M,N,M,N$ ;
```

```
{ Invariant:   $0<m \wedge 0<n \wedge M \,\mathrm{gcd}\, N = m \,\mathrm{gcd}\, n$ }
do $m<n \rightarrow n,q := n-m, p+q$
□ $n<m \rightarrow m,p := m-n, p+q$
od
{ $M \,\mathrm{gcd}\, N = m$ } .
```

e) Identify an invariant property relating all of m, n, p and q. The property involves a linear combination of some, but not all of, $m \times p$, $n \times p$, $m \times q$ and $n \times q$.

f) The least common multiple of M and N, denoted $m \,\mathrm{lcm}\, n$, has the property that

$$(m \,\mathrm{lcm}\, n) \times (m \,\mathrm{gcd}\, n) \;=\; m \times n \;\; .$$

Using this fact, and your solution to part (e), show how to calculate the least common multiple of M and N from the final values of p and q.

□

5 Quantifiers

These questions are mainly about the use of quantifiers.

Exercise 10 Suppose a is an array of numbers of length N. For integers i, h and k, define $asc.(i,h,k)$ by:

$$asc.(i,h,k) \;\equiv\; 0 \leq i \leq i{+}h \leq k \wedge \langle \forall j : i \leq j < i{+}h : a[i] \leq a[j] \rangle$$

(In words, $asc.(i,h,k)$ means that the segment of a of length h beginning at index i is an ascending subsegment of the k-long initial segment of a.) Define the function len by

$$len.k \;=\; \langle \Uparrow h,i : asc.(i,h,k) : h \rangle \;\; .$$

(In words, $len.k$ is the maximum length of an ascending segment of the k-long initial segment of a.)

In answering the following questions, state clearly the properties of quantifiers that you use at each step of your calculations.

a) Simplify $len.0$.

b) Show that

$$len.(k{+}1) \;=\; len.k \uparrow \langle \Uparrow h : asc.(k{+}1{-}h, h, k{+}1) : h \rangle$$

□

6 Miscellaneous

Exercise 11 It is required to *reverse* the elements of a large array. For example, if the array stores the values

$$3\ 6\ 2\ 0\ 1\ 4\ 0\ 9\ 4\ ,$$

(in that order) after the reversal it should store the values

$$4\ 9\ 0\ 4\ 1\ 0\ 2\ 6\ 3\ .$$

Design a program that performs an *in-situ* reversal. (That is, the program may use only a constant amount of additional storage space; it is not allowed to copy the array into another array.) The program should be designed to work for arrays of arbitrary length (including zero, of course). You should assume that the procedure $swap$ is such that $swap(i,j)$ swaps the array values indexed by i and j, provided that i and j are within the bounds of the array. When i equals j, you may assume that the procedure call is equivalent to skip.

Your program should be annotated with sufficient assertions that it is possible for an independent reader to determine its specification and formally verify that the specification is met by the program. In particular, the assertions should guarantee that no array-bound errors can occur.

□

Exercise 12 Consider the process of repeatedly removing from a bag of values any two distinct values until it is no longer possible to do so. A bag that results from this process is called a *reduction* of the initial bag.

For example, given the bag of values

$$0\ 0\ 2\ 1\ 0\ 5\ 2\ ,$$

removing 0 and 2, followed by 1 and 0, and then 5 and 2 results in a bag with one element 0. Alternatively, removing 0 and 2, followed by 0 and 1, and then 0 and 5 results in a bag with one element 2.

Note that these two examples demonstrate that a reduction of a bag is not uniquely defined.

a) What relation holds between the size of a reduction and the size of the original bag? What can be said about the number of distinct elements in a reduction?

b) Construct a program that calculates a reduction of a given bag. Assume that the bag of values is represented by an array B of length N. Your program should be linear in the size of the initial bag; it should inspect the elements of the array one-by-one, maintaining the invariant that R is a reduction of the inspected array elements. Choose a suitable representation of R. (Hint: Exploit your solution to (a).)

c) Suppose an element of the initial bag occurs a majority of times (i.e. more than $N{\div}2$ times, where N is the size of the bag). What can be said about any reduction of the bag? Justify your claim. Making use of your answer, extend the program so that it determines whether there is an element of the initial bag that occurs a majority of times.

□

Exercises 13 (below) and 12(above) are (disguised) variations of the same problem.

Exercise 13 Assume that V is an array of length N. The function count is defined for all k, $0\leq k\leq N$, and all x of the same type as the array elements, by

$$\mathsf{count}(x,k) \;=\; \langle \Sigma j : 0\leq j<k \wedge V[j]=x : 1\rangle \quad .$$

(In words, count counts the number of times the value x occurs in the initial segment of V of length k.)

The skeleton program below maintains three variables k, x and e, where k and x are as above, and e is a natural number.

```
{ $0\leq N$ }
    $k,x,e \;:=\; 0,any,0$ { any value will do for $x$ }
; { **Invariant:**
          $0\leq k\leq N \;\wedge\; k-e\leq e$
      $\wedge\;\; \mathsf{count}(x,k)\leq e$
      $\wedge\;\; \langle\forall y : y\neq x : \mathsf{count}(y,k) \leq k-e\rangle$
      **Bound function:**    $N-k$ }
    do $k<N \;\rightarrow\;$ body ; $k := k+1$
    od
{ $\langle\forall y : y\neq x : \neg(\mathsf{count}(y,N) > \lfloor N/2\rfloor)\rangle$ }
```

a) Complete the program by filling in the details of the loop body (identified by "body" in the program).
b) Check formally the initialisation of the loop (i.e. the assignment "$k,x,e \;:=\; 0,any,0$"). State clearly the rules of the quantifier calculus that you use.
c) Check formally that the postcondition is indeed satisfied when the program terminates, stating clearly any properties of the floor function that you use.

□

Mex numbers are used in combinatorial game theory to determine winning strategies. This question is interesting for students who have been exposed to the theory.

Exercise 14 The mex number of an array of natural numbers is the smallest natural number that is not an element of the array. For example, the array

$$5 \quad 3 \quad 8 \quad 1 \quad 2 \quad 7 \quad 0 \quad 6 \quad 4$$

has mex number 9, and the array

$$0 \quad 3 \quad 2 \quad 1 \quad 0 \quad 5 \quad 2$$

has mex number 4. This question is about constructing an algorithm to calculate the mex number of a given array.

a) Write a formal specification of the mex number $mex.k$ of the initial segment of length k of array A. Assume that array indices begin with 0, and the ith element of the array is denoted by $A[i]$.
b) Use your specification to derive the value of $mex.0$.

One way to compute the mex number of an array is to first sort the array into ascending order..
c) Assuming that the array is sorted, give an algorithm to compute its mex number. Make explicit the invariant of any loops in your algorithm.

Sorting the array is not a very efficient way of calculating the mex number of an array when the length of the array is very large. A more efficient way would be to sort the array only to the extent that is necessary. This is the goal of the final part of this question.

d) Consider the following (informally stated) sketch of an algorithm.

Introduce variables a, m, k and s. The variable m records the mex number of the initial segment of length k of the given array, A. The variable a records a "semi-sorted" permutation of A; it is semi-sorted in that the initial segment of length k of a is a permutation of the initial segment of length k of A, the first s elements of a are all less than m, and the next $k-s$ elements of a are all greater than m. For example, suppose A is the following:

$$0 \quad 3 \quad 1 \quad 1 \quad 0 \quad 5 \quad 2$$

Then, when $k=2$, m and s would both be 1, and the array a would be equal to A. When $k=4$, m would be 2, s would be 3, and the array a would be as follows:

$$0 \quad 1 \quad 1 \quad 3 \quad 0 \quad 5 \quad 2$$

Construct an algorithm to calculate the mex number of a given array using the above as the invariant. Your algorithm should make use of only a *swap* procedure to change the array a, once it has been initialised. (Assume that $swap(i,j)$ swaps the elements $A[i]$ and $A[j]$, and that $swap(i,i)$ is valid, but has no effect.) For this reason, it is not necessary to include the fact that a is a permutation of A in your invariant. (Hint: for progress, use a lexicographic ordering on the pair (m,k); that is, progress is made by either increasing m, or not changing m and increasing k.)

$\Box$

Exercise 15 An array a is said to be a *permutation* if its elements form a permutation of the first N natural numbers, where N is the length of a. (That is, each of the numbers 0, 1, ..., $N-1$ occurs exactly once in the array.) For example, the array

$$5\ \ 3\ \ 8\ \ 1\ \ 2\ \ 7\ \ 0\ \ 6\ \ 4$$

of length 9 is a permutation of the numbers 0, 1, ..., 8.

An *inversion* is a pair of indices i and j such that $0 \le i < j < N \wedge a[j] < a[i]$. An example of an inversion is the pair $(2,7)$ — the array elements $a[2]$ (which equals 8) and $a[7]$ (which equals 6) satisfy $a[7] < a[2]$. The array has, in fact, a total of 19 inversions. An array that is sorted in ascending order would have no inversions.

Given a permutation a, the corresponding *inversion count* is an array b of the same length as a such that the element $b[i]$ is the number of inversions with first component i. For the array above, the inversion count is the array

$$5\ \ 3\ \ 6\ \ 1\ \ 1\ \ 3\ \ 0\ \ 1\ \ 0\ .$$

It is not difficult to design an algorithm that given a permutation computes its inversion count. More difficult is to design an efficient algorithm that, given an inversion count, computes the corresponding permutation. This question is about the latter problem. To help you, part (a) suggests a non-obvious method of computing an inversion count. Part (b) is then about reversing the steps in the inversion-count algorithm in order to solve the problem of constructing a permutation from an inversion count.

In order to specify the two problems, we use the notation

$$\langle \mathsf{N}i : r : p \rangle$$

to denote the number of values i in the range r that satisfy the property p. For example,

$$\langle \mathsf{N}i : 0 \le i < M : a[0] < a[i] \rangle \ = \ a[0]$$

is a property of a permutation a of length M.

This example is simultaneously a hint for later! It is illustrated by the permutation and inversion count given above — the number 5 is the first element in both arrays. The property is true because, for any number k, there are k natural numbers less than k, and the inversion count of $a[0]$ is the number of natural numbers less than $a[0]$.

Also, that array a_0 is a permutation is specified formally by

$$\langle\forall j : 0\le j<M : \langle Ni{:}0\le i<M{:}a_0[i]=j\rangle = 1\rangle \quad .$$

This property of the array a_0 should be assumed throughout.

(a) Develop an algorithm to compute the inversion count of the permutation a_0. The algorithm should use a loop variable k that is initialised to 0 and is incremented by 1 at each iteration of the loop. The loop should update the given array so that its initial value is a permutation and its final value is the inversion count. That is, it should satisfy the specification

$$\{ \quad 0\le M \ \wedge \ \langle\forall j : 0\le j<M : a[j] = a_0[j]\rangle \quad \}$$
$$CountInversions$$
$$\{ \quad \langle\forall i : 0\le i<M : a[i] = \langle Nj{:}i\le j<M{:}a_0[j]<a_0[i]\rangle\rangle \quad \}$$

It should do so by maintaining the following invariant properties:

- The final segment of the array of length $M-k$ is a permutation. (Recall that this means that every natural number less than $M-k$ occurs exactly once in the segment.)

- The final segment of the array of length $M-k$ has the same inversion count as the given permutation a_0. That is,

  (16) $\quad \langle\forall i{:}k\le i<M{:}\ \langle Nj{:}i\le j<M{:}a[j]<a[i]\rangle = \langle Nj{:}i\le j<M{:}a_0[j]<a_0[i]\rangle\rangle \quad ,$

- The initial segment of the array of length k is the inversion count of the initial segment of the given permutation a_0. That is,

  (17) $\quad \langle\forall i{:}0\le i<k{:}a[i] = \langle Nj{:}i\le j<M{:}a_0[j]<a_0[i]\rangle\rangle \quad .$

For example, for the permutation above, when k has the value 5, the array should have been transformed to

$$5 \quad 3 \quad 6 \quad 1 \quad 1 \quad 3 \quad 0 \quad 2 \quad 1$$

The initial segment of length 5

$$5 \quad 3 \quad 6 \quad 1 \quad 1$$

is the inversion count for the corresponding initial segment of the input permutation. Moreover, the final segment,

$$3 \quad 0 \quad 2 \quad 1 \ ,$$

is a permutation (of the numbers 0, 1, 2 and 3). It has the same inversion count as

$$7 \quad 0 \quad 6 \quad 4 \ ,$$

which is the corresponding final segment of the input permutation.

In your solution, you may find it convenient to refer to the invariant properties by the labels given to them above.

□

Drawing a parabola is a longer exercise than drawing a hyperbola. Nevertheless, it is very straightforward if the principles are well understood.

Exercise 18 A *raster display* is a two-dimensional grid of *pixels*. The pixels are all squares of equal size. A black-and-white *drawing* is an assignment of booleans to each of the pixels.

The equation of a *parabola* is

$$x,y \ :: \ a{\times}y \ = \ b{\times}x^2 \ ,$$

where a and b are given constants.

Develop a program to construct a drawing of a parabola in the quadrant

$$m,n \ :: \ 0{\le}m \wedge 0{\le}n \ .$$

You should assume that a and b are positive integers.

Your program should have two phases. In the first phase, coordinate m is incremented, whilst coordinate n (the best integer approximation to $(b{\times}m^2)/a$) is computed. In the second phase, n is incremented whilst m (the best integer approximation to $\sqrt{(a{\times}n)/b}$) is computed. The first phase should be terminated when increasing m by 1 results in an increase in n by more than 1. There is no need to provide a termination condition for the second phase.

□

Solutions to Exercises

1 (“Everywhere” brackets denote universal quantification with bound variables M, r, q.)

a) The verification condition for termination is:

$$[\neg(10<r) \wedge M{+}1 = 9\times q+r \Rightarrow 0\leq r{-}1<9 \wedge M = 9\times q+(r{-}1)] \quad .$$

The bug is that $\neg(10<r)$ does not imply $r{-}1<9$. The termination test should be $10\leq r$.

b) At this point, we note that $0\leq r{-}1$ is not guaranteed. This (or the equivalent $1\leq r$) must be added to the invariant. From now on, the invariant is

$$1\leq r \wedge M{+}1 = 9\times q+r \quad .$$

c)

r	q
1368	0
144	136
18	150
9	151
8	151

Table 1: $M = 1367$

r	q
25680	0
2568	2568
264	2824
30	2850
3	2853
2	2853

Table 2: $M = 25679$

d) From the test cases, it is clear that the intention is that r decreases at each iteration. So, the bound function is r.

(An alternative bound is obtained by observing that q always increases. However, in that case, it is necessary to show, from the invariant, that there is an upper bound on

q . A suitable upper bound is M . This gives as bound function $M-q$. Both solutions are acceptable. The details for the first are given below.)

e)
Initialisation:

$$[0\le M \;\Rightarrow\; M{+}1 = 9\times 0 + M{+}1 \wedge 1\le M{+}1] \quad .$$

Termination:

$$[\neg(10\le r) \wedge 1\le r \wedge M{+}1 = 9\times q + r \;\Rightarrow\; 0\le r{-}1 < 9 \wedge M = 9\times q + (r{-}1)] \quad .$$

Loop body. There are two conditions, corresponding to making progress and maintaining the invariant. (Solutions that combine the two into one statement are also acceptable.)
Making progress (K is a ghost variable):

$$[10\le r \wedge 1\le r \wedge M{+}1 = 9\times q + r \wedge r{=}K \;\Rightarrow\; (r \bmod 10) + (r \div 10) < K] \quad .$$

Maintaining the invariant:

$$\begin{array}{ll} [& 10\le r \;\wedge\; 1\le r \;\wedge\; M{+}1 = 9\times q + r \\ \Rightarrow & \quad 1 \le (r \bmod 10) + (r\div 10) \\ . & \quad \wedge \quad M{+}1 \;=\; 9\times(q + (r\div 10)) + (r \bmod 10) + (r \div 10) \end{array}$$

Each of the verification conditions is everywhere true, so the program is correct.
□

2 (“Everywhere” brackets denote universal quantification over the bound variables m, M, x, X and z.)
Initialisation:

$$[0\le M \;\Rightarrow\; 0\le M \wedge M\times X + 0 = M\times X] \quad .$$

Termination (conditional correctness):

$$[\neg(m{\ne}0) \wedge 0\le m \wedge m\times x + y = M\times X \;\Rightarrow\; y = M\times X] \quad .$$

Loop body. There are four verification conditions, corresponding to the two branches of the conditional statement, making progress and maintaining the invariant.
Making progress:

$$\begin{array}{ll} [& m{\ne}0 \wedge m{=}K \wedge 0\le m \wedge m\times x + y = M\times X \wedge \mathsf{even}.m \\ \Rightarrow & m{\div}2 < K \;] \quad . \end{array}$$

$$\begin{array}{ll} [& m\neq 0 \wedge m=K \wedge 0\leq m \wedge m\times x+y = M\times X \wedge \mathsf{odd}.m \\ \Rightarrow & (m-1)\div 2<K \;] \quad . \end{array}$$

Maintaining the invariant:

$$\begin{array}{ll} [& m\neq 0 \wedge 0\leq m \wedge m\times x+y = M\times X \wedge \mathsf{even}.m \\ \Rightarrow & (m\div 2)\times 2\times x+y = M\times X \;] \quad . \end{array}$$

$$\begin{array}{ll} [& m\neq 0 \wedge 0\leq m \wedge m\times x+y = M\times X \wedge \mathsf{odd}.m \\ \Rightarrow & ((m-1)\div 2)\times 2\times x+y-x = M\times X \;] \quad . \end{array}$$

Note that the latter verification condition is invalid.

Formally, two additional verification conditions are needed: that the measure of progress is a natural number is implied by the invariant:

$$[0\leq m \wedge m\times x+y = M\times X \;\Rightarrow\; 0\leq m] \quad ,$$

and the two cases in the conditional are exhaustive:

$$[\mathsf{even}.m \vee \mathsf{odd}.m] \quad .$$

b) As pointed out above, one verification condition is invalid. This is corrected by replacing "$y-x$" in the assignment by "$y+x$".

$\Box$

3 We introduce variables m and n, with the invariant properties:

$$\begin{array}{ll} & \langle\forall i : 0\leq i<m : \langle\forall j : 0\leq j : f.i\neq g.j\rangle\rangle \\ \wedge & \langle\forall j : 0\leq j<n : \langle\forall i : 0\leq i : f.i\neq g.j\rangle\rangle \end{array}$$

We observe that, because g is ascending, if $f.m<g.n$, m can be incremented by 1. Conversely, because f is ascending, if $g.n<f.m$, n can be incremented by 1. We thus obtain the following algorithm.

```
{ true }
$m,n := 0,0$ ;
{ **Invariant:**   as above }
do $f.m<g.n \rightarrow m := m+1$
$\Box$ $g.n<f.m \rightarrow n := n+1$
od .
```

The measure of progress in $(M-m)+(N-n)$ where M and N are the known values satisfying $f.M=g.N$.

□

4 a)

$$
\begin{array}{ll}
 & \langle\Downarrow i,j : m\leq i<M \wedge n\leq j<N : |f[i]-g[j]|\rangle \\
= & \{ \quad \text{range splitting (on } j=n\text{)} \quad \} \\
 & \langle\Downarrow i : m\leq i<M : |f[i]-g[n]|\rangle \downarrow \langle\Downarrow i,j : m\leq i<M \wedge n{+}1\leq j<N : |f[i]-g[j]|\rangle \\
= & \{ \quad \langle\Downarrow i : m\leq i<M : |f[i]-g[n]|\rangle \\
 & \quad = \quad \{ \quad \text{range splitting (on } i=m\text{)} \quad \} \\
 & \quad |f[m]-g[n]| \downarrow \langle\Downarrow i : m{+}1\leq i<M : |f[i]-g[n]|\rangle \\
 & \quad = \quad \{ \quad f[m]-g[n]\geq 0\,, \\
 & \qquad\qquad \text{for all } i\ ,\ m{+}1\leq i<M \Rightarrow f[m]\leq f[i] \\
 & \qquad\qquad \text{(and hence } f[m]-g[n]\leq f[i]-g[n]\text{)} \quad \} \\
 & \quad f[m]-g[n] \quad \} \\
 & (f[m]-g[n]) \downarrow \langle\Downarrow i,j : m\leq i<M \wedge n{+}1\leq j<N : |f[i]-g[j]|\rangle \quad .
\end{array}
$$

b)

$$
\begin{array}{ll}
 & \langle\Downarrow i,j : m\leq i<M \wedge n\leq j<N : |f[i]-g[j]|\rangle \\
= & (g[n]-f[m]) \downarrow \langle\Downarrow i,j : m{+}1\leq i<M \wedge n\leq j<N : |f[i]-g[j]|\rangle \quad .
\end{array}
$$

c) (A complete solution would go through the process of introducing variables m and n, capturing their function in the invariant, and then analyzing the invariant to determine when m can be increased and when n can be increased.)

The initialisation of d is to a value that is at least as large as the final answer. The value $|f[0]-g[0]|$ is as good as any. (Infinity is less desirable for implementation purposes, but must be chosen if M or N is allowed to be zero.).

$$
\begin{array}{l}
\{ \quad \langle\forall i : 0\leq i<M : \langle\forall j : i\leq j<M : f[i]\leq f[j]\rangle\rangle \\
\quad \wedge \quad \langle\forall i : 0\leq i<N : \langle\forall j : i\leq j<N : g[i]\leq g[j]\rangle\rangle \quad \} \\
m,n,d := 0,0,|f[0]-g[0]| \ ; \\
\{ \ \textbf{Invariant:} \\
\qquad d \downarrow \langle\Downarrow i,j : m\leq i<M \wedge n\leq j<N : |f[i]-g[j]|\rangle \\
\quad = \langle\Downarrow i,j : 0\leq i<M \wedge 0\leq j<N : |f[i]-g[j]|\rangle \quad \}
\end{array}
$$

```
do $m \neq M \wedge n \neq N$ $\rightarrow$ if    $f.m < g.n \rightarrow m,d := m+1, d\downarrow(g[n]-f[m])$
                                     □ $g.n < f.m \rightarrow n,d := n+1, d\downarrow(f[m]-g[n])$
                                     □ $f.m = g.n \rightarrow m,n,d := M,N,0$
   fi
od .
```

The measure of progress (bound function) is $(M-m)+(N-n)$.

□

5 As pointed out, the function mapping k and l to $M\times k - N\times l$ is strictly increasing in k and strictly decreasing in l. A saddleback search can be used to determine how often this function has the value 0. We introduce variables k, l and count, with the invariant property:

$$0\leq k\leq N \wedge 0\leq l\leq M \quad\wedge\quad \mathsf{count}+S.(k,l) \;=\; S.(0,0)$$

where

$$S.(k,l) \;=\; \langle \Sigma i,j : k\leq i\leq N \;\wedge\; l\leq j\leq M \;\wedge\; M\times i = N\times j : 1\rangle \quad .$$

The initialisation is straightforward: set all of k, l and count to 0. For the loop body, we observe that, because $M\times k - N\times l$ is strictly increasing in k, if $M\times k < N\times l$, k can be incremented by 1. Conversely, because $M\times k - N\times l$ is strictly decreasing in l, if $N\times l < M\times k$, l can be incremented by 1. When $M\times k$ and $N\times l$ are equal, the count is incremented as well as both k and l. The loop is terminated when either k equals N or l equals M. On termination of the loop,

$$(k = N \vee l = M) \quad\wedge\quad \mathsf{count}+S.(k,l) \;=\; S.(0,0) \quad .$$

Since M is the unique solution of the equation $j{::}\; M\times N = N\times j$, and N is the unique solution of the equation $i{::}\; M\times i = N\times M$, we conclude that $S.(N,l) = S.(k,M) = 1$ (when $l\leq M$ and $k\leq N$). That is, on termination,

$$\mathsf{count}+1 \;=\; S.(0,0) \quad .$$

We thus obtain the following algorithm.

```
{ $0\leq M \wedge 0\leq N$ }
$k, l, \mathsf{count} := 0, 0, 0$ ;
{ **Invariant:**   as above }
do $k\neq N \wedge l\neq M$ $\rightarrow$   if  $M\times k < N\times l \rightarrow k := k+1$
```

```
                          □  $N\times l < M\times k \rightarrow l := l+1$
                          □  $M\times k = N\times l \rightarrow k, l, count := k+1, l+1, count+1$
                          fi
od ;
count := count+1
{  $count = S.(0,0) = \langle \Sigma i,j : 0\leq i\leq N \wedge 0\leq j\leq M \wedge M\times i = N\times j : 1\rangle$  }  .
```

The measure of progress is $(M-l)+(N-k)$. This is bounded below by 0 (because $0\leq k\leq N \wedge 0\leq l\leq M$) and is decreased at each iteration (either by 1 or by 2).

In order to avoid the repeated multiplication of M by k and N by l, we maintain a variable d with the invariant property $d = M\times k - N\times l$. The modifications are straightforward:

```
{  $0\leq M \wedge 0\leq N$  }
$k, l, count, d := 0, 0, 0, 0$ ;
{ **Invariant:**   as above  }
do $k\neq N \wedge l\neq M \rightarrow$  if  $d<0 \rightarrow k, d := k+1, d+M$
                       □  $d>0 \rightarrow l, d := l+1, d-N$
                       □  $d=0 \rightarrow k, l, count, d := k+1, l+1, count+1, d+M-N$
                       fi
od ;
count := count+1
{  $count = S.(0,0) = \langle \Sigma i,j : 0\leq i\leq N \wedge 0\leq j\leq M \wedge M\times i = N\times j : 1\rangle$  }  .
```

□

6 It suffices to use two variables, one to record $f^k.0$ for successive values of k, and the other to record $f^{2\times k}.0$. The program terminates when the two values are equal:

```
{  f is ultimately periodic  }
$f1, f2, k := f.0, f^2.0, 1$ ;
{ **Invariant:**   $f1 = f^k.0 \wedge f2 = f^{2\times k}.0 \wedge 1\leq k$  }
do $f1\neq f2 \rightarrow f1, f2, k := f.f1, f.(f.f2), k+1$
od
```

$\{\quad f^k.0 = f^{2\times k}.0 \quad\}$;

$p := k$

$\{\quad p$ is a multiple of the period of $f \quad\}$

□

7 a) The precise statement of the property is: for all m and n,

$$\langle\forall j :: d.(m+j+1) \ = \ d.(m+n+j+1)\rangle \ \equiv \ \langle\forall j :: r.(m+j) \ = \ r.(m+n+j)\rangle \quad .$$

To prove this property, we first calculate as follows:

$d.(i+1) \ = \ d.(i+j+1)$

$= \qquad \{ \qquad$ definition of $d \quad \}$

$(10\times r.i)\div N \ = \ (10\times r.(i+j))\div N$

$= \qquad \{ \qquad$ Leibniz, $0<N \quad \}$

$(10\times r.i)\div N\times N \ = \ (10\times r.(i+j))\div N\times N$

$= \qquad \{ \qquad p \ = \ p\div N\times N + p \bmod N$ (applied twice) $\quad \}$

$10\times r.i - (10\times r.i) \bmod N \ = \ 10\times r.(i+j) - (10\times r.(i+j)) \bmod N$

$= \qquad \{ \qquad$ arithmetic $\quad \}$

$10\times(r.i - r.(i+j)) \ = \ (10\times r.i) \bmod N - (10\times r.(i+j)) \bmod N$

$= \qquad \{ \qquad$ definition $\quad \}$

$10\times(r.i - r.(i+j)) \ = \ r.(i+1) - r.(i+j+1) \quad .$

So,

$\langle\forall j :: d.(m+j+1) \ = \ d.(m+n+j+1)\rangle$

$= \qquad \{ \qquad$ above with $i,j := m+j\, ,n \quad \}$

$\langle\forall j :: 10\times(r.(m+j) - r.(m+n+j)) \ = \ r.(m+j+1) - r.(m+n+j+1)\rangle$

$= \qquad \{ \qquad$ An easy inductive proof shows that,

for any sequence of numbers c,

$\langle\forall j :: 10\times c.j \ = \ c.(j+1)\rangle$

$\equiv \ \langle\forall j :: 10^j\times c.0 \ = \ c.j\rangle \quad .$

Instantiate $c.j$ to $r.(m+j) - r.(m+n+j)$. $\quad \}$

$\langle\forall j :: 10^j\times(r.m - r.(m+n)) \ = \ r.(m+j+1) - r.(m+n+j+1)\rangle$

$=$ { $r.(m{+}j{+}1) - r.(m{+}n{+}j{+}1)$ is bounded
above and below (by $-N$ and N)
$10^j \times (r.m - r.(m{+}n))$ is bounded
exactly when $r.m - r.(m{+}n) = 0$ }

$\langle \forall j :: 0 \; = \; r.(m{+}j) - r.(m{+}n{+}j) \rangle$

$=$ { arithmetic }

$\langle \forall j :: r.(m{+}j) \; = \; r.(m{+}n{+}j) \rangle$.

b) All $N{+}1$ numbers in the sequence $r.0$, $r.1$, $\ldots$, $r.N$ are at most 0 and less than N. So, at least two of them must be equal. That is,

$$\langle \exists j,k : 0 \le j < k \le N : r.j = r.k \rangle \quad .$$

From the definition of r, it follows that

$$\langle \exists j,k \; : \; 0 \le j < k \le N \; : \; \langle \forall i : 0 \le i : r.(i{+}j) = r.(i{+}k) \rangle \rangle \quad .$$

(Formally, this is proved by induction on i.) Instantiating i to $N{-}j$, we get

$$\langle \exists j,k \; : \; 0 \le j < k \le N \; : \; r.N = r.(N{-}j{+}k) \rangle \quad .$$

Thus (with $m := k{-}j$)

$$\langle \exists m : 0 < m : r.N = r.(N{+}m) \rangle \quad .$$

c) We find the Nth remainder, and then determine the smallest positive k for which $r.N = r.(N{+}k)$.

```
i,r := 0,1 ; do $i \neq N \rightarrow i,r \; := \; i{+}1\,,(10{\times}r) \bmod N$ od
{ $r = r.N$ } ;
$k\,,r' \; := \; 1\,, r \bmod N$ ;
do $r \neq r' \rightarrow k,r \; := \; k{+}1\,,(10{\times}r') \bmod N$ od
```

The first loop has invariant $r = r.i$; the second loop has invariant

$$r = r.N \wedge r' = r.(N{+}k) \wedge \langle \forall j : 0 < j < k : r.N \neq r.(N{+}j) \rangle \quad .$$

(These are easily checked.) The second loop terminates as a result of part (b). It terminates with $r = r'$. So, combining with the invariant, this means that k is set to the period of r. From part (a), this is the period of the decimal representation of $\frac{1}{N}$.
□

8 a) $m{+}n$.
b) Following the mechanical process for constructing verification conditions, we get:

$$\begin{array}{ll} & [m{<}n \;\wedge\; 0{<}m \;\wedge\; 0{<}n \;\wedge\; M \,\mathsf{gcd}\, N = m \,\mathsf{gcd}\, n \\ \Rightarrow & 0{<}m \wedge 0{<}n{-}m \wedge M \,\mathsf{gcd}\, N = m \,\mathsf{gcd}\,(n{-}m)\,] \end{array}$$

$$\begin{array}{ll} & [n{<}m \;\wedge\; 0{<}m \;\wedge\; 0{<}n \;\wedge\; M \,\mathsf{gcd}\, N = m \,\mathsf{gcd}\, n \\ \Rightarrow & 0{<}m{-}n \wedge 0{<}n \wedge M \,\mathsf{gcd}\, N = (m{-}n)\,\mathsf{gcd}\, n\,] \end{array}$$

and

$$\begin{array}{ll} & [0{<}m \;\wedge\; 0{<}n \;\wedge\; M \,\mathsf{gcd}\, N = m \,\mathsf{gcd}\, n \;\wedge\; \neg(m{<}n \vee n{<}m) \\ \Rightarrow & M \,\mathsf{gcd}\, N = m\,] \end{array}$$

c) The verification conditions are valid if we can prove that

$$[m \,\mathsf{gcd}\, n = m \,\mathsf{gcd}\,(n{-}m)]$$

and

$$[m \,\mathsf{gcd}\, m = m] \quad .$$

(The latter arises because $\neg(m{<}n \vee n{<}m) \;\equiv\; m{=}n$.)
The former is established by rule of indirect equality. For all p ,

$$\begin{array}{ll} & p \backslash (m \,\mathsf{gcd}\,(n{-}m)) \\ = & \{ \qquad (9) \quad \} \\ & p\backslash m \;\wedge\; p\backslash(n{-}m) \\ = & \{ \qquad \text{definition of division} \quad \} \\ & \langle \exists k :: m{=}k{\times}p \rangle \;\wedge\; \langle \exists k :: n{-}m{=}k{\times}p \rangle \\ = & \{ \qquad \text{distributivity, renaming} \quad \} \\ & \langle \exists k :: m{=}k{\times}p \;\wedge\; \langle \exists j :: n{-}m{=}j{\times}p \rangle\rangle \\ = & \{ \qquad \text{Leibniz} \quad \} \\ & \langle \exists k :: m{=}k{\times}p \;\wedge\; \langle \exists j :: n - k{\times}p = j{\times}p \rangle\rangle \\ = & \{ \qquad \text{arithmetic, range translation: } j{:=}j{-}k \quad \} \\ & \langle \exists k :: m{=}k{\times}p \;\wedge\; \langle \exists j :: n{=}j{\times}p \rangle\rangle \\ = & \{ \qquad \text{distributivity} \quad \} \end{array}$$

$$\langle \exists k :: m = k \times p \rangle \;\wedge\; \langle \exists k :: n = k \times p \rangle$$

$= \qquad \{ \qquad$ definition of divides $\quad \}$

$$p \backslash m \;\wedge\; p \backslash n$$

$= \qquad \{ \qquad (9) \quad \}$

$$p \backslash (m \,\mathrm{gcd}\, n) \quad .$$

Substituing $m \,\mathrm{gcd}\, (n-m)$ for p, and $m \,\mathrm{gcd}\, n$ for p, and using the anti-symmetry of the divides relation, we get

$$[m \,\mathrm{gcd}\, n = m \,\mathrm{gcd}\, (n-m)] \quad .$$

For the second property, we have, for all p:

$$p \backslash (m \,\mathrm{gcd}\, m)$$

$= \qquad \{ \qquad (9) \quad \}$

$$p \backslash m \;\wedge\; p \backslash m$$

$= \qquad \{ \qquad$ idempotency of conjunction $\quad \}$

$$p \backslash m \quad .$$

Substituing $m \,\mathrm{gcd}\, m$ for p, and m for p, and using the anti-symmetry of the divides relation, we get

$$[m \,\mathrm{gcd}\, m = m] \quad .$$

d) The invariant is

$$\langle \forall c, k \;::\; k \backslash (c \times M) \wedge k \backslash (c \times N) \;\equiv\; k \backslash (c \times m) \wedge k \backslash (c \times n) \rangle$$

The definition of gcd is

$$\langle \forall k \;::\; k \backslash M \wedge k \backslash N \;\equiv\; k \backslash (M \,\mathrm{gcd}\, N) \rangle$$

On termination of the algorithm, m and n are both equal to $M \,\mathrm{gcd}\, N$. It follows that

$$\langle \forall c, k \;::\; k \backslash (c \times M) \wedge k \backslash (c \times N) \;\equiv\; k \backslash (c \times (M \,\mathrm{gcd}\, N)) \rangle$$

Since $c \times M \,\mathrm{gcd}\, c \times N$ is the unique number x satisfying

$$\langle \forall k \;::\; k \backslash (c \times M) \wedge k \backslash (c \times N) \;\equiv\; k \backslash x \rangle$$

it follows that $c \times (M \,\mathrm{gcd}\, N)$ and $c \times M \,\mathrm{gcd}\, c \times N$ are equal.

In order to verify the invariant, we need to show that

$$k \backslash (c \times m) \wedge k \backslash (c \times n) \;\equiv\; k \backslash (c \times (m-n)) \wedge k \backslash (c \times n)$$

for all k and c, and all strictly positive m and n such that $n < m$.

e)

$$m{\times}q + n{\times}p = 2{\times}M{\times}N \ .$$

(This can be calculated from the hint by assuming that a , b , c and d are such that

$$a{\times}m{\times}p + b{\times}m{\times}q + c{\times}n{\times}p + d{\times}n{\times}q$$

is a constant. Then, using the assignment axiom, we require that

$$\begin{array}{ll} & a{\times}m{\times}p + b{\times}m{\times}q + c{\times}n{\times}p + d{\times}n{\times}q \\ = & a{\times}(m{-}n){\times}(p{+}q) + b{\times}(m{-}n){\times}q + c{\times}n{\times}(p{+}q) + d{\times}n{\times}q \end{array}$$

This is true if a and d are both 0 and b and c are both 1 . Symmetrically, we require that

$$\begin{array}{ll} & a{\times}m{\times}p + b{\times}m{\times}q + c{\times}n{\times}p + d{\times}n{\times}q \\ = & a{\times}m{\times}p + b{\times}m{\times}(p{+}q) + c{\times}(n{-}m){\times}p + d{\times}n{\times}(p{+}q) \end{array}$$

This is also true if a and d are both 0 and b and c are both 1 .)
f) On termination, $m = n = M \,\mathsf{gcd}\, N$. So,

$$\begin{array}{ll} & \mathsf{true} \\ = & \{ \quad \text{(d) and } m = n = M \,\mathsf{gcd}\, N \quad \} \\ & (M \,\mathsf{gcd}\, N) \times (p{+}q) \ = \ 2{\times}M{\times}N \\ = & \{ \quad (m \,\mathsf{lcm}\, n) \times (m \,\mathsf{gcd}\, n) \ = \ m{\times}n \quad \} \\ & (M \,\mathsf{gcd}\, N) \times (p{+}q) \ = \ 2 \times (M \,\mathsf{lcm}\, N) \times (M \,\mathsf{gcd}\, N) \\ \Rightarrow & \{ \quad \text{arithmetic} \quad \} \\ & (p{+}q){\div}2 \ = \ M \,\mathsf{lcm}\, N \ . \end{array}$$

That is, on termination, the average of p and q is the least common multiple of M and N .
□

10 a)

$$\begin{array}{ll} & len.0 \\ = & \{ \quad \text{definition of } len \quad \} \\ & \langle \Uparrow h,i : asc.(i,h,0) : h \rangle \\ = & \{ \quad \text{definition of } asc \quad \} \\ & \langle \Uparrow h,i \ : \ 0 \le i \le i{+}h \le 0 \wedge \langle \forall j : i \le j < i{+}h : a[i] \le a[j] \rangle \ : \ h \rangle \end{array}$$

$=\qquad \{\quad 0\le i\le i+h\le 0 \equiv 0=i=h\,,$
$\qquad\qquad$ Leibniz $\}$

$\quad\langle\Uparrow h,i : 0=i=h \wedge \langle\forall j:0\le j<0:a[i]\le a[j]\rangle : h\rangle$

$=\qquad \{\quad$ empty range $\}$

$\quad\langle\Uparrow h,i:0=i=h:h\rangle$

$=\qquad \{\quad$ one-point rule $\}$

$\quad 0\ .$

b)

$\quad len.(k+1)$

$=\qquad \{\quad$ definition of len $\}$

$\quad\langle\Uparrow h,i : asc.(i,h,k+1) : h\rangle$

$=\qquad \{\quad$ definition of asc $\}$

$\quad\langle\Uparrow h,i : 0\le i\le i+h\le k+1 \wedge \langle\forall j : i\le j<i+h : a[i]\le a[j]\rangle : h\rangle$

$=\qquad \{\quad$ range splitting on $i+h=k+1$,
$\qquad\qquad$ definition of len $\}$

$\quad len.k\uparrow\langle\Uparrow h,i : 0\le i\le i+h=k+1 \wedge \langle\forall j : i\le j<i+h : a[i]\le a[j]\rangle : h\rangle$

$=\qquad \{\quad$ one-point rule $\}$

$\quad len.k\uparrow\langle\Uparrow h : 0\le k+1-h\le k+1 \wedge \langle\forall j : k+1-h\le j<k+1 : a[i]\le a[j]\rangle : h\rangle$

$=\qquad \{\quad$ definition of asc $\}$

$\quad len.k\uparrow\langle\Uparrow h : asc.(k+1-h,h,k+1) : h\rangle\ .$

□

11 The array elements are reversed from the outside inwards. The two indices j and k delimit that part of the array still to be reversed. N is the length of the array.

```
{ $0\le N \wedge \langle\forall i : 0\le i<N : a[i]=a_0[i]\rangle$ }
$j,k := 0\,,N-1$
{ **Invariant:**
        $0\le j<N \wedge 0\le k<N \wedge j=N-1-k$
   $\wedge \langle\forall i : 0\le i<j \vee k\le i<N : a[i]=a_0[N-1-i]\rangle$
```

```
        $\wedge \; \langle \forall i : j \le i \le k : a[i] = a_0[i] \rangle$
        **Bound function:** $k-j$ }
;   do $j < k$ →        $swap(j,k)$ ; $j,k := j+1, k-1$
    od
    { $\langle \forall i : 0 \le i < N : a[i] = a_0[N-1-i] \rangle$ }
```

Note the use of two variables j and k. This preserves the symmetry between the top and bottom halves of the array and helps to avoid error in the calculation of the array indices.

(This question seems very straightforward. However, a common mistake is to assert that $j \le k$, and to terminate the loop when j and k are equal.)
□

12 a) The size of the reduction is even equivales the size of the initial bag is even. There is at most one distinct element in the reduction. That is, all elements of the reduction have a common value.
b) We represent R by a pair (c, n). The number c is the number of elements in the reduction, and n is the (common) value of all the elements in the reduction. If c is zero, the value of n is arbitrary.

```
{ $0 < M \wedge 0 < N$ }
$c,n,k := 0, AnyValue, 0$ ;
{ **Invariant:**   the pair $c$, $n$ represents a reduction of $B[0..k)$ }
do $k < N$ →     if   $c \ne 0 \wedge n = B[k]$ → $c := c+1$
                 □   $c \ne 0 \wedge n \ne B[k]$ → $c := c-1$
                 □   $c = 0$ → $c,n := 1, B[k]$
                 fi ;
                 $k := k+1$
od
{ the pair $c$, $n$ represents a reduction of $B[0..N)$ }
```

c) If an element occurs more than $N \div 2$ times, any reduction of the bag contains that element at least once. This is because each element is removed from the initial bag at most $N \div 2$ times.

After execution of the above algorithm, if c is non-zero, it suffices to compute the number of occurrences of the value n in the array. If c is zero, no value occurs a majority of times.

An algorithm for counting the number of occurrences of value n in the array is, of course, very straightforward to write.
□

13 a) Incrementing k may falsify one or more of the three conjuncts in the invariant other than $0 \le k \le N$. If $V[k] = x$, the value of $\mathsf{count}(x,k)$ increases by 1. The three conjuncts are truthified by the assignment $e := e+1$. If $V[k] \ne x$, the value of $\mathsf{count}(x,k)$ remains unchanged. If $k-e < e$, the conjunct $k-e \le e$ is not falsified and nor is the universal quantification. So, in this case nothing needs to be done. In the final case, when $V[k] \ne x \wedge k-e = e$, the conjunct $k-e \le e$ is falsified. We note, however, that $\mathsf{count}(x,k) \le k-e$ (because $\mathsf{count}(x,k) \le e$ and $k-e$ equals e). This is the opportunity to change the value of x. The only reasonable candidate is $V[k]$. Summarising, the loop body becomes:

```
if  $V[k] = x \ \rightarrow\ e := e+1$
□   $V[k] \ne x \wedge k-e < e \rightarrow \mathsf{skip}$
□   $V[k] \ne x \wedge k-e = e \rightarrow x, e \ := \ V[k], e+1$
fi
```

To save writing inv is an abbreviation for the invariant as stated in the algorithm. (We write it out in full when substitutions are made for the variables.)

The verification conditions are as follows. ("Everywhere" brackets denote universal quantification over the bound variables count, N, V, k, e and z.)
Initialisation:

$$[0 \le N \ \Rightarrow\ 0 \le 0 \le N \wedge 0-0 \le 0 \wedge \langle \forall y : y \ne any : \mathsf{count}(y,0) \le 0-0 \rangle] \quad .$$

Termination (conditional correctness):

$$[\neg(k < N) \wedge \mathsf{inv} \Rightarrow \langle \forall y : y \ne x : \neg(\mathsf{count}(y,k) > \lfloor k/2 \rfloor)\rangle] \quad .$$

Loop body. (Not asked in the question, but supplied here anyway.) There are four verification conditions, corresponding to the three branches of the conditional statement and making progress.
Maintaining the invariant:

$$\begin{array}{ll} [& k < N \wedge \mathsf{inv} \wedge V[k] = x \\ \Rightarrow & \quad 0 \le k+1 \le N \wedge (k+1)-(e+1) \le e+1 \\ & \wedge \quad \langle \forall y : y \ne x : \mathsf{count}(y,k) \le (k+1)-(e+1) \rangle \ ] \quad . \end{array}$$

$$\begin{array}{ll} [& k<N \wedge \mathsf{inv} \wedge V[k]\neq x \wedge k-e<e \\ \Rightarrow & \quad 0\leq k+1\leq N \wedge (k+1)-e\leq e \\ & \wedge \quad \langle\forall y : y\neq x : \mathsf{count}(y,k) \leq (k+1)-e\rangle \ ] \quad . \end{array}$$

$$\begin{array}{ll} [& k<N \wedge \mathsf{inv} \wedge V[k]\neq x \wedge k-e=e \\ \Rightarrow & \quad 0\leq k+1\leq N \wedge (k+1)-(e+1)\leq e+1 \\ & \wedge \quad \langle\forall y : y\neq V[k] : \mathsf{count}(y,k) \leq (k+1)-(e+1)\rangle \ ] \quad . \end{array}$$

Making progress: at each iteration the variable k is increased. Formally,

$$[N-k=K \Rightarrow N-(k+1)<K] \quad .$$

b) After simplification, and unfolding the definition of count, the initialisation is verified by checking that:

$$\begin{array}{ll} [& 0\leq N \\ \Rightarrow & 0\leq N \wedge \langle\forall y : y\neq any : \langle\Sigma j : 0\leq j<0 \wedge V[j]=y : 1\rangle \leq 0\rangle \ ] \quad . \end{array}$$

The simplification of the universal quantification proceeds as follows:

$$\begin{array}{ll} & \langle\forall y : y\neq any : \langle\Sigma j : 0\leq j<0 \wedge V[j]=y : 1\rangle \leq 0\rangle \\ = & \quad \{ \quad 0\leq j<0\equiv false \text{ empty range} \quad \} \\ & \langle\forall y : y\neq any : 0\leq 0\rangle \\ = & \quad \{ \quad 0\leq 0\equiv true,\ true \text{ is the unit of conjunction} \quad \} \\ & true \end{array}$$

Substituting, we have to check that

$$[0\leq N \Rightarrow 0\leq N\wedge true]$$

which is clearly $true$.

c)

$$\begin{array}{ll} & \neg(k<N) \wedge \mathsf{inv} \\ \Rightarrow & \quad \{ \quad \text{definition of the invariant} \quad \} \\ & k=N \wedge k-e\leq e \wedge \langle\forall y : y\neq x : \mathsf{count}(y,k) \leq k-e\rangle \\ = & \quad \{ \quad \text{Leibniz, arithmetic, definition of ceiling function} \quad \} \end{array}$$

$k = N \;\wedge\; \lceil N/2 \rceil \leq e \;\wedge\; \langle \forall y : y \neq x : \mathsf{count}(y,N) \leq N - e \rangle$

$\Rightarrow$ { monotonicity }

$\langle \forall y : y \neq x : \mathsf{count}(y,N) \leq N - \lceil N/2 \rceil \rangle$

$=$ { $N = \lceil N/2 \rceil + \lfloor N/2 \rfloor$ }

$\langle \forall y : y \neq x : \mathsf{count}(y,N) \leq \lfloor N/2 \rfloor \rangle$

$=$ { converse of $\leq$ is $>$ }

$\langle \forall y : y \neq x : \neg(\mathsf{count}(y,N) > \lfloor N/2 \rfloor) \rangle$

$\Box$

14 a)

$mex.k \;=\; \langle \Downarrow n : \langle \forall i : 0 \leq i < k : n \neq A[i] \rangle : n \rangle$

b)

$mex.0$

$=$ { definition }

$\langle \Downarrow n : \langle \forall i : 0 \leq i < 0 : n \neq A[i] \rangle : n \rangle$

$=$ { empty range }

$\langle \Downarrow n : \mathsf{false} : n \rangle$

$=$ { range of n is the natural numbers }

0 .

c)

```
{ $A$ is sorted }
$m,k,done := 0,0,\mathsf{false}$ ;
{ **Invariant:** $0 \leq k \leq N \wedge m = mex.k \wedge (done \Rightarrow m = mex.N)$
  **Bound function:** $k$ (is bounded above) }
do $\neg done \wedge k < N$ →   if   $m = A[k] \rightarrow m := m+1$
                                □    $m < A[k] \rightarrow done := \mathsf{true}$
                                □    $m > A[k] \rightarrow \mathsf{skip}$
                                fi ;
                                $k := k+1$
```

```
    od
    { $m = mex.N$ }
```

d)

```
    { $0 \le N$ }
    $m,s,k := 0,0,0$ ;
    { Invariant:
            $0 \le m \le s \le k \le N \wedge m = mex.k$
        $\wedge$  $\langle \forall i : 0 \le i < s : a[i] < m \rangle$
        $\wedge$  $\langle \forall i : s \le i < k : a[i] > m \rangle$
      Bound function:   $(m, k)$ ordered lexicographically }
    do $k < N$ →   if   $m = A[k]$ → $swap(s,k)$ ; $m,s,k := m+1, s+1, s+1$
                   □ $m < A[k]$ → $k := k+1$
                   □ $m > A[k]$ → $swap(s,k)$ ; $s,k := s+1, k+1$
                   fi
    od
    { $m = mex.N$ }
```

□

15 a)

For convenience, we use the notation $a[m..n)$ to refer to the segment of the array a indexed from m up to, but not including, n. We also use $count.i$ and $count0.i$ to denote the inversion counts of $a[i]$ and of $a_0[i]$, respectively. That is,

(19) $count.i = \langle Nj : i \le j < M : a[i] < a[j] \rangle$,

and

(20) $count0.i = \langle Nj : i \le j < M : a_0[i] < a_0[j] \rangle$,

The invariant properties are then:

(21) $permutation.a[k..M)$,

(22) $\langle \forall i : k \le i < M : count.i = count0.i \rangle$,

(23) $\langle \forall i : 0 \le i < k : a[i] = count0.i \rangle$.

All three are trivially established by the assignment $k{:=}0$. ((21) and (22) are given.)

Now, supposing all three invariant properties hold and $k \neq M$, the crucial observation is that

$$a[k] \; = \; count.k \quad .$$

(This is the hint given in the question. It is a consquence of (21).) So, incrementing k by 1 maintains (17). It also maintains (16) but it may invalidate (21). To re-establish (21) and simultaneously maintain (16) after incrementing k, divide the array indices into three disjoint sets: those indices i for which $0 \leq i \leq k$, those for which $k < i < M \wedge a[i] < a[k]$ and those for which $k < i < M \wedge a[i] > a[k]$. (Note that $k < i < M$ and $a[i] = a[k]$ is not possible.) By decrementing $a[i]$ by 1 whenever i is in the third set (i.e. $k < i < M \wedge a[i] > a[k]$), whilst leaving all other array elements unchanged, we

- maintain invariant the inversion count for elements in the first set,
- maintain invariant the inversion count for elements in the third set (since all are uniformly decremented),
- maintain invariant the inversion count between elements in the second set (since all are unchanged),
- maintain invariant the inversion count between elements in the second set and elements in the third set (since the difference between elements in the two sets is at least 2 before the decrement and hence at least 1 after the decrement),
- maintain invariant the uniqueness of all array values in the final segment of the array (again because the difference between elements in the second and third sets is at least 2 before the decrement and hence at least 1 after the decrement, and because elements in the third set are uniformly decremented).

In other words, we reestablish (21) and simultaneously maintain (16) as required. Thus the program is as follows:

```
{    $0 \leq M \;\wedge\; \langle \forall j : 0 \leq j < M : a[j] \,=\, a_0[j] \rangle$
   $\wedge \;\; permutation.a[0..M)$   }
$k{:=}0$
{  **Invariant:**
   $0 \leq k \leq M \wedge (1) \;\wedge\; (2) \;\wedge\; (3)$
   **Bound function:** $M{-}k$   }
```

```
;   do $k \neq M$ →   $k := k+1$
                      ; $innerloop$
    od
    { $\langle \forall j : 0 \leq j < M : a[j] = count0.j \rangle$ }
```

where $innerloop$ is the program segment

```
    $j := k+1$
;   do $j < M$ →   if   $a[j] < a[k]$ → skip
                   □   $a[j] > a[k]$ → $a[j] := a[j]-1$
                   fi
                 ; $j := j+1$
    od
```

□

18 In the first phase, we approximate the straight line by the function g from integers to integers given by

$$g.m \;=\; \left\lceil \frac{bm^2}{a} - \frac{1}{2} \right\rceil \quad .$$

Thus, the program must compute $g.m$ successively for m equal to 0, 1, 2, $\cdots$. Termination of this phase occurs when

$$g.(m+1) > g.m+1 \quad .$$

More precisely, the first phase has the following form.

```
    { true }
    $m,n := 0,0$ ;
    { **Invariant:**   $n = g.m$ }
    do $g.(m+1) \leq g.m+1$ →   $plot.(m,n)$ ;
                                $m,n := m+1 , g.(m+1)$
    od .
```

In order to compute g incrementally, we must determine when $g.(m+1) \leq g.m$. We have:

$$g.(m+1) \leq g.m$$

$=$ { $n = g.m$ is an invariant of the algorithm,
so we may replace $g.m$ on the right side by n,
definition of g on the left side }

$$\left\lceil \frac{b\times(m+1)^2}{a} - \frac{1}{2} \right\rceil \leq n$$

$=$ { definition of ceiling }

$$\frac{b\times(m+1)^2}{a} - \frac{1}{2} \leq n$$

$=$ { arithmetic and floor function }

$$0 \leq a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor$$

$=$ { assume $h = a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor$ }

$$0 \leq h \ .$$

In order to compute the termination condition incrementally, we must determine when $g.(m+1) \leq g.m+1$. We have:

$$g.(m+1) \leq g.m+1$$

$=$ { as above }

$$\frac{b(m+1)^2}{a} - \frac{1}{2} \leq n+1$$

$=$ { arithmetic }

$$0 \leq a\times n - b\times(m+1)^2 + \lfloor\frac{3\times a}{2}\rfloor$$

$=$ { $h = a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor$ }

$$0 \leq h+a \ .$$

Introducing the variable h, the algorithm now takes the form:

```
{ true }
$m,n,h \ := \ 0,0,\lfloor\frac{a}{2}\rfloor - b$ ;
{ **Invariant:**
        $n = g.m$
    $\wedge$  $h \ = \ a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor$
    $\wedge$  $(-a \leq h < 0 \ \equiv \ g.(m+1) - g.m = 1)$ }
do $0 \leq h+a \rightarrow$    plot.(m,n) ;
```

```
                    if $0 \leq h$ →        { $g.(m+1) = g.m$ }
                                           skip
                    □ $\neg(0 \leq h)$ →   { $g.(m+1) = g.m+1$ }
                                           $n,h := n+1 , p$
                    fi ;
                    $m,h := m+1 , q$
od .
```

The unknowns in this algorithm are p and q. The requirements on p and q are that they should maintain invariant the property

$$h = a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor \ .$$

Now, for arbitrary k,

```
{ $h = k - b\times(m+1)^2$ }
$m,h := m+1 , h - b\times(2\times m+3)$
{ $h = k - b\times(m+1)^2$ } .
```

Similarly, for arbitrary k,

```
{ $h = a\times n + k$ }
$n,h := n+1 , h+a$
{ $h = a\times n + k$ } .
```

Substituting for p and q completes the derivation of the first phase:

```
{ true }
$m,n,h := 0,0,\lfloor\frac{a}{2}\rfloor - b$ ;
{ **Invariant:**
          $n = g.m$
     ∧    $h = a\times n - b\times(m+1)^2 + \lfloor\frac{a}{2}\rfloor$
     ∧    $(-(2\times a) \leq h < 0 \equiv g.(m+1) - g.m = 1)$ }
do $0 \leq h+a$ →   plot.(m,n) ;
                    if $0 \leq h$ →    { $g.(m+1) = g.m$ }
                                       skip
                    □ ¬(0 ≤ h) → { g.(m+1) = g.m+1 }
                                 n,h := n+1 , h+a
                    fi ;
                    m,h := m+1 , h − b×(2×m+3)
    od .
```

For the second phase, n is incremented by one at each iteration and we must determine whether m is, or is not, incremented as well.

Formally, we consider the function f from integers to integers given by

$$f.n \;=\; \left\lceil \sqrt{\frac{a\times n}{b}} \;-\; \frac{1}{2} \right\rceil \quad .$$

The second phase then has the following form.

```
    { n = g.m }
    m := f.n ;
    { Invariant:   m = f.n }
    do true →   plot.(m,n) ;
                m,n := f.(n+1) , n+1
    od .
```

We determine whether $f.(n{+}1) \leq f.n$ as follows.

$$f.(n{+}1) \leq f.n$$

$=$ { $m = f.n$ is an invariant of the algorithm, so we may replace the right side by m, definition of g on the left side }

$$\left\lceil \sqrt{\frac{a\times(n{+}1)}{b}} \;-\; \frac{1}{2} \right\rceil \;\leq\; m$$

$=$ { definition of ceiling }

$$\sqrt{\frac{a\times(n{+}1)}{b}} \;-\; \frac{1}{2} \;\leq\; m$$

$=$ { arithmetic (mainly squaring) }

$$a\times(n{+}1) \;\leq\; b\times(m^2+m+\frac{1}{4})$$

$=$ { arithmetic and floor function }

$$0 \;\leq\; b\times(m^2+m) \;-\; a\times(n{+}1) \;-\; \lfloor\frac{b}{4}\rfloor \quad .$$

As before, we introduce a variable h to compute incrementally the right side of this expression and, hence, to determine when m should be increased.

```
{   $n = g.m$   }
$m$  :=  $f.n$ ;
$h$  :=  $b \times (m^2 + m) - a \times (n+1) - \lfloor \frac{b}{4} \rfloor$ ;
{  **Invariant:**
            $m = f.n$
    $\wedge$    $h = b \times (m^2 + m) - a \times (n+1) - \lfloor \frac{b}{4} \rfloor$
    $\wedge$    $(f.(n+1) \leq f.n \equiv 0 \leq h)$   }
do true $\rightarrow$   plot.(m,n) ;
            if  $0 \leq h \rightarrow$      {  $f.(n+1) = f.n$   }
                                 skip
            $\Box \; \neg(0 \leq h) \rightarrow$    {  $f.(n+1) = f.n + 1$   }
                                  $m,h$  :=  $m+1$ , $h + 2 \times b \times (m+1)$
            fi  ;
            $n,h$  :=  $n+1$ , $h - a$
od  .
```

An additional optimization (in both phases) is to eliminate the incremental evaluation of $b \times m$. This is done by introducing an additional variable to record this value; this variable is then incremented by b every time that m is incremented.

(Note that the assignments to h in the two phases are almost inverses of each other. This is to be expected, and increases confidence in the correctness of the calculations.)
$\Box$