CSC236 Winter 2020
Assignment #2: recurrences & correctness
Solutions

1. In lecture, we used the following recurrence to represent the steps taken by an implementation of mergesort on a list of size n:

$$T_0(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + 2T_0(n/2) & \text{if } n > 1 \end{cases}$$

(This recurrence assumes n is a power of 2, hence the absence of floor and ceiling. You may maintain this assumption throughout this question.)

In reality, some implementations of divide-and-conquer algorithms stop the recursion before the input size becomes trivial. For example, a programmer may find that their mergesort implementation ends up running a bit faster if they stop recursing when the list size is less than 10, sorting these small lists using selection sort.

Consider the following recurrence, which models this scenario:

$$T(n) = \begin{cases} c & \text{if } n \leq k \\ n + 2T(n/2) & \text{if } n > k \end{cases}$$

$k, c \in \mathbb{N}^+$ are fixed constants, where k represents the largest problem size which is solved non-recursively, and c represents the cost of solving these small problems.

(a) Use unwinding[1] to find a closed form for $T(n)$ when $n \geq k$. (You do not need to prove that your closed form is correct, but it should be clear how you arrived at it.)

Solution:
I will assume that n is a power of 2 and $n \geq k$.

[1] **Logistical note:** If you wish to use tree diagrams for the unwinding portions of this question (parts (a) and (c)), you are welcome to include scanned hand-drawn images, or diagrams generated using other software. See this chapter of the LaTeXWikibook for information on including images in LaTeXdocuments. You may also describe the solution tree without explicitly drawing it (a table may be helpful).

Let $a = \lfloor \log k \rfloor$, and $k' = 2^a$. In other words, k' is the largest power of 2 which is $\leq k$.

$$\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= n + 2(n/2 + 2(n/4 + 2T(n/8)) \\
&= n + n + n + 8T(n/8) \\
&= n + n + n + \ldots (n/k')T(\frac{n}{n/k'})
\end{aligned}$$

We repeatedly divide the input size n by 2 until the recursion bottoms out when we reach a number $\leq k$. By construction, that number will be k'. Thus the final term in this sum (i.e. the leaf layer of the solution tree) should involve $T(k')$. Based on the pattern that calls of the form $T(n/q)$ are multiplied by q, and since $T(k')$ can be rewritten as $T(\frac{n}{n/k'})$, I hypothesize that the final term will be $(n/k')T(k')$. The number of 'n' terms in this sum will correspond to the number of times I must divide n by 2 before I reach k'. By definition, this is $\log(n/k')$. This yields the summation:

$$\begin{aligned}
T(n) &= (\sum_{i=0}^{\log(n/k')-1} n) + (n/k')T(k') \\
&= n\log(n/k') + (n/k')T(k') \\
&= n\log(n/k') + (n/k')c \qquad \text{\# By definition, } T(k') = c \\
&= n(\log n - \log k') + (n/k')c \qquad \text{\# Log identity}
\end{aligned}$$

Remark: While not necessary to include as part of your solution, as usual, it's a salutary practice to verify our closed form on some small values of n to ensure we don't have any errors (particularly off-by-one errors). As a sanity check, consider the case where $k = 9$ and $n = 32$. The recursive definition of $T(n)$ gives:

$$\begin{aligned}
T(32) &= 32 + 2T(16) \\
&= 32 + 2(16 + 2T(8)) \\
&= 32 + 32 + 4c \\
&= 64 + 4c
\end{aligned}$$

For our closed form, we get $a = \lfloor \log 9 \rfloor = 3$, $k' = 2^a = 8$. Thus we have

$$\begin{aligned} T(32) &= 32(\log 32 - \log 8) + (32/8)c \\ &= 32(5-3) + 4c \\ &= 64 + 4c \end{aligned}$$

So our recursive definition and closed form are in agreement in this case.

(b) What is the big-Θ complexity of $T(n)$? Does it depend on k? Briefly justify your answer (no proof required). You may not assume $n \geq k$ for this part. Do not use the master theorem.

Solution:
According to the definition of big-Θ, it suffices to show that the asymptotic behaviour of $T(n)$ has the desired form beyond some starting point n_0 of my choosing. By choosing $n_0 \geq k$, it's clear that the closed form I found in part (a) will have the same big-Θ as $T(n)$. Starting from the closed form from (a) and doing some rewriting, we get

$$\begin{aligned} T(n) &= n(\log n - \log k') + (n/k')c \\ &= n \log n - n \log k' + \frac{c}{k'} n \\ &= n \log n + (\frac{c}{k'} - \log k')n \end{aligned}$$

Since k is a fixed constant, $k' = 2^{\lfloor \log k \rfloor}$ is also constant. Since c is also a fixed constant, $(\frac{c}{k'} - \log k')$ is a constant, call it q'. So

$$T(n) = n \log n + qn$$

The growth of $n \log n$ dominates n (i.e. $n \log n \in \Omega(n) \wedge n \log n \notin O(n)$), so regardless of whether the constant q is positive, negative, or zero, qn is a lower-order term, which can be dropped. Thus $T(n) \in \Theta(n \log n)$.

(c) Rather than assigning a fixed cost to the $n \leq k$ case, it may be more realistic to use a function of n, since there are a range of input sizes which are handled non-recursively. Since selection sort is a $\Theta(n^2)$ algorithm, we'll define our new recurrence $T'(n)$ to be

$$T'(n) = \begin{cases} n^2 & \text{if } n \leq k \\ n + 2T'(n/2) & \text{if } n > k \end{cases}$$

Find a closed form for $T'(n)$ for $n \geq k$, and show how you got there. Rather than unwinding from scratch, you may find it simpler to build on your work from part (a).

Solution:
My unwinding in part (a) assumed that n was a power of 2 no less than k. For such n, the expansion of $T(n)$ will never involve terms of the form $T(q)$ where $q < k'$ (the largest power of $2 \leq$ k). Informally, starting from some power of 2 greater than k' and repeatedly dividing by 2, we must eventually reach k' - it's impossible to 'skip over' it.

This means that I can modify my closed form for $T(n)$ to make a closed form for $T'(n)$ by replacing all instances of c with k'^2, giving

$$\begin{aligned} T'(n) &= n \log n + (\frac{k'^2}{k'} - \log k')n \\ &= n \log n + (k' - \log k')n \end{aligned}$$

Remark: Many answers to this question fell into the pitfall of treating the amount of work at the leaf layer as n^2, where n was the *original* size of the input. This is not correct. For example, if $k = 4$, then $T'(8) = 8 + 2T'(4) = 8 + 2 \times 16$, rather than $8 + 2 \times 64$.

(d) Is $T'(n) \in \Theta(T(n))$? Why or why not? Briefly justify your answer. As in part (b), you may not assume $n \geq k$. Do not use the master theorem.

Solution:
Yes. $T'(n)$ is also of the form

$$T'(n) = n \log n + qn$$

Where q is a fixed constant (in this case, $k - \log k$). So by the same reasoning as before, $T'(n) \in \Theta(n \log n) = \Theta(T(n))$. (Also, the same reasoning holds for why I'm justified in extrapolating from a closed form which required n to be no smaller than k.)

2. Our boss has tasked us with writing a program to find the *unique* maximum of a non-empty list of positive integers. If there is no unique maximum, our program should signal this by returning a negative number. For example, on input `[5, 2, 1, 2]`, our algorithm should return 5. Given `[2, 1, 2]`, we may return -1, -2, -236, or any other negative number.

   Below is our first attempt to solve this problem.[2]

[2] You can download the code for this question from http://www.cs.toronto.edu/~colin/236/W20/assignments/umax.py

```python
def umax(A):
    if len(A) == 1:
        return A[0]
    head = A[0]
    tail = A[1:]
    tmax = umax(tail)
    if head == tmax:
        return -1
    elif head > tmax:
        return head
    else:
        return tmax
```

(a) Based on the informal specification above, write precise pre- and post-conditions for umax. Your postcondition should use symbolic notation rather than restating the English description above ("find the unique maximum..."). The following postcondition was used in lecture for the function max, which found the (not necessarily unique) maximum of a list. It may be a useful starting point:

`max(A) = x` where $(\exists j \in \mathbb{N}, A[j] = x) \wedge (\forall i \in \mathbb{N}, i < \text{len}(A) \implies A[i] \leq x)$

You may find it helpful to formally define 'helper' functions or predicates, as is done in question 3.

Solution:
$Pre(A)$: A is a non-empty list of positive natural numbers.
For convenience, we will define the predicate $U(A,x)$, which informally corresponds to the statement "x is the unique maximum of list A":
$U(A,x) : \exists j \in \mathbb{N}, A[j] = x \wedge \forall i \in \mathbb{N}, (i < \text{len}(A) \wedge i \neq j) \implies A[i] < A[j]$
And also the predicate $HU(A)$, which can be understood as "A has a unique maximum":
$HU(A) : \exists z \in \mathbb{N}, U(A,z)$
With these helpers defined, we will now define our postcondition (as a predicate over lists):
$Post(A)$: `umax(A)` terminates and returns a number x, such that $(HU(A) \implies U(A,x)) \wedge (\neg HU(A) \implies x < 0)$
Remark: The following formulation is equivalent and shorter (though perhaps less intuitive): $U(A,x) \vee x < 0$

(b) The given Python code above has a bug. Demonstrate the bug by finding a value of A which meets the precondition, where umax misbehaves. For the value of A that you find, you should state the expected behaviour (according to your postcondition) and how it differs from the function's actual behaviour on that input.

Solution:
Consider $A = [1, 2, 2]$. According to the postcondition, umax should return -1 for this input, since it has no unique maximum. In reality, umax returns 1, because the recursive call $umax([2, 2])$ returns -1, then the check on line 7 fails, and we reach line 10, returning the head of the original list, which is 1.

(c) Consider our second draft of the function umax below:

```
1  def umax(A):
2      if len(A) == 1:
3          return A[0]
4      head = A[0]
5      tail = A[1:]
6      tmax = umax(tail)
7      if head == tmax:
8          return -1 * head
9      elif head > abs(tmax):
10         return head
11     else:
12         return tmax
```

Prove that this function is correct with respect to the specifications you devised in part (a).

Solution:
Define
$M(A, x) : \exists j \in \mathbb{N}, A[j] = x \wedge \forall i \in \mathbb{N}, i < \text{len}(A) \implies A[i] \leq j$
$C(A)$: `umax(A)` terminates and returns a number x, such that $(HU(A) \implies U(A, x)) \wedge (\neg HU(A) \implies M(A, -x))$
$P(n)$: For any list A of length n, $Pre(A) \implies C(A)$
I will use induction to prove $\forall n \in \mathbb{N}^+, P(n)$. Note that this implies the correctness of umax with respect to the earlier pre and postconditions, because $C(A)$ is stronger than $Post(A)$. To see why, note that A may only contain positive integers, so $M(A, -x) \implies x < 0$.
Base Case:Let $A = [x]$ be a single-element list, with $x \in \mathbb{N}^+$. Then x is the unique max of A, and by lines 2-3, umax(A) returns x. So $C(A)$ is satisfied. Thus $P(1)$.
Inductive Step:Let $n \in \mathbb{N}^+$ and assume $P(n)$. Let A be a list of length $n + 1$ such that $Pre(A)$ is satisfied.
By the IH, $C(tail)$ holds, since tail is of length n and satisfies the precondition.

Case 1: $HU(tail)$. Then tmax is the unique maximum of tail. If head = tmax, then A has no unique maximum, and head is the non-unique maximum. So we should return -head, which we do by lines 7-8. If head is greater than the maximum of the tail, then head is the unique max, and we should return it, which we do by line 9-10. Otherwise, tmax is the unique max of A, and we should return it, which we do by lines 11-12.

Case 2: $\neg HU(tail)$. Then by $C(tail)$, $M(tail, -tmax)$, i.e. tmax is the negation of the non-unique max of the tail. Since head is positive, we cannot reach line 8. If head is greater than the absolute value of tmax, then it is the unique max of A, and we should return it, which we do by lines 9-10. Otherwise, we have $\neg HU(A) \wedge M(A, -tmax)$, so we should return tmax, which we do by lines 11-12.

In every case $C(A)$ holds. Thus $P(n+1)$.

3. The function `maj` takes as input a list of natural numbers and, if the list has a majority element (one that appears more often than all other elements combined), it returns that element.[3] For example `maj([1, 3, 3, 2, 3])` returns 3.[4]

```python
def R(A):
    B = []
    i = 0
    while i < len(A):
        a = A[i]
        b = A[(i+1) % len(A)]
        if a == b:
            B.append(a)
        i += 1
    return B

def maj(A):
    prev = A
    curr = R(A)
    while len(curr) != len(prev):
        prev = curr
        curr = R(curr)
    return curr[0]
```

Prove that `maj` is correct.

[3] Note that the function's behaviour is undefined if the input list does not have a majority element. i.e. the presence of a majority element is a precondition of `maj`.

[4] You can download the code for this question from http://www.cs.toronto.edu/~colin/236/W20/assignments/maj.py

Solution:

Prelude: This sample proof is more detailed and pedantic than what we will expect when marking this question. Some of the results (particularly in section 3) are 'obvious' enough that they could be safely stated without proof, or given just a brief justification of a sentence or two.

1 Overview

Our proof of the correctness of maj follows from three major results:

- We prove that R's output is never larger than its input. This entails maj's termination.
- We prove that R stops shrinking its input only if that input is uniform. This means that when maj's loop exits, the list curr is uniform.
- We prove that R conserves 'advantage' (see definition below). A consequence of this is that if m is a majority element of A, it will also be majority element of $\mathrm{R}(A)$, $\mathrm{R}(\mathrm{R}(A))$, etc. Therefore, when the loop exits, curr contains *only* the majority element.

The first two results are proven in section 3. The last is proven in section 4. In all three cases, our proofs are really in terms of the functions COUNT and PAIRS, and we use the following theorem which was given as part of the question to tie the results back to the function R:

Theorem 1.1 (R "correctness"). *Given any $A \in \mathbb{N}^*$, $\mathrm{R}(A)$ terminates and returns a list of natural numbers B such that $\forall x \in \mathbb{N}, \mathrm{COUNT}(B, x) = \mathrm{PAIRS}(A, x)$*

Finally, in section 5, we present a correctness proof for maj using these results.

2 Definitions

We will use $\mathbb{N}^*$ to denote the set of lists of natural numbers. Each of the functions below takes as its first argument a list $A \in \mathbb{N}^*$.

$\mathrm{COUNT}(A, x)$: $|\{i \in \mathbb{N} \mid A[i] = x\}|$

i.e. the number of occurrences of x in list A

$$\mathrm{SUCC}(A, i) : \begin{cases} 0 & \text{if } i = \mathrm{len}(A) - 1 \\ i + 1 & \text{if } i < \mathrm{len}(A) - 1 \end{cases}$$

i.e. the 'next' index in list A after index i, treating A as a circular list (with the last index 'wrapping around' back to index 0). Not defined if $i \geq \text{len}(A)$.

$\text{PAIRS}(A, x)$: $|\{i \in \mathbb{N} \mid A[i] = A[\text{SUCC}(A, i)] = x\}|$

i.e. the number of $[x, x]$ pairs in list A, again treating A as circular

$\text{ADVANTAGE}(A, x)$: $\text{COUNT}(A, x) - \mathit{len}(A)/2$

$\text{MAJORITY}(A, x)$: $\text{ADVANTAGE}(A, x) > 0$

i.e. x is the majority element of A

3 Pair counts

In this section, we present a number of results relating $\text{PAIRS}(A, x)$ to $\text{COUNT}(A, x)$. In particular, we will show that $\text{PAIRS}(A, x) \leq \text{COUNT}(A, x)$, and that the inequality is strict except in the specific case where A is uniform. This will have immediate applications to function R, via 1.1.

Lemma 3.1. $\text{COUNT}(A, x) - \text{PAIRS}(A, x) = |\{i \in \mathbb{N} \mid A[i] = x \wedge A[\text{SUCC}(A, i)] \neq x\}|$ *for all A, x.*

Proof. Let A be an arbitrary list, and let $x \in \mathbb{N}$ Define

$$I_c = \{i \in \mathbb{N} \mid A[i] = x\}$$

$$I_p = \{i \in \mathbb{N} \mid A[i] = A[\text{SUCC}(A, i)] = x\}$$

Note that, by definition, $\text{COUNT}(A, x) = |I_c|$, and $\text{PAIRS}(A, x) = |I_p|$. So

$$\text{COUNT}(A, x) - \text{PAIRS}(A, x) = |I_c| - |I_p|$$

It is clear by inspection that $I_p \subseteq I_c$, since the conditions for inclusion in the former set are stronger than the latter. Thus

$$\begin{aligned}\text{COUNT}(A, x) - \text{PAIRS}(A, x) &= |I_c - I_p| \\ &= |\{i \in \mathbb{N} \mid A[i] = x \wedge A[\text{SUCC}(A, i)] \neq x\}|\end{aligned}$$

□

Corollary 3.1.1. $\text{PAIRS}(A, x) \leq \text{COUNT}(A, x)$ *for all A, x.*

Theorem 3.2 (R outputs are no larger than inputs). $\forall A \in \mathbb{N}^*, \text{len}(\text{R}(A)) \leq \text{len}(A)$

Proof. This follows from 1.1 (which says that $\mathrm{R}(A)$ contains an x for every pair of x's in A), combined with 3.1.1 (which says that the number of pairs of x's in a list is no more than the number of x's in that list), plus the observation that the length of a list is equal to the sum of the counts of all elements of the list. □

Lemma 3.3. *For $k > 0$, $\mathrm{COUNT}(A, x) = \mathrm{PAIRS}(A, x) = k$ iff A is uniform and consists of $k = \mathrm{len}(A)$ repetitions of element x.*

Proof. Consider the case when $\mathrm{COUNT}(A, x) = \mathrm{PAIRS}(A, x) = k$, $k > 0$, for some A, x. By 3.1, this occurs iff

$$S = \{i \in \mathbb{N} \mid A[i] = x \wedge A[\mathrm{SUCC}(A, i)] \neq x\} = \emptyset$$

In other words, $\forall i \in \mathbb{N}, A[i] = x \implies A[\mathrm{SUCC}(A, i)] = x$.

Since we know there is at least one index i such that $A[i] = x$, this sets up an induction-like domino effect that implies $\forall i \in \mathbb{N}, A[i] = x$.

The opposite direction also works. If A is non-empty and uniformly consists of element x, it clearly follows that the set S above is empty, since there is no index j such that $A[j] \neq x$. □

Theorem 3.4 (R fixed points). *For any $A \in \mathbb{N}^*$, if $\mathrm{len}(\mathrm{R}(A)) = \mathrm{len}(A)$, then A is uniform (i.e. all elements are equal).*

Proof. By 1.1, each element x appears in $\mathrm{R}(A)$ $\mathrm{PAIRS}(A, x)$ times. If $\mathrm{len}(\mathrm{R}(A)) = \mathrm{len}(A)$, it follows that $\mathrm{COUNT}(A, x) = \mathrm{PAIRS}(A, x)$ for every element. By 3.3, this can only happen when A is uniform. □

4 R conserves advantage

In this section, we prove a result which will form our key loop invariant for maj. (Note: for the sake of correctness, we only *need* to show that R preserves majority elements, i.e. $\mathrm{MAJORITY}(A, x) \iff \mathrm{MAJORITY}(\mathrm{R}(A), x)$. But I found it more straightforward to prove this stronger result.)

Theorem 4.1 (R conserves advantage). *For any $A \in \mathbb{N}^*, m \in \mathbb{N}$,*

$$\mathrm{ADVANTAGE}(A, m) \leq \mathrm{ADVANTAGE}(\mathrm{R}(A), m)$$

Proof. By definition of ADVANTAGE, the claimed inequality is equivalent to

$$\mathrm{COUNT}(A, x) - \mathrm{len}(A)/2 \leq \mathrm{COUNT}(\mathrm{R}(A), x) - \mathrm{len}(\mathrm{R}(A))/2 \quad (1)$$

Which, by 1.1, is equivalent to

$$\text{COUNT}(A, x) - \text{len}(A)/2 \leq \text{PAIRS}(A, x) - \text{len}(\text{R}(A))/2 \tag{2}$$

I will prove this by induction on the number of non-x elements. Define

$P(k)$: for any A, x, where $\text{len}(A) - \text{COUNT}(A, x) = k$, inequality (2) holds.

It suffices to show that $\forall k \in \mathbb{N}, P(k)$. I will do so by simple induction.

Base Case:Suppose $k = 0$, and consider some corresponding A and x. Then A uniformly consists of element x. If A is empty, then by inspection, both sides of the inequality are 0. If A is non-empty, then by 3.3, $\text{PAIRS}(A, x) = \text{COUNT}(A, x) = \text{len}(A) = \text{len}(\text{R}(A))$. In either case, LHS = RHS. This verifies $P(0)$.

Inductive Step:Assume $P(k)$ for arbitrary $k \in \mathbb{N}$.

Let A, x be a list and element such that $\text{len}(A) - \text{COUNT}(A, x) = k + 1$. Consider the list A' formed by removing an arbitrary non-x element (call it y) from A. By the IH, we know:

$$\text{COUNT}(A', x) - \text{len}(A')/2 \leq \text{PAIRS}(A', x) - \text{len}(\text{R}(A'))/2 \tag{3}$$

Now consider how each side of this inequality is affected when we reinsert the deleted element at its original position. (i.e. as we replace A' with A in the LHS and RHS of the inequality).

The count of x is the same in both cases, and $\text{len}(A) = \text{len}(A') + 1$, so the LHS has a net decrease of .5.

The RHS may change in one of three ways, depending on the neighbours of the reinserted element y:

- If y is inserted between two x's, then $\text{PAIRS}(A, x)$ and $\text{len}(\text{R}(A))$ are each 1 less than their A' counterpart, resulting in a net decrease of .5 for the RHS. (Note: in all other cases, $\text{PAIRS}(A', x) = \text{PAIRS}(A, x)$.)
- If y is inserted to the left or right of another y, a new pair is created, and $\text{len}(\text{R}(A)) = \text{len}(\text{R}(A')) + 1$, resulting in a net decrease of .5 for the RHS.
- If y is inserted between two identical elements (which are neither x nor y), a pair is destroyed, and $\text{len}(\text{R}(A)) = \text{len}(\text{R}(A')) - 1$. The RHS is increased by .5.
- Otherwise, if y is inserted between a pair of elements (a, b) such that $a \neq b$, and neither is equal to y, then the total number of pairs stays the same. The RHS is unchanged.

In every case, the RHS is decreased by no more than .5, so the inequality is preserved, i.e.

$$\text{COUNT}(A, x) - \text{len}(A)/2 \leq \text{PAIRS}(A, x) - \text{len}(\text{R}(A))/2 \tag{4}$$

Therefore, by simple induction, $\forall k \in \mathbb{N}, P(k)$. □

5 maj correctness

Lemma 5.1 (maj loop invariant). *For an input array $A \in \mathbb{N}^*$ having a majority element m, the following invariant holds at the end of each iteration j of maj's while loop:*

(a) $\text{prev}_j = \text{curr}_{j-1}$ *if* $j > 0$

(b) $\text{curr}_j = R(\text{prev}_j)$

(c) $\text{ADVANTAGE}(A, m) \leq \text{ADVANTAGE}(\text{curr}_j, m)$

(d) $\text{curr}_j \in \mathbb{N}^*$

Proof. Before the first iteration (i.e. when $j = 0$), we have:

(a) Vacuously true, since $j = 0$

(b) $\text{curr}_0 = \text{R}(A) = \text{R}(\text{prev}_0)$ (by lines 13-14)

(c) by 4.1, $\text{ADVANTAGE}(A, m) \leq \text{ADVANTAGE}(\text{R}(A), m) = \text{ADVANTAGE}(\text{curr}_0, m)$

(d) $\text{curr}_0 = \text{R}(A) \in \mathbb{N}^*$, by 1.1

Assume the invariant holds at the end of some jth iteration, and that a $j+1$th iteration occurs. Then

(a) By line 16, $\text{prev}_{j+1} = \text{curr}_j$.

(b) By line 17, $\text{curr}_{j+1} = \text{R}(\text{curr}_j)$. By the above line, $\text{R}(\text{curr}_j) = \text{R}(\text{prev}_{j+1})$.

(c) By the IH, $\text{ADVANTAGE}(A, m) \leq \text{ADVANTAGE}(\text{curr}_j, m)$. $\text{ADVANTAGE}(\text{curr}_{j+1}, m) = \text{ADVANTAGE}(\text{R}(\text{curr}_j), m)$. By 4.1 R conserves advantage, so $\text{ADVANTAGE}(\text{R}(\text{curr}_j), m) \geq \text{ADVANTAGE}(\text{curr}_j, m) \geq \text{ADVANTAGE}(A, m)$.

(d) $\text{curr}_{j+1} = \text{R}(\text{curr}_j) \in \mathbb{N}^*$, by 1.1

□

Lemma 5.2 (maj termination). *For any input A satisfying the precondition, maj terminates.*

Proof. We only call subroutine R on lists of natural numbers ($A \in \mathbb{N}^*$ by the precondition, and $\text{curr}_j \in \mathbb{N}^*$ for all j by 5.1), therefore these calls to R terminate (see Lemma 1.4 in the A2 appendix). Thus it suffices to show that the while loop beginning on line 15 does not loop infinitely.

Consider the quantity $\text{len}(\text{prev}_j)$. By the definition of the len function, this must be a natural number. Furthermore, by 5.1, $\text{prev}_{j+1} = \text{curr}_j = \text{R}(\text{prev}_j)$.

By 3.2 $\text{len}(\text{R}(A)) \leq \text{len}(A)$ for any list A. But in order for a $j+1$th iteration to occur, we must have $\text{len}(\text{curr}_j) \neq \text{len}(\text{prev}_j)$, by the loop condition on line 15. Thus $\text{len}(\text{prev}_{j+1} = \text{curr}_j = \text{R}(\text{prev}_j)$ is strictly less than $\text{len}(\text{prev}_j)$.

Therefore, $\langle \text{len}(\text{prev}_0), \text{len}(\text{prev}_1), \text{len}(\text{prev}_2), \ldots \rangle$ is a decreasing sequence of natural numbers, and must be finite, so the loop terminates. □

Theorem 5.3 (maj correctness). *For any input A satisfying its precondition, maj terminates and returns the majority element.*

Proof. Let $A \in \mathbb{N}^*$ be a list having a majority element m, and consider the execution of $maj(A)$. By 5.2, we will exit the while loop after some number of iterations (call it j) and reach line 18 and return curr[0].

$\text{ADVANTAGE}(A, m) > 0$, so by invariant 5.1, $\text{ADVANTAGE}(\text{curr}_j, m) > 0$.

By the while loop condition, when we exit the loop, $\text{len}(\text{prev}_j) = \text{len}(\text{curr}_j) = \text{len}(\text{R}(\text{prev}_j))$ (by 5.1). By 3.4, it follows that curr is uniform. Since $\text{ADVANTAGE}(\text{curr}_j, m) > 0$, curr consists entirely of some non-zero number of repetitions of majority element m. Therefore curr[0], our return value, is the majority element, as required. □