**Reasoning About Programs**

**Week 8 - PMT: BinarySearch**

**Sample Answers**

Sophia Drossopoulou and Mark Wheelhouse  
 based on exercise sheet by Krysia Broda

**1st Question:**

i) **Does not hold**For example, consider the case where a[0..a.length) < x i.e. every element of a is smaller than x. Then search(a, x)= a.length satisfies the specification, but a.length is not an index of a.

ii) **Does not hold**

For a counterexample, consider an array b=[`d’,`e’,`n’,`t’].

Then, the first occurrence of `d’ in b is at index 0.

On the other hand,  
 b[0.. **-2**) < `d’ ≤ b[**-2**..4).   
Therefore,

search(b,`d’) = -2

satisfies the specification. (In fact, any value smaller than 1 satisfies the specification.)

**2nd Question:**

i) **Does not hold**Same counterexample as in 1st Question, part (i).

ii) **Holds**

In order to prove the statement, we will call F, the index of the first occurrence of x in a, and S, the result of calling search(a,x). We ignore the distinction between a and a0 since the array does not change, and have:

**Given:**

(3) a[0..S) < x ≤ a[S..a.length) (POST)

(4) 0 ≤ S ≤a.length (POST)

(5) 0 ≤ F ≤a.length (occurs)

(6) a[F]=x (occurs)

(7) a[0..F)<x (first occurrence)

**RTS:**

() F=S

**Proof:**

Assume that F ≠ S. We will get a contradiction.

**1st Case**, (8) F < S.

Then we obtain:

(9) F  [0..S) from (5) and (8)

(10) a[F]<x from (9) and (3)  
(11) contradiction from (11) and (6)

**2nd Case**, (8)S < F.

Then we obtain:

(9) S  [0..F) from (4) and (8)

(10) a[S]<x from (9) and (7)  
(11) contradiction from (11) and (3)

Note that the requirement that 0≤S, from (4), was crucial in being able to establish (9).

iii) **Holds**

Namely, if all elements in a are greater than x, then search(a,x)=0, and otherwise, search(a,x)=a.length.  
We can prove the above more formally as follows:

**Given:**

(3) a[0..S) < x ≤ a[S..a.length) (POST)

(4) 0 ≤ S ≤a.length (POST)

(5) a[0..a.length)=y for some y (assumption)

**RTS:**

() S=a.length or S=0

**Proof:**

**1st Case**, (6) x<y.

Then we obtain:

(9) a[0..a.length)<x from (5) and (6)

(10) S=0 from (3), (4) and (9)

 from (10)

2nd  **Case**, (6) y≤x. Similar

iv) **Does not hold**

For a counterexample, consider an array b=[`a’,`n’,`t’]. Then

search(b,`f’) = 1 = search(b,`g’).

v) **Does not hold**

Use the same counterexample as in part iv).

vi) **Holds**

In order to prove the statement, we assume characters x1 an x2, with x1 ≤ x2, and will call S1 the result of calling search(a,x1) and S2 the result of calling search(a,x2):

**Given:**

(0) x1 ≤ x2 (assumption)

(1) a[0..S1) < x1 ≤ [S1..a.length) (POST for x1)

(2) 0 ≤ S1 ≤a.length (POST for x1)

(3) a[0..S2) < x2 ≤ a[S2..a.length) (POST for x2)

(4) 0 ≤ S2 ≤a.length (POST for x2)

**RTS:**

() S1 ≤ S2

**Proof:**

We will assume the opposite of (), ie that S1>S2, and will obtain a contradiction.

(5) S1>S2 (assumption)

(6) S1>0 from (5) and (4)

(7) 0 ≤ S1-1 from (6)

(8) S1-1  [S2..a.length) from (5) and (2)

(9) x2 ≤ a[S1-1] from (8) and (7) and (3)

(10) a[S1-1] < x1 from (7) and (1)

(11) contradiction from (9) and (10)

**3rd Question:**

d(a,x) **is the number of occurrences of** x **in** aNamely, assume that x appears exactly k times in a sorted array a. We call S1, rsep. S2 the result of calling search(a,x), resp. search(a,x+1):

**Given:**

(1) Sorted(a) (PRE)

(2) 0 ≤ S1 ≤a.length (POST for x)

(3) a[0..S1) < x ≤ a[S1..a.length) (POST for x)

(4) 0 ≤ S2 ≤a.length (POST for x+1)

(5) a[0..S2) < x+1 ≤ a[S2..a.length) (POST for x+1)

(6) a[S1..S1+k) = x (x occurs k times, and starts at S1, because of part ii)

(7) x < a[S1+k..a.length] (x occurs k times, and array is sorted, by 1)

**RTS:**

() S2 = S1+k

**Proof:**

(8) a[0..S1+k) < x+1 from (3) and (6)

(9) x+1 ≤ a[S1+k..a.length] from (7)

(10) S2=S1+k from (8), (9) and (5)

Note that the argument from above is valid even if x does *not* appear in the array, i.e. even if k=0.

**4th Question:**

1. ***From invariant to loop condition:***

Our invariant says  
 a[0..left) < x ≤ a(right..a.length)  
Observe that the above is equivalent with

a[0..left) < x ≤ a[right+1..a.length)  
Therefore, if we had that   
 left = right+1  
we would obtain   
 a[0..left) < x ≤ a[left..a.length).

This means that the loop condition must be  
 left ≤ right   
 and the I2 part of the invariant can be completed to be   
 0 ≤ left ≤ right+1 ≤ a.length

(Note: The invariant 0 ≤ left ≤ right+1 ≤ a.length+1 is slightly weaker. In part iv) we will see why it is too weak.)

1. ***From invariant to loop code:***

**while (**left <= right) { **(b)**

**int** middle=(left+right)/2;

**if** ( a[middle] < x) left = middle+1;

**else** right = middle-1; **(c)**

}

**iii) *From invariant to initialization:***

**int** left = 0, right = a.length-1;

Note: the ranges [0..0) and (a.length-1..a.length) are empty, therefore the invariant trivially holds.

Note also, that if we had initialized with right = a.length then we would have broken the invariant.

**iv) *Array Accesses:***

We can see that left ≤ middle ≤ right, and therefore, because of the invariant from paart i) we also obtain that 0 ≤ middle ≤ a.length-1. (Now we can see why the invariant 0 ≤ left ≤ right+1 ≤ a.length+1 is too weak, and why the initialization right = a.length would create an array access exception.)

**v) *Termination:***

The variant right-left decreases after each loop iteration (because either left increases, or right decreases. And because of the invariant, the variant has -1 as its lower bound.

**vi) *Putting it all together:***

**int** search (**char**[] a, **char** x) {  
// PRE: a.length>0  Sorted(a)  
// POST: a0[0..**r**)< x0 ≤a0[**r**..a0.length)

**int** left = 0, right = a.length-1; **(a)**

// INV: (I1) aa0  x=x0 

// (I2) 0 ≤ left ≤ right+1 ≤ a.length    
// (I3) a[0..left) < x ≤ a(right..a.length)

// VAR: right-left

**while (**left <= right) { **(b)**

**int** middle=(left+right)/2;

// left ≤ middle ≤ right **if** ( a[middle] < x) left = middle+1;

**else** right = middle-1; **(c)**

}

**// MID:** (M1) aa0  x=x0 

// (M2) 0 ≤ left ≤ a.length    
// (M3) a[0..left) < x ≤ a[left..a.length)

**return** left;

}

**vii) *Prove that the code satisfies its specification*,**Since the array is not modified we shall ignore the distinction between a0 and a.

***(a) Prove that the initialization code establishes the invariant.***

**Given:**

(1) left=0 (code)

(2) right=a.length-1 (code)

**RTS:**

(I2) 0 ≤ left ≤ right+1 ≤ a.length

(I3) a[0..left) < x ≤ a(right..a.length)

**Proof:**

(3) 0 ≤ 0 ≤ a.length ≤ a.length by arithmetic

(4) I2 holds by (3), (1) and (2)

(5) the ranges [0..0) and (a.length-1..a.length) are empty by arithmetic

(6) the ranges [0.. left) and (right..a.length) are empty by (5), (1), (2)

(4) I3 holds by (6)

For convenience, we prove b and d together:

***(b) Prove that when cond holds, the loop body preserves the invariant and also***

***(d) Prove that when cond holds, the loop body decreases the variant and that the variant   
is bounded.***

**Given:**

(0) Sorted(a) (PRE)

(1) 0 ≤ left ≤ right+1 ≤ a.length (INV)

(2) a[0..left) < x ≤ a(right..a.length) (INV)

(3) left≤right (cond)

(4) midde=(left+right)/2 (code)

(5) ( a[midde]<x  left’=middle+1  right’=right ) 

( a[midde] x  left’=left  right’=middle-1 ) (code)

**RTS:**

(R1) 0 ≤ left’ ≤ right’+1 ≤ a.length (INV)

(R2) a[0..left’) < x ≤ a(right’..a.length) (INV)

(R3) right’-left’ < right-left (decr var)

(R4) right-left  -1 (invar bound)

**Proof:**

From (3) and (4), we obtain that  
(6) left ≤ middle ≤ right by arithmetic

**1st Case:** (7) a[middle]<x

(8) left’=middle+1  right’=right by (7) and (5)

(9) middle+1≤right+1 by (6) and arithmetic

(10) left’≤right’+1 by (10) and (8)

(11) right’+1 ≤ a.length by (2) and (8)

(12) left ≤ left’ by (6) and (8)

(13) 0 ≤ left’ by (12) and (1)

(14) R1 holds by (10), (11) and (13)

(15) a[0..middle] < x by (7) and (0)

(16) a[0..middle+1) < x by (15)

(17) a[0..left’) < x by (16) and (8)

(18) R2 holds by (17), (2) and (8)

(19) right’-left’ = right -(middle+1) by (8)

(20) right’-left’ < right-middle ≤ right-left by (6) and (19)

(21) R3 holds by (20)

(22) right-left  -1 by (1) and arithm.

(23) R4 holds by (22)

Note, that the proof that the variant is bounded only considers the invariant, and does not consider the loop body.

**2nd Case:** (7) x ≤ a[middle]

Similar to 1st Case.

***(c) Prove that Invariant and cond imply MID.***

**Given:**

(0) Sorted(a) (PRE)

(1) 0 ≤ left ≤ right+1 ≤ a.length (INV)

(2) a[0..left) < x ≤ a(right..a.length) (INV)

(3) left>right (NOT cond)

**RTS:**

(R1) 0 ≤ left ≤ a.length (INV)

(R2) a[0..left) < x ≤ a[left..a.length) (INV)

**Proof:**

(4) R1 holds by (1)

(5) left = right+1 by (1) and (3)

(6) The range (right..a.length) is the same as   
 the range [right+1..a.length) .  
(7) The range (right..a.length) is the same as   
 the range [left..a.length) . by (6) and (5)

(4) R2 holds by (2) and (7)

***(e) Prove that all array accesses are valid.***

**Given:**

(0) Sorted(a) (PRE)

(1) 0 ≤ left ≤ right+1 ≤ a.length (INV)

(2) a[0..left) < x ≤ a(right..a.length) (INV)

(3) left≤right (cond)

(4) midde=(left+right)/2 (code)

**RTS:**

(R1) 0 ≤ middle < a.length (valid access)

**Proof:**

From (3) and (4), we obtain that  
(5) left ≤ middle ≤ right by arithmetic

(6) R1 holds by (5) and (1)