1 Proof of Correctness for Linear Search

The linear search algorithm is given below.

Algorithm LinearSearch(x, L, n)
Input: Array L, value x, and number n of elements in L
Output: Position i, $0 \leq i < n$ such that $L[i] = x$, or -1 if $x \notin L$

$i \leftarrow 0$
while $(i < n)$ **and** $(L[i] \neq x)$ **do** $i \leftarrow i + 1$
if $i < n$ **then return** i
else return -1

To prove that an algorithm is correct, we need to show two things: (1) that the algorithm terminates, and (2) that it produces the correct output.

1) **Algorithm LinearSearch terminates after a finite amount of time**.

   To prove the above claim, note that at the beginning variable i takes value zero, and after each iteration of the **while** loop the value of i increases by 1. If the **while** loop performed an infinite number of iterations, then the value of i would grow without bound; however, this is not possible as one of the conditions of the **while** loop is $i < n$. Thus the **while** loop will end as soon as $i = n$.

2) **Algorithm LinearSearch outputs the correct answer**.

   In the second to last line the algorithm returns the value i. Note that this is the correct value to return since as the **while** loop ended and $i < n$, this implies that $L[i] = x$.

   To show that the algorithm returns the value -1 if and only if the value x is not in the array, note that at the beginning i takes the value 0 and in each iteration of the loop the value of i increases by 1. This means that the algorithm compares the value of x against all values stored in L. If at the end of the **while** loop the condition $i < n$ is false, this means that $i = n$, so the algorithm has considered all values in L and none of them is equal to x. Hence, the value returned in the last line of the algorithm is also correct.

2 Proof of Correctness for Binary Search

Algorithm BinarySearch is given below.

Algorithm BinarySearch($L, x, \mathit{first}, \mathit{last}$)
Input: Array $L[\mathit{first}, \mathit{last}]$ and value x.
Output: Position i, $0 \leq i < n$ such that $L[i] = x$, or -1 if $x \notin L$

if *first* $>$ *last* **then return** -1 // Size of L is zero
else {

$middle \leftarrow \lfloor \frac{first+last}{2} \rfloor$
if $L[middle] = x$ **then return** $middle$
else if $L[middle] < x$ **then return** BinarySearch$(L, x, middle + 1, last)$
else return BinarySearch$(L, x, first, middle - 1)$
}

1) **Algorithm BinarySearch terminates after a finite amount of time**

Every time that the algorithm calls itself recursively, the value of one of its parameters changes: For the first call the value of *first* changes to $middle + 1$ and in the second call the value of *last* changes to $middle - 1$. Hence, in every recursive call the size of the array that the algorithm still needs to consider decreases in size by at least one, because the value $L[middle]$ is not considered anymore in any of the two calls.

Since the size of the array decreases by at least 1 in each recursive call and the algorithm will not make more recursive calls when the size of the array is zero, then the algorithm cannot perform an infinite number of recursive calls.

2) **Algorithm BinarySearch outputs the correct answer**

There are 4 **return** statements in the algorithm. The first 2 **return** statements return either *middle* or -1; the last 2 simply return the value computed by the recursive calls. Hence, we only need to show that the values returned by the algorithm in the first two **return** statements are correct.

Note that the value, *middle*, returned in the fourth line of the algorithm is the correct one, as $L[middle] = x$. To show that the value returned in the fist line is also correct, notice that

- if $L[middle] < x$, then every value in L located in the first half of the array (including $L[middle]$) is smaller than x and thus by BinarySearch discarding all these values, no copy of x is lost;
- if $L[middle] > x$ then every value in L located in the second half of the array (including $L[middle]$) is larger than x and thus by BinarySearch discarding all these values, no copy of x is lost;

As the algorithm never discards any copies of the value x stored in L, if at the end array L is empty this means that the value x was not stored in L and so the algorithm correctly returns the value -1.