Recursive Algorithm Correctness (Continued)

Example 1 (Binary search algorithm)**.** *Consider the following recursive implementation of binary search algorithm:*

```
function RecBSearch(x, A, s, f)
    if $s == f$ then
        if $x == A[s]$ then
            return $s$
        else
            return $-1$
        end if
    else
        $m = (s + f) / 2$                          ▷ Integer Division
        if $x \leq A[m]$ then
            return RecBSearch(x, A, s, m)
        else
            return RecBSearch(x, A, m + 1, f)
        end if
    end if
end function
```

Precondition:

1. Elements of A comparable with each other and with x
2. Assume array indices start at 0 and hence $0 \leq s \leq f < length(A)$
3. Array A issorted in nondecreasing order ($A[s] \leq \cdots \leq A[f]$)

Postcondition: RecBSearch(x,A,s,f) terminates and returns index p such that:

1. $s \leq p \leq f$ or $p = -1$
2. If $s < p$, then $A[p-1] < x$
3. If $s \leq p \leq f$, then $x = A[p]$

Now, Let's prove the correctness of this algorithm.

Proof. By induction on size $n = f + 1 - s$, we prove (precondition and execution) implies (termination and postcondition). Inductive structure of proof will follow recursive structure of algorithm.

Base case: $n = 1$, i.e., $s = f$. Then, algorithm terminates (lines 2-7 contain no loop or call), and returns s if $x = A[s]$, -1 if $x \neq A[s]$, which satisfies postcondition.

Induction Step: Let $n > 1$ and suppose postcondition holds after execution for all inputs of size k that satisfy precondition, for $1 \leq k < n$ (IH). Consider call RecBSearch(x,A,s,f) when $f + 1 - s = n \geq 2$. Test on line 2 fails, so $s < f$ (since $s \leq f$ by precondition and $s \neq f$ by negation of test) and algorithm executes line 9. Next, test on line 10 executes.

Case 1 ($x \leq A[m]$): Because $m < f$ then $m + 1 - s < f + 1 - s$ and hence by IH, RecBSearch(x,A,s,m) returns index p such that:

1. $s \leq p \leq m$ or $p = -1$
2. if $s < p$, then $A[p-1] < x$

3. if $s \leq p \leq m$, then $x = A[p]$

Hence,

1. $s \leq p \leq f$ (since $m < f$) or $p = -1$
2. if $s < p$, then $A[p-1] < x$ since $p \leq m$ and because of IH
3. becuase we recursed on the first half then if $s \leq p \leq f$ then $p \leq m$, and by IH for $s \leq p \leq m$ then $x = A[p]$

Therefore, current call satisfies postcondition.

Case 2 ($A[m] < x$): Because $s \leq m$ then $s < m+1$ so $f+1-(m+1) < f+1-s$ and hence by IH, RecBSearch(x,A,$m+1$,f) returns index p such that:

1. $m+1 \leq p \leq f$ or $p = -1$
2. if $m+1 < p$, then $A[p-1] < x$
3. if $m+1 \leq p \leq f$ then $x = A[p]$

Hence,

1. $s \leq p \leq f$ (since $s < m+1$) or $p = -1$
2. if $s < p$ then we know that $m+1 \leq p$ since we recursed on the second half. By IH $A[p-1] < x$ for $m+1 < p$ and by the test of line 10, $A[m] < x$ in this case. Therefore, if $s < p$ then $A[p-1] < x$
3. if $s \leq p \leq f$ then $m+1 \leq p \leq f$ since we recursed on the second half and by IH $x = A[p]$

Therefore, current call satisfies postcondition. In all cases, current call satisfies postcondition. Therefore, by induction, RecBSearch is correct. □

Example 2. *In this example we prove the correctness of MergeSort algorithm.*

```
function MergeSort(A,s,f)
    if $s == f$ then
        return
    else
        $m = (s+f)/2$                                        ▷ Integer Division
        MergeSort(A,s,m)
        MergeSort(A,m+1,f)
        # merge sorted $A[s..m]$ and $A[m+1..f]$ back into $A[s..f]$
        for $i = s, \cdots, f$ do
            B[i] = A[i]
        end for
        $c = s$
        $d = m+1$
        for $i = s, \cdots, f$ do
            if $d > f$ or ($c \leq m$ and B[c] < B[d]) then
                A[i] = B[c]
                $c = c+1$
            else                                ▷ $d \leq f$ and ($c > m$ or B[c] $\geq$ B[d])
                A[i] = B[d]
                $d = d+1$
            end if
        end for
```

```
23:     end if
24: end function
```

Precondition:

1. $s, f \in \mathbb{N}$, $0 \leq s \leq f < length(A)$
2. elements of $A[s..f]$ comparable with each other

Postcondition: $A[s..f]$ contains same elements as before, but sorted in non-decreasing order ($A[s] \leq \cdots \leq A[f]$)

Proof. By induction on size $n = f + 1 - s$, we prove precondition and execution implies termination and postcondition, for all inputs of size n. Once again, the inductive structure of proof will follow recursive structure of algorithm.

Base case: Suppose (A,s,f) is input of size $n = f - s + 1 = 1$ that satisfies precondition. Then, $f = s$ so algorithm terminates and returns A unchanged (on line 2), which satisfies postcondition.

Induction Step: Suppose $n > 1$ and, for $1 \leq k < n$, for all inputs of size k that satisfy precondition, algorithm terminates and postcondition holds after execution (IH). Suppose (A,s,f) is input of size $n = f - s + 1 > 1$ that satisfies precondition, and consider call MergeSort(A,s,f). Test on line 2 fails because $f - s + 1 > 1$ iff $f > s$ and hence the algorithm executes line 5. Since $s \leq \lfloor \frac{s+f}{2} \rfloor < f$, IH implies that MergeSort(A,s,m) terminates and the output $A[s..m]$ contains same elements as input $A[s..m]$ but sorted in non-decreasing order. For the same reason, MergeSort(A,$m + 1$,f) terminates and output $A[m + 1..f]$ contains same elements as input $A[m + 1..f]$ but sorted in non-decreasing order. Lines 9-11 copies $A[s..f]$ into $B[s..f]$ (exercise: prove this). Lines 12-22 merge $B[s..m]$ and $B[m+1..f]$ into $A[s..f]$, which satisfies postcondition. This last statement requires formal proof, but we haven't learned how to prove the correctness of iterative algorithms yet! Hence, we skip this part of the proof for now and we will come back to it later. □

From these two example, you can see that sometimes it becomese tedious to use a proof by induction as it requires you to write down a lot of details. There is another way of proving the correctness which requires less elaboration and minimizes the writing effort. In this technique we have the following steps:

1. Write down the correct specification (pre/post-conditions)
2. Specify what is the size of an instance for the purpose of induction
3. List all program paths to a return point. For each path, argue why that path terminates and the postcondition is true at the return point
   I. if the path includes no recursive calls, this is usually straightforward reasoning. Don't forget the termination part.
   II. if the path includes recursive calls then:
      - Show that the precondition for each recursive call holds just before the call is made
      - Argue (assuming the postconditions of the recursive calls hold) why the postcondition of the function holds
      - Show that each recursive call is made on a smaller-sized instance. Based on this, argue why the function terminates (assuming that the recursive calls terminate)

This technique is called proving that the specification is inductive.

Example 3. *Let's prove the correctness of RecBSearch by proving that the specification is inductive.* Just to recap:
Precondition:

1. Elements of A comparable with each other and with x

2. Assume array indices start at 0 and hence $0 \leq s \leq f < length(A)$
3. Array A issorted in nondecreasing order ($A[s] \leq \cdots \leq A[f]$)

Postcondition: RecBSearch(x,A,s,f) terminates and returns index p such that:

1. $s \leq p \leq f$ or $p = -1$
2. If $s \leq p \leq f$, then $x = A[p]$

Note that this postcodition is slightly weaker than the one provided in the previous example. But it will still serve the purpose of this example. *Path 1 (2,3,4):* Since $f = s$, and $p = s$, then $s \leq p \leq f$ (first item in the postcondition is true). Since line 3 is true, $x = A[p]$ (second item in the postcondition is true). Therefore, the postcondition is satisfied. It is obvious that the algorithm terminates in this case.

Path 2 (2,5,6): In this case $p = -1$ and hence the first postcondition is true because one element of disjunction is true. Since the if condition doesn't hold the the second postcondition is also true.

Path 3 (8,9-11): In this case $s \neq f$. Hence, $s \leq m < f$ and by precondition $f < length(A)$. Since $A[s..f]$ is sorted then $A[m..f]$ should also be sorted and by preconditions the elements in $A[m..f]$ are comparable to A. From our arguments so far, we can see that the preconditions of input to the recursive call in line 11 holds. Since $m - s + 1 < f - s + 1$, then we are recursing on an instance of smaller size. Hence if the recursive call terminates, by postconditions, it should either return -1 or an index p for which we have $s \leq p \leq m$ and $x = A[p]$. If the former holds then the postcondition of our bigger instance also holds. If the latter holds, then the postcondition of our bigger instance also holds because $s \leq p \leq m < f$ and $x = A[p]$.

Path 4 (8,9,12-13): This case is similar to the previous case. We only need to consider the fact that since $s \neq f$ and $s \geq 0$ then $0 \leq s \leq m < m + 1 \leq f < length(A)$. We can then argue similar to Path 3 that preconditions of the recursive call holds and then argue that postconditions should hold which means that either an index of -1 is returned or an index $m + 1 \leq p \leq f$ is returned and $x = A[p]$. It is obvious that if former holds then the postcondition of our bigger instance holds and if latter holds, then $s \leq m < m + 1 \leq p \leq f$ and $x = A[p]$ which again means that the postcondition of our bigger instance holds.

Exercise. *Try to prove the correctness of MergeSort by proving that the specification is inductive (at one point, you need to prove that the for loops are correct, for now assume that they are correct and they terminate).*