Proving the Correctness of a Simple Recursive Algorithm
Solutions for Suggested Exercise

For this exercise, you were asked to recall the problem "Maximal Element in Part of an Integer Array", considered in the "lecture" part of this unit, and you were asked to consider the following recursive algorithm.

```
maxInRange2 ( integer[] A, integer low, integer high ) {
1. if (low == high) {
2.   return A[low]
   } else {
3.   return max(maxInRange2(A, low, high − 1), A[high])
   }
}
```

1. You were asked to sketch (or, better yet, write out in full) a proof that this algorithm also correctly solves the "Maximal Element in Part of an Integer Array" problem.

   As part of this you were asked what form(s) of mathematical induction can be used to prove this claim.

   Solution: Recall the precondition and postcondition for the "Maximal Element in Part of an Array" problem. One can see, by inspection of the pseudocode, that this algorithm does not access or modify data in any way that is unmentioned in the specification of requirements for this problem (it has no "undocumented side-effects") — so a consideration of this specification of requirements confirms that it is necessary and sufficient to prove the following in order to prove the correctness of the `maxInRange2` algorithm:

   Claim: If the `maxInRange2` algorithm is executed with an input array `A` with some positive length n, and with integer inputs `low` and `high` such $0 \leq \texttt{low} \leq \texttt{high} \leq n-1$, then this execution of the algorithm eventually ends, and the value

   $$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low}+1], \ldots, \texttt{A}[\texttt{high}])$$

   is returned as output.

Proof: By induction on $\texttt{high} - \texttt{low}$. The standard form of mathematical induction will be used.

Basis: Suppose first that $\texttt{high} - \texttt{low} = 0$, so that $0 \leq \texttt{low} = \texttt{high} \leq n - 1$. Then — during an execution of the maxInRange2 algorithm with these inputs — the test at line 1 is passed, so that step 2 is executed after that. One can see by examination of this line of the pseudocode that

$$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high}]) = \texttt{A}[\texttt{low}]$$

is then returned as output as the execution of the algorithm ends — establishing the claim in this case.

Inductive Step: Let k be an integer such that $k \geq 0$. It is necessary and sufficient to use the following

> Inductive Hypothesis: If the maxInRange2 algorithm is executed with an integer array A with some positive length n as input, and with integer inputs low and high such that $0 \leq \texttt{low} \leq \texttt{high} \leq n - 1$, where $\texttt{high} - \texttt{low} = k$, then this execution of the algorithm eventually ends, and the value
>
> $$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high}])$$
>
> is returned as output.

to prove the following

> Inductive Claim: If the maxInRange2 algorithm is executed with an integer array A with some positive length n as input, and with integer inputs low and high such that $0 \leq \texttt{low} \leq \texttt{high} \leq n - 1$, where $\texttt{high} - \texttt{low} = k + 1$, then this execution of the algorithm eventually ends, and the value
>
> $$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high}])$$
>
> is returned as output.

With that noted, consider an execution of the maxInRange2 algorithm with an integer array A as described above as input, along with integer inputs low and high such that $0 \leq \texttt{low} \leq \texttt{high} \leq n - 1$, where $\texttt{high} - \texttt{low} = k + 1$.

Since $k \geq 0$, $k + 1 \geq 1$ and it follows that $\texttt{low} \neq \texttt{high}$. The test at line 1 therefore fails and the execution continues with the step at line 3. Now, since $\texttt{high} - \texttt{low} = k + 1$, $(\texttt{high} - 1) - \texttt{low} = k \geq 0$ so that $0 \leq \texttt{low} \leq (\texttt{high} - 1) \leq n - 1$, with $(\texttt{high} - 1) - \texttt{low} = k$, and it follows by the Inductive Hypothesis that the execution of the maxInRange2 algorithm included in this step eventually halts, with

$$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high} - 1])$$

returned as output. One can now see by the inspection of the pseudocode at line 3 that the current execution also halts after this step, with

$$\max(\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low}+1], \texttt{A}[\texttt{high}-1]), \texttt{A}[\texttt{high}]) \\ = \max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low}+1], \ldots, \texttt{A}[\texttt{high}])$$

returned as output — as needed to establish the Inductive Claim, complete the Inductive Step, and establish the claim. □

As shown above, the ***standard*** form of mathematical induction can be used to prove the correctness of this algorithm. The ***strong*** form of mathematical induction could be used too — and, in this case, a proof using the strong form of mathematical induction would probably not look very different from the above one: The "Inductive Hypothesis" would just be a little more complicated.

2. You were next asked to give a bound function for this recursive algorithm and explain, briefly, why it is correct.

   Solution:
   The function $f(\texttt{low}, \texttt{high}) = \texttt{high} - \texttt{low}$ is a bound function for the `maxInRange2` algorithm.

   In order to confirm this, it is sufficient to check that this function satisfies all the properties included in the definition of a "bound function for a recursive algorithm":

   - This is a well-defined integer-valued total function of (some of) the algorithm's inputs (there is no global data being accessed that must be considered and the function's value does not depend on the input array `A`).
   - The only way that this algorithm can call itself is recursively is by executing the step at line 3. Since the input `low` is not changed and the input `high` is replaced by $\texttt{high} - 1$, the value of the function is certainly decreased by at least one (indeed, by exactly one) when this algorithm calls itself recursively.
   - Suppose that this algorithm is executed with the precondition for the "Maximal Element in Part of an Integer Array" problem satisfied — so that $\texttt{low} \leq \texttt{high}$. Suppose, as well, that the value of the function f is less than or equal to zero — so that $\texttt{low} \geq \texttt{high}$ as well. Then $\texttt{low} = \texttt{high}$, so that the test at line 1 is passed and execution of the algorithm ends after the step at line 2 is executed, without the algorithm having called itself recursively.

   Thus the function $f(\texttt{low}, \texttt{high}) = \texttt{high} - \texttt{low}$ satisfies all the required properties for a "bound function of a recursive algorithm", as needed.

   Note, as well, that the proof shown above is a proof by induction on the value of this function.

3. You were next asked to give a set of assertions for this recursive algorithm that can be used to document a proof that it is correct.

   Solution: There is certainly more than one correct "solution" for this problem: Information can almost always be given in more than one way. With that noted, one solution is as follows.

   - An assertion appearing immediately *before* line 1 might be as follows:

     Assertion:

     (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied — so that `A` is an integer input array with some positive length n, and `low` and `high` are integer inputs such that $0 \leq \texttt{low} \leq \texttt{high} \leq n-1$.

   - An assertion appearing immediately *after* line 1 might be as follows:

     Assertion:

     (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied.
     (b) $\texttt{low} = \texttt{high}$.

   - An assertion appearing immediately *after* line 2 might be as follows:

     Assertion:

     (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied.
     (b) $\texttt{low} = \texttt{high}$.
     (c) The value

     $$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low}+1], \ldots, \texttt{A}[\texttt{high}]) = \texttt{A}[\texttt{low}]$$

     has been returned as output.

   - An assertion appearing immediately *before* line 3 might be as follows.

     Assertion:

     (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied.
     (b) $\texttt{high} \geq \texttt{low} + 1$.

- An assertion appearing immediately *after* line 3 might be as follows.

  Assertion:

  (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied.
  (b) $\texttt{high} \geq \texttt{low} + 1$.
  (c) The value

  $$\max(\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high} - 1]), \texttt{A}[\texttt{high}]) = \max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high}])$$

  has been returned as output.

- Even though it might seem silly — this point in the pseudocode never actually gets reached — one might also include the following assertion at the end of the `if-then-else` test — that is, between the two "curly braces" at the end of the pseudocode, just to close things off:

  Assertion:

  (a) The precondition for the "Maximal Element in Part of an Array" problem is satisfied.
  (b) The value

  $$\max(\texttt{A}[\texttt{low}], \texttt{A}[\texttt{low} + 1], \ldots, \texttt{A}[\texttt{high}])$$

  has been returned as output — so that the postcondition for the "Maximal Element in Part of an Array" problem has now been satisfied.

Note: One not-so-good thing about this solution is that ***it is far too long***: The code itself would be lost within all this in-line documentation. Indeed, it is not clear that very many (if any) assertions are needed, as in-line documentation, when an algorithm is as simple as this one. It might better to describe the precondition and postcondition for the problem being solved and — possibly — provide a reference to a proof of the correctness of the algorithm, if one has been published.

One somewhat better thing is about this is that — between the identification of a bound function (which should also be listed in inline documentation) and the above assertions, ***a proof of the correctness of the algorithm has been fully documented***. Someone reading the source code does not (necessarily) need to look elsewhere to understand why the algorithm is correct.

Students proceeding to CPSC 413 and various 500-level courses in computer science will be introduced to various algorithms whose correctness is not at all obvious. These might — possibly — serve as better motivation for the kind of in-line documentation being considered in the last two problems.

4. You were next asked to give trace(s) of execution and a recursion tree for this algorithm when it is executed on an input array A with length 5 such that

$$\texttt{A}[0] = 8, \quad \texttt{A}[1] = 10, \quad \texttt{A}[2] = 4, \quad \texttt{A}[3] = 24, \quad \text{and} \quad \texttt{A}[4] = 3,$$

and with inputs $\texttt{low} = 0$ and $\texttt{high} = 4$.

Solution: Traces of execution for this recursive algorithm, when it is executed on the above inputs, are as follows. Since the array A is never changed its lengths and entries will not be repeated.

Trace of Execution #1: The algorithm is executed with the above array A and the values $\texttt{low} = 0$ and $\texttt{high} = 4$ as input.

1. Since $\texttt{low} = 0 < 4 = \texttt{high}$ the test at line 1 fails. Execution of the algorithm continues with the step at line 3.
2. (a) The algorithm is called recursively with the same input array A, with $\texttt{low} = 0$, and with $\texttt{high} = 3$. This execution terminates with $\texttt{A}[3] = 24$ returned — see Trace of Execution #2 for details.
   (b) The execution of the algorithm then ends. Since $\texttt{A}[3] = 24 > 3 = \texttt{A}[4]$, the value $\texttt{A}[3] = 24$ is returned as output.

Trace of Execution #2: The algorithm is executed with the above array A and the values $\texttt{low} = 0$ and $\texttt{high} = 3$ as input.

1. Since $\texttt{low} = 0 < 3 = \texttt{high}$ the test at line 1 fails. Execution of the algorithm continues with the step at line 3.
2. (a) The algorithm is called recursively with the same input array A, with $\texttt{low} = 0$, and with $\texttt{high} = 2$. This execution terminates with $\texttt{A}[1] = 10$ returned — see Trace of Execution #3 for details.
   (b) The execution of the algorithm then ends. Since $\texttt{A}[3] = 24 > 10 = \texttt{A}[1]$, the value $\texttt{A}[3] = 24$ is returned as output.

Trace of Execution #3: The algorithm is executed with the above array A and the values $\texttt{low} = 0$ and $\texttt{high} = 2$ as input.

1. Since $\texttt{low} = 0 < 2 = \texttt{high}$ the test at line 1 fails. Execution of the algorithm continues with the step at line 3.
2. (a) The algorithm is called recursively with the same input array A, with $\texttt{low} = 0$, and with $\texttt{high} = 1$. This execution terminates with $\texttt{A}[1] = 10$ returned — see Trace of Execution #4 for details.
   (b) The execution of the algorithm then ends. Since $\texttt{A}[1] = 10 > 4 = \texttt{A}[2]$, the value $\texttt{A}[1] = 10$ is returned as output.

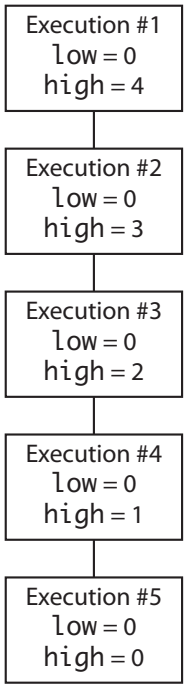
Trace of Execution #4: The algorithm is executed with the above array A and the values $\texttt{low} = 0$ and $\texttt{high} = 1$ as input.

1. Since $\texttt{low} = 0 < 1 = \texttt{high}$ the test at line 1 fails. Execution of the algorithm continues with the step at line 3.
2. (a) The algorithm is called recursively with the same input array A, with $\texttt{low} = 0$, and with $\texttt{high} = 0$. This execution terminates with $\texttt{A}[0] = 8$ returned — see Trace of Execution #5 for details.
   (b) The execution of the algorithm then ends. Since $\texttt{A}[1] = 10 > 8 = \texttt{A}[0]$, the value $\texttt{A}[1] = 10$ is returned as output.

Trace of Execution #5: The algorithm is executed with the above array A and the values $\texttt{low} = 0$ and $\texttt{high} = 0$ as input.

1. Since $\texttt{low} = 0 = \texttt{high}$ the test at line 1 passes. Execution of the algorithm continues with the step at line 2.
2. The value $\texttt{A}[0] = 8$ is then returned as output.

A corresponding recursion tree is as follows.



One might wonder ***why*** these might be of interest. Traces of executions can be surprisingly useful for code inspection and debugging. Recursion trees can be valuable for the analysis of various recursive algorithms.