Problem Set 1: Due Tues, Sept 13

Instructions: Turn in your answers to the questions below in class on or before the due date. You must include each question below as part of your answer (including the question number). Be sure your answers are legible. Note that each question is worth 1 point (for a total of 10 points).

1. Write an algorithm (using pseudocode) that takes an integer array \( A[] \) of any length \( n \geq 0 \) and returns true if the array is in ascending (sorted) order and false otherwise.

2. Show that your algorithm in Question 1 is correct using a loop invariant. Give the loop invariant and show that the invariant satisfies the three criteria given in class.

3. Consider the following insertion sort algorithm. Show the result of the algorithm on the array \( \{40, 10, 30, 50, 20\} \) after each iteration of the algorithm’s outer (for) loop.

Algorithm: InsertionSort

\[
\begin{align*}
\text{Input:} & \quad \text{An integer array } A[] \text{ of size } n \geq 0. \\
\text{Result:} & \quad \text{The elements of } A \text{ are in ascending order.} \\
& \text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do} \\
& \quad \text{key } \leftarrow A[i] \\
& \quad j \leftarrow i - 1 \\
& \quad \text{while } j \geq 0 \text{ and } A[j] > \text{key} \text{ do} \\
& \quad \quad A[j + 1] \leftarrow A[j] \\
& \quad \quad j \leftarrow j - 1 \\
& \quad A[j + 1] \leftarrow \text{key}
\end{align*}
\]

4. Assuming that the inner loop is correct (i.e., correctly places the \( i \)-th value into the subarray \( A[0 .. i] \)), show that the algorithm is correct using a loop invariant for the outer (for) loop. Give the loop invariant and show that the invariant satisfies the three criteria given in class.

5. Assuming primitive operations of assignment, comparison, addition, and array access (e.g., \( A[i] \)), where each operation has a cost of one time unit: (a) give the detailed cost as a function \( T(n) \) for the “Sum” algorithm below; (b) state why using Big-\( \Theta \) is appropriate for this algorithm; (c) give the Big-\( \Theta \) cost based on your \( T(n) \) function; and (d) show that the Big-\( \Theta \) cost is correct (based on the definition of Big-\( \Theta \)).
Algorithm: Sum

**Input:** An integer array $A[]$ of size $n \geq 0$.

**Result:** The sum of all elements of $A$ or 0 if $A$ is empty.

1. $\text{sum} \leftarrow 0$
2. $i \leftarrow 0$
3. while $i < n$ do
4.   $\text{sum} \leftarrow \text{sum} + A[i]$
5.   $i \leftarrow i + 1$
6. return $\text{sum}$

6. Assuming primitive operations of assignment, comparison, addition, and array access (e.g., $A[i]$), where each operation has a cost of one time unit: (a) give the detailed cost as a function $T(n)$ for the “FindMax” algorithm below (note that there is both a best and worst case); (c) give the Big-$\Omega$ cost based on your $T(n)$ function and show that it is correct (using the definition of Big-$\Omega$); and (c) give the Big-O cost based on your $T(n)$ function and show that it is correct (using the definition of Big-O).

Algorithm: FindMax

**Input:** An array $A[]$ of size $n \geq 0$ of positive integers.

**Output:** The maximum value of all elements of $A[]$ or 0 if $A[]$ is empty.

1. $\text{max} \leftarrow 0$
2. for $i \leftarrow 0$ to $n - 1$ do
3.   if $A[i] > \text{max}$ then
4.     $\text{max} \leftarrow A[i]$
5. return $\text{max}$

7. Use the insertion sort algorithm above to answer the following questions.

(a). Assume we are only interested in counting array comparisons (i.e., the expression “$A[j] > \text{key}$” on line 4, which counts as a single operation). Give the best case scenario, worse case scenario, and corresponding cost function $T(n)$.

(b). Assume we are only interested in counting operations that move data into an array (i.e., line 2 counts as a single operation, line 5 counts as a single operation, and line 7 counts as a single operation). Give the best case scenario, worse case scenario, and corresponding cost function $T(n)$.

8. Develop a (recursive) divide and conquer algorithm to find the largest value in a given array of positive integers. The input to your algorithm should be an array $A[]$ of size $n \geq 0$ consisting of
positive integers (similar to Question 6). Your algorithm should return the smallest value in $A[][]$ or else 0 if $A[][]$ is empty.

9. Give the cost of your algorithm in Question 8 as a recurrence relation $T(n)$ and determine its Big-O complexity using the (divide and conquer) master theorem. Note your recurrence relation must be in “normal” recurrence form.

10. As noted in Question 7, insertion sort has both a best and worst case. Assuming we are only counting comparisons as in 7(a), determine the worst case cost of insertion sort as a recurrence relation $T(n)$. Your recurrence must be in “normal” recurrence form. Use the (reduce and conquer) master theorem to determine the Big-O time complexity.