Homework 2 Solutions
CSE 21 Summer Academy 2016

For the next three problems, refer to the two algorithms below:

```
BubbleSort( A[1, ..., n], array of integers)
1.  For i ← 1 To n-1
2.     For j ← 1 To n-i
3.        If A[j] > A[j+1] Then
4.           Interchange A[j] and A[j+1]
```

```
RevisedBubbleSort( A[1, ..., n], array of integers)
1.  For i ← 1 To n-1
2.     done ← true
3.     For j ← 1 To n-i
4.        If A[j] > A[j+1] Then
5.           Interchange A[j] and A[j+1]
6.           done ← false
7.     If done Then Break
```

1. (a) (2 points) How many comparisons does BubbleSort make on an array of size n that is already sorted from smallest to largest?

   Solution: On the ith iteration, BubbleSort makes i comparisons. In total, the algorithm does $1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}$ comparisons.

   (b) (2 points) How many comparisons does BubbleSort make on an array of size n that is sorted from largest to smallest?

   Solution: The answer is the same, $1+2+\cdots+(n-1) = \frac{(n-1)n}{2}$ comparisons. For this particular algorithm (BubbleSort) the number of comparisons is independent of the input.

   (c) (2 points) Explain the difference between BubbleSort and RevisedBubbleSort.

   Solution: RevisedBubbleSort breaks out of the outer loop if no swaps were performed in the inner loop. That is, it recognizes when the array is sorted and terminates early if it makes a pass through the array without needing to swap anything.

(d) (2 points) How many comparisons does RevisedBubbleSort make on an array of size n that is already sorted from smallest to largest?

Solution: RevisedBubbleSort makes $n-1$ comparisons on an array of size n that is sorted from smallest to largest. This is the number of comparisons made for the first iteration of the outer loop. No other iterations are done because no swaps were made.

(e) (2 points) How many comparisons does RevisedBubbleSort make on an array of size n that is sorted from largest to smallest?

Solution: RevisedBubbleSort makes $\frac{(n-1)n}{2}$ comparisons on an array of size n that is sorted from largest to smallest. On the i-th pass, the largest element must get swapped all the way from the front of the array to the back. Thus, every comparison results in a swap, and there is no early termination.

2. (10 points) Prove the following loop invariant for BubbleSort.
After the i^{th} pass, positions $A[n-i+1,\ldots,n]$ contain the i largest elements of the input, in sorted order.

Solution: We do a proof by induction on i, the number of outer loop iterations. When $i=0$, before we run the outer loop, the last 0 positions contain the 0 largest elements in sorted order. This is trivially true because the last 0 positions is the empty set and anything is true of the empty set.

For the induction hypothesis, suppose for some value of i, that after the i^{th} pass, positions $A[n-i+1,\ldots n]$ contain the i largest elements of the input, in sorted order. We must show that after the $(i+1)^{st}$ pass, positions $A[n-i,\ldots n]$ contain the $i+1$ largest elements of the input, in sorted order.

During the $(i+1)^{st}$ iteration of the outer loop, the algorithm finds the maximum value among the first $n-i$ elements of the array and places it in position $A[n-i]$. We can see this because the inner loop compares consecutive elements and swaps elements as necessary so that the larger element of the pair ends up further to the right. Whenever the maximum value among the first $n-i$ elements is encountered, it will be swapped all the way to the right as far as possible, which is position $A[n-i]$.

Thus after the $(i+1)^{st}$ iteration of the outer loop, $A[n-i]$ is larger than all of $A[1]$ through $A[n-i-1]$. At the same time, by the inductive hypothesis, the last i positions of A contain the i largest elements. Since $A[n-i]$ is not among them, it is not one of the i largest elements. But it is larger than everything in $A[1]$ through $A[n-i-1]$, which means it is the $i+1$-st largest element. This says that positions $A[n-i,\ldots n]$ contain the $i+1$ largest elements of the input, in sorted order, which is what we wanted to show.

3. (10 points) List all possible arrays containing the numbers 1, 2, 3, 4, and 5 for which RevisedBubbleSort will terminate with a value of $i=2$.

Solution: There are 15 arrays:
15234
14235
13245
13254
12354
12435
12534
51234
41235
31245
31254
21345
21354
21435
21534

4. In this problem, we are given a sequence $a_1, a_2, \ldots, a_n$ of integers and we want to return a list of all terms in the sequence that are greater than the sum of all previous terms of the sequence. For example, on an input sequence of $1, 4, 6, 3, 2, 20$, the output should be the list $1, 4, 6, 20$. The following algorithm solves this problem.

   procedure PartialSums($a_1, a_2, \ldots, a_n$: a sequence of integers with $n \geq 1$)

   ```
   1.    $total := 0$
   2.    Initialize an empty list $L$.
   3.    for $i := 1$ to $n$
   4.        if $a_i > total$
   5.            Append $a_i$ to list $L$.
   6.        $total := total + a_i$
   7.    return $L$
   ```

   (a) (6 points) Prove the following loop invariant by induction on the number of loop iterations:

   Loop Invariant: After the kth iteration of the **for** loop, $total = a_1 + a_2 + \cdots + a_k$ and L contains all elements from $a_1, a_2, \ldots, a_k$ that are greater than the sum of all previous terms of the sequence.

   Solution: The base case is when $k = 0$, before the loop ever iterates. In line 1, *total* is set to 0, which is the value of the sum $a_1 + a_2 + \cdots + a_0$

because this sum has no terms. In line 2, L is set to the empty list, which does contain all elements from the set $\{a_1, a_2, \dots, a_0\}$ that are greater than the sum of all previous terms of the sequence, because this set is empty and anything is true of the empty set. Thus, the invariant is true when $k = 0$.

For the induction step, let k be a positive integer. Suppose that we have gone through the loop $k-1$ times, and that $total = a_1 + a_2 + \cdots + a_{k-1}$ and L contains all elements from $a_1, a_2, \dots, a_{k-1}$ that are greater than the sum of all previous terms of the sequence. On the kth iteration of the loop, the value of i is k. In lines 4 and 5, a_k is added to list L if and only if $a_k > total$, which equals $a_1 + a_2 + \cdots + a_{k-1}$ by the inductive hypothesis. So a_k is added to list L if and only if it is greater than the sum of all previous terms of the sequence. Since, by the inductive hypothesis, L already contained all elements from $a_1, a_2, \dots, a_{k-1}$ that are greater than the sum of all previous terms of the sequence and we have added a_k exactly when it also satisfies this property, we know that now after the kth iteration, L contains all elements from $a_1, a_2, \dots, a_k$ that are greater than the sum of all previous terms of the sequence. Similarly, on the kth iteration of the loop, we add a_k to the current *total* in line 6. Since we know from the inductive hypothesis that $total = a_1 + a_2 + \cdots + a_{k-1}$ and we just added a_k, then after the kth iteration, we have $total = a_1 + a_2 + \cdots + a_k$. This proves the inductive step, and so we conclude that the loop invariant is true for all $k \geq 0$ as desired.

(b) (4 points) Use the loop invariant to prove that the algorithm is correct, i.e., that it returns a list of all terms in the sequence that are greater than the sum of all previous terms of the sequence.

Solution: The for loop in this algorithm runs n times, as seen in line 3. We have already proved that the loop invariant is true for all values of $k \geq 0$, in particular when $k = n$. This says that after the nth iteration of the **for** loop (that is, when the algorithm is finished), $total = a_1 + a_2 + \cdots + a_n$ and L contains all elements from $a_1, a_2, \dots, a_n$ that are greater than the sum of all previous terms of the sequence. The algorithm returns the value of L, which is a list containing all elements from $a_1, a_2, \dots, a_n$ that are greater than the sum of all previous terms of the sequence, which is exactly what this algorithm was supposed to return according to the problem specification.