CS 367 Homework 5

Alex Pizzuto

---

1 Question One

We are given the following shaker sort (bi-directional bubble sort) algorithm:

```
begin = 0, end = A.length-1

// At the beginning of every iteration of this loop, we know that the
// elements in A are in their final sorted positions from A[0] to A[begin-1]
// and from A[end+1] to the end of A. That means that A[begin] to A[end] are
// still to be sorted.
do
    for i going from begin to end-1
        if A[i] and A[i+1] are out of order, swap them
    end--

    if no swaps occurred during the preceding for loop, the sort is done

    for i going from end to begin+1
        if A[i] and A[i-1] are out of order, swap them
    begin++
 until no swaps have occurred or begin >= end
```

1.1 A: Best Case Time Complexity

The best case time complexity is just $O(N)$ where N is the size of the array. This occurs when the values were already in sorted order. In this case, no swaps occur, and the array is only iterated over once, with $O(1)$ comparisons being made N times, resulting in $O(N)$. The sort will then stop as no swaps occurred.

1.2 B: Worst Case Time Complexity

The worst case time complexity for this sort is $O(N^2)$. This occurs when the array is in reverse order. In this case, the maximum number of swaps occurs during each pass forwards and backwards through the array. This means that during the first (forward) iteration, there would be N swaps. Then, backwards, there would be $N-1$ swaps (as now one element is in place). This will keep happening, and the overall complexity is given by

$$\sum_{i=0}^{N} i = 1 + 2 + \cdots + N$$

which we know is bounded by $O(N^2)$.

2 Question Two

2.1 **A** Now, suppose instead of needing to sort an entire array, we need only extract the smallest (or largest) K items from the list. If we can only choose between insertion sort and selection sort, then selection sort is easier to modify so that it *efficiently* gives us only the smallest K items in sorted order.

2.2 **B** In order to make this change, we need to change the algorithm so that the outer loop, instead of iterating from index 0 to index `length - 1`, we only go from index 0 to index `K - 1`.

2.3 **C** Insertion sort cannot be modified to what we want to do efficiently. We will prove this via counterexample. Take the case where we want the smallest 5 values in the array `A`. Also suppose that one of these values lies in `A[length - 1]`. For insertion sort, the entire algorithm needs to operate, and we need to sort all elements in smaller indices before finding this value, so the worst case complexity is the worst case complexity of a full insertion sort, while the worst case for selection sort with modification is $O(N * K)$, where N is the length of the array and K is the number of extremes that we desire.

3 Question Three

Using median-of-three pivot selection, we show the successive changes that are made as quicksort partitions the following list of integers:

80	90	50	10	80	70	30	40	70	50	40	20	60

Use median of three pivot selection, move these three items

30	90	50	10	80	70	20	40	70	50	40	60	80

Switch 90 and 40

30	40	50	10	80	70	20	40	70	50	90	60	80

Switch 80 and 50

30	40	50	10	50	70	20	40	70	80	90	60	80

Switch 70 and 40

30	40	50	10	50	40	20	70	70	80	90	60	80

Switch to return pivot

30	40	50	10	50	40	20	60	70	80	90	70	80