Steven Rosendahl
Homework 1

Derive the order of magnitude worst-case running time of the following algorithm, and give a clear explanation of how you arrived at this bound, in the same manner as the CLRS insertion-sort example from chapter 2 which we did this in class. As I did in class, you can leave out the constants c_i for each line of code; but, where appropriate, you must analyze loops by setting up a sum over the loop iterations. Here swap(A, i, j) means interchange array elements $A[i]$, $A[j]$. Note that swap is not one of our basic constant-time operations; how will you deal with this?

```
i = 1
j = N
while i < j
    for k = i to j - 1
        if A[k] > A[k + 1]
            swap(A, k, k + 1)
    for k = j - 1 to i + 1 by -1
        if A[k - 1]>A[k]
            swap(A, k - 1, k)
    i = i + 1
    j = j - 1
```

We know that the while loop will execute until $i = j$. This means that the sum will execute at most $\lceil \frac{N}{2} \rceil$. We can analyze both the inner for loops individually in order to formulate a sum that is representative of the number of instructions for that loop. The first loop executes $N - 1$ times on the first iteration, then $N - 3$ times on the second iteration, and so on. This yields the sum

$$\sum_{i=0}^{N} N - (2i + 1).$$

Similarly, the second for loop will execute almost the exact same number of times; however, only the even i terms will be taken into account. As such, we have that the second loop will execute

$$\sum_{i=0}^{N} N - 2i$$

times. Assuming the worst case scenario, we can say that the comparison in the if statement of both loops will return true every iteration. As a result, the swap function will be executed on every iteration. We don't actually know the execution time of the swap function, so we can assign it the value $\mathcal{S}_t$. Combining both of the sums we already have and multiplying them by the swap time yields

$$\mathcal{S}_t \sum_{i=0}^{N} 2N - 4i - 1.$$

The only other operations in the while loop are constant time, so we will ignore those.

We now need to express the number of times the while loop executes as a sum. We know that every time the loop executes, we increment i and decrement j. As a result, we know that the loop will only execute $\lceil \frac{N}{2} \rceil$ times. We also know that our i starts at 1. Combining the sums together gives us

$$\mathcal{S}_t \sum_{i=1}^{\lceil \frac{N}{2} \rceil} \sum_{i=0}^{N} 2N - 4i - 1.$$

Of course, we don't want to have a sum of sums. Through algebraic manipulation, we have that

$$\sum_{i=0}^{N} 2N - 4i - 1 = -(1 - n),$$

so we are now dealing with

$$\mathcal{S}_t \sum_{i=1}^{\lceil \frac{N}{2} \rceil} -(1-N).$$

We have two cases to deal with now: when N is even and when N is odd. If N is even, then we are dealing with

$$\mathcal{S}_t \sum_{i=1}^{\frac{N}{2}} -(1-N).$$

This can be simplified to

$$\frac{1}{2}(N-1)N = \frac{N^2-N}{2}$$

which is $O(N^2)$. If N is odd, then we have

$$\mathcal{S}_t \sum_{i=1}^{\frac{N+1}{2}} -(1-N),$$

which is equivalent to

$$\frac{N^2-1}{2},$$

and that is still $O(N^2)$.