CSC 654
Homework #1

A1

The algorithm in discussion forms a type of sorting to the input array called *Selection Sort*, the algorithm takes an input array of numbers, for example $A = (a_1, a_2, a_3, \dots, a_n)$, and returns a permutation of the input array $A = (a'_1, a'_2, a'_3, \dots, a'_n)$, such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

A2

The algorithm in discussion in an iterative form would go like this:

```
1. for k = 1 to A.length do
2.     x = ∞
3.     i = 0
4.     for j = k to A.length do
5.         if A[j] < x then
6.             x = A[j]
7.             i = j
8.     A[i] = A[k]
9.     A[k] = x
```

A3

Let n be the number of elements in A, t_j is the number for loops for j.

Line #	Cost	Times	Best	Worst
1	C_1	$n+1$	$n+1$	$n+1$
2	C_2	n	n	n
3	C_3	n	n	n
4	C_4	$\sum_{j=k}^{n} t_j$	$\sum_{j=1}^{n} (n-j+1)$	$\sum_{j=1}^{n} (n-j+1)$
5	C_5	$\sum_{j=k}^{n} (t_j - 1)$	$\sum_{j=1}^{n} (n-j)$	$\sum_{j=1}^{n} (n-j)$

6	C_6	$\sum_{j=k}^{n}(t_j-1)$	$\sum_{j=1}^{n}1$	$\sum_{j=1}^{n}(n-j)$
7	C_7	$\sum_{j=k}^{n}(t_j-1)$	$\sum_{j=1}^{n}1$	$\sum_{j=1}^{n}(n-j)$
8	C_8	n	n	n
9	C_9	n	n	n

Best Case

$$T(n) = C_1(n+1) + C_2(n) + C_3(n) + C_4\left(\frac{n(n+1)}{2}\right) + C_5\left(\frac{n^2-n}{2}\right) + C_6(n) + C_7(n) + C_8(n) + C_9(n)$$

$$T(n) = n^2\left(\frac{C_4}{2}+\frac{C_5}{2}\right) + n\left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} + C_6 + C_7 + C_8\right) + C_1$$

$$\theta(n^2)$$

Worst Case

$$T(n) = C_1(n+1) + C_2(n) + C_3(n) + C_4\left(\frac{n(n+1)}{2}\right) + C_5\left(\frac{n^2-n}{2}\right) + C_6\left(\frac{n^2-n}{2}\right) + C_7\left(\frac{n^2-n}{2}\right) + C_8(n) + C_9(n)$$

$$T(n) = n^2\left(\frac{C_4}{2}+\frac{C_5}{2}+\frac{C_6}{2}+\frac{C_7}{2}\right) + n\left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} - \frac{C_7}{2} + C_8\right) + C_1$$

$$\theta(n^2)$$

A4

Loop Invariants and the correctness of *selection sort*

1. $A[1 \dots k-1]$ consists of elements originally in $A[1 \dots k-1]$ but in a sorted order
2. Elements in output array A' are a permutation of the elements that were originally in input array A

Initialization

We begin the algorithm with $k = 1$, that means our array of interest will be $A[1]$ which is considered trivially sorted, and naturally is considered the same element.

Maintenance

For an arbitrary iteration of our algorithm, what we observe is that we loop over each element of the $A[k \dots n]$ until an element that's less than $A[k]$ is found, in that case a swap operation takes place, otherwise, we keep iterating until we reach $A[n]$, either way, the only affected positions are $A[k]$ and $A[j]$ where j is the index of the lesser element found in $A[k \dots n]$. Therefore, the array $A[1 \dots k]$ consists

of the elements originally in $A[1 \ldots k]$ but in sorted order, thus incrementing k for the next iteration preserves the loop invariant.

Termination

Algorithm terminates when $k > n$, because we increment k by 1 each iteration, we should have $k = n + 1$ by the last iteration, by substituting $n + 1$ for k in the wording of the loop invariant, we have that the subarray $A[1 \ldots n]$ consists of the elements originally in $A[1 \ldots n]$ but in sorted order, observing that the subarray $A[1 \ldots n]$ is the entire array, we conclude that the entire array is sorted, Hence, the algorithm is correct.