## Exercise 1

The worst case complexity can be seen in the following code and is O(N^2). This is also the best case since we always have to check the rest of the array to find the smallest value.

*Exercise1/sort.h – Line 14 - 33*

*template*<*typename* T>

*void* selectSort(vector<T> *&a*)

{

*int* min; *// O(1)*

T temp; *// O(1)*

for (*int* i = 0; i < a.size() - 1; i++) *// O(N)*

{

min = i; *// O(1)*

for (*int* j = i + 1; j < a.size(); j++) *// (N-1)/2 = O(N)*

{

if(a[j] < a[min]) *// O(1)*

{

min = j; *// O(1)*

}

}

temp = a[i]; *// O(1)*

a[i] = a[min]; *// O(1)*

a[min] = temp; *// O(1)*

}

}

## Exercise 2

First the counting array is updated where the amount of each number in A are counted at their index. (This has a complexity of O(N) since we must read alle the values from A)

Then the index array is updated with the sums of numbers less than x. (This also has a complexity of O(N) since we are calculating the sum of all previous values, thus only needing to go through once)

Finally, the output array is filled using the information from A and the counting array. (Here we also only go through A once and thus only using O(N))

Combined we get 3O(N), which is the same as O(N).

## Exercise 3

We know that for each element in the heap we remove we must run heapify. Since heapify has a complexity of O(logN) and we have N elements, we get O(NlogN). This means that it does not matter if A is already sorted in increasing or decreasing order.

## Exercise 4