Homework 6

Problem 6.1

a) Implemted in "Bubblesort.cpp".

b) Best case.

Best case would be when the array is already sorted therefore the loop would only take place once so complexity would be $\Theta(n)$

Worst case

worst case would be a reverse order array. It would need both loops to work hence the complexity would be $\Theta(n.(n-1))$ which $\Theta(n^2)$ complexity.

Average case

For averse case half of the element need to be swapped so it woda be $\Theta\left(\left(\frac{n.(n-1)}{2} \times \frac{1}{2}\right)\right)$ ~~for~~ which would losolt in $\Theta(n^2)$ complexity.

c) Insertion sort:

Stable sorting algorithm if elements are in order they will not be swapped.

Merge Sort.

This is quite a tricky one. when we are mersing the array at the end we must make sure we use $L <= R$ and if this is the case it is stable else unstable.

Bubble sort.

Stable sorting algorithm because if they are in order they will not be swapped.

~~Insertio~~ Heap sort.

This is an unstable algorithm as even if the elements are in order they might be swapped

d) Inseretion sort

It is adaptive as best case $\Theta(n)$ and worst case is $\Theta(n^2)$.

Merge sort

It is non-adaptive as both the best case and worst case come out to be $\Theta(n \log n)$ complexity.

Heap sort.

It is non-adaptie as both the best case and worst case have $\Theta(n \log n)$ complexity

~~Bubble sort~~
~~Insertion sort~~

It is adaptive as best case has $\Theta(n)$ complexity whereas worst case has $\Theta(n^2)$ complexity.

Problem 6.2

a) Implementation side "Heapsort.cpp"

b) Implementation inside "Heapsortvariant.cpp"

c) As you can see from the graph the variant takes in less time after a high number of N has been reached. (~~this~~ according to the data that has been shown by my plot.).