# ADS-HW4

**Solved by :**

**SHOROUK GABR AWWAD**

**30002030**

**8.3.2019**

**First QUESTION:**

**a)**

```
bubbleSort(A, n )

   // n is the length of A ; A is an array of elements
 While (1)
       swapped = false
      for i = 1 to n-1
            /* if this pair is out of order */
           if A[i] > A[i+1] then
             // swap the elements and keep the change
             swap (A[i], A[i+1] )
             swapped = true
       if (swapped == false)
             break
```

/*the algorithm checks first if the elements are ordered or not, if they are the inner for loop doesn't execute. If they are not, it sorts them by swapping the element which are not in order each iteration for while n number of times inside the inner for-loop.*/

**b)**

**worst case : the whole array in reverse order**

| | |
|---|---|
| **While (1)** | **T(n)** |
| **swapped = false** | **T(n)** |
| **for i = 1 to n-1** | **T($\sum_0^{n-1} n$) = ($n^2$-n)** |
| **if (A[j]>A[j+1])** | **T($\sum_0^{n} n$) = (n-1)$^2$** |
| **swap (A[i], A[i+1] )**<br>**swapped = true** | **T($\sum_0^{n} n$) = (n-1)$^2$** |
| **if (swapped == false)**<br>**break** | **T(n)** |

**The Bubble sort belongs to OT($n^2$) for worst cases**

**The best case time complexity belongs to O(n)**

| | |
|---|---|
| While (1) | T(1) |
| swapped = false | T(1) |
| for i = 1 to n-1 | $T(\sum_0^1 n)$ = O(n) |
| if (A[j]>A[j+1]) | $T(\sum_0^1 n)$ = O(n-1) |
| swap (A[i], A[i+1] )<br>swapped = true | $T(\sum_0^1 n)$ = O(n) |
| if (swapped == false)<br>break | T(1) |

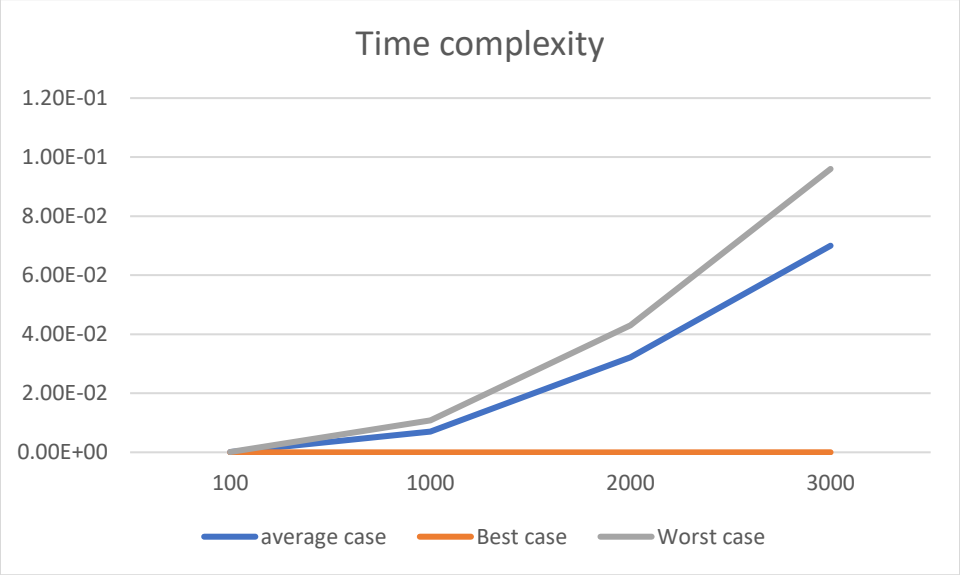
**The best-case time complexity belongs to O(n)**

**While worst case and best case both belong to $T(n^2)$, then the average case will belong to $\Theta(n^2)$**

**Proof by implementation:**

**p.s. see the attached code file**

**graph:**




**The time complexity in the graph appears for best case and worst case to be of $O(n^2)$,**

**While for the best case it's way too much faster than them so the time complexity of the best case is of O(n) which is slightly linear time complexity as shown in the graph.**

**C)**

**Stability key:**

| | | |
|---|---|---|
| **Merge sort** | **Stable** | **In merge function, the way this function compares two front elements to merge them puts the position in consideration, so if the element was in the first half of the array and the second element was in the second half. The element in the first half with the superior position comes first in the array.** |
| **Bubble sort** | **Stable** | **While bubble sort iterates from left to right through the array and swaps the consequent elements, if two consequent elements had the same value the priority of coming first goes to the element to the first position.** |
| **Insertion sort** | **Stable** | **the iteration of this algorithm states that if an element with a small value comes first before an equal element, the first element will come first in the sorted array before the equal element. Which means, the algorithm considers the priority of position of elements when it iterates.** |
| **Heap sort** | **Unstable** | **In the mechanism of heap sorting if we have two equal elements in the leaves level; when the heapify function is called, it picks the first element with the first position and move it to the parent node. And when the build_heap function is called, it tries to preserves the order of the elements of** |

| | | |
|---|---|---|
| | | **the heap as possible. But when the heap sort is called, the elements from bottom up are arranged in an ascending order, which means the element with first position comes after an equal element with a later position.** |

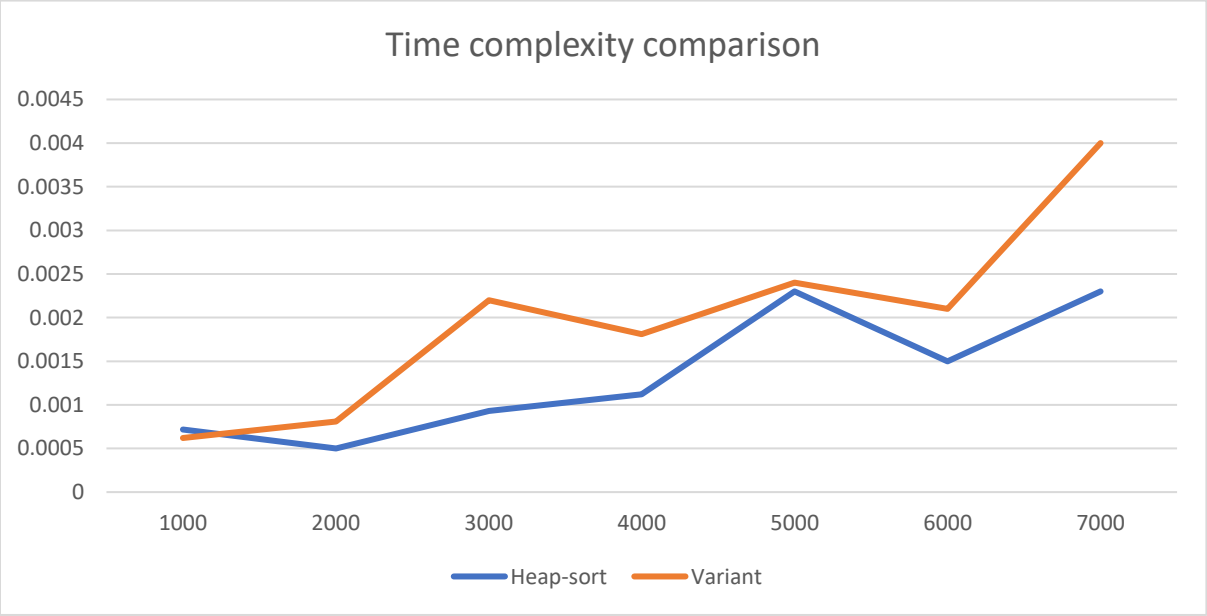
**d) Adaptiveness key :**

| | | |
|---|---|---|
| **Insertion sort algorithm** | **Adaptive** | **In worst case the time complexity of sort algorithm is of $O(n^2)$, but for the best case it is of O(n). which means the time cost varies according to the entries case.** |

| | | |
|---|---|---|
| **Heap sort** | **Non- adaptive** | **The time complexity for heap sort in worst case is of O(nlgn) and in best case Ω(nlgn) ; the worst case for the heap sort is when the array is already sorted. In this case, the time taken to build the maximum hep is the maximum.** |
| **Bubble sort** | **Adaptive** | **The time complexity for both worst and best cases doesn't depend on the nature of the time complexity O(n), which means the algorithm doesn't depend on the nature of the entries.** |
| **Merge sort** | **Non – adaptive** | **Merge sort is always of time complexity O(nlgn), whatever the case of the entries.** |

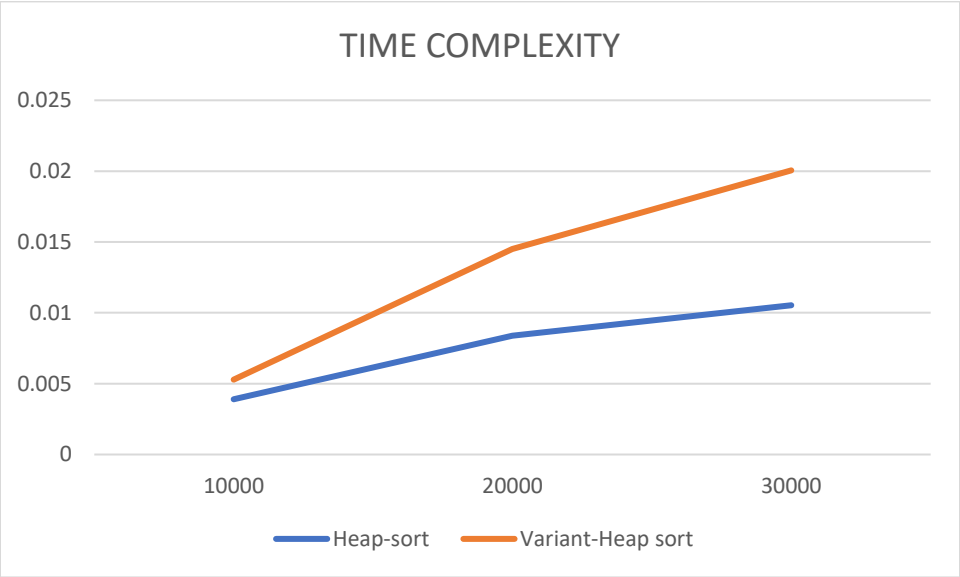
## Second question:

**c)**



*In this graph, the number of elements in a randomly generated array grows from 1000 to 7000.

From the graph, the nature of the array affects the variant sort, but in most cases the heap variant algorithm took much more time than the heap variant algorithm for the greater number of comparisons it requires.




For larger numbers of elements the difference grows apparently, and the time complexity for the same random generated arrays is greater for the variant heap sort than it is for the ordinary heap sort.