Problem 6.1

Ans a.)

```
Bubblesort (A, N)
do :
swaps = 0 .
for. count = 2 to N
    if. A[count - 1] > A[count] :
        copy = A[count - 1].
        A[count - 1] = A[count]
        A[count] = copy
        swaps++.
while(swaps != 0)
```

A is the array to be sorted and N is the number of elements in it.

– Swapping A[count - 1] and A[count]

— Increasing the number of swaps.

Ans b.)

Worst case.

The time complexity of bubble sort depends on the number of. swaps. made within the for loop and. the. no. of iterations made. by the while loop (which depends on how long the array takes to be sorted). So, assuming the array is being sorted in ascending order, during the n^{th} iteration, the n^{th} largest element will land on its proper position and no more. swaps will be made.

	cost
Bubblesort (A, N)	
do :	n
swaps = 0	n
for count = 2 to N	$n \cdot n$
if A[count - 1] > A[count] :	$n \cdot n$
copy = A[count - 1]	$(n - count) \cdot n$
A[count - 1] = A[count]	$(n - count) \cdot n$
A[count] = copy	$(n - count) \cdot n$
swaps++	$(n - count) \cdot n$
while (swaps != 0)	n

Costs are only for the worst case.

sum $= \Theta(n^2)$

So, $T(n) = O(n^2)$

Best case.

The best case scenario occurs when the number of swaps is zero and the iterations of the while loop is 1. This occurs when the array is already sorted. In such case the for loop runs n times and that is the highest order term in the sum of costs. So

$T(n) = \Omega(n)$

Average case:

For the average case we have.

$$T(n) = \frac{\text{best-case} + \text{worst-case}}{2}$$

$$= \frac{n + n^2}{2}$$

$$= \Theta(n^2)$$

Ans c.)

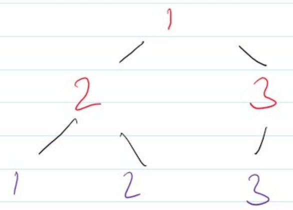
Bubble sort is stable because given two elements a and b, where a precedes b in the unsorted collection of elements. and a=b., b will be swapped. with larger. elements. until it just. succeeds. a and then the swapping will stop. So, we are left with a preceding b.

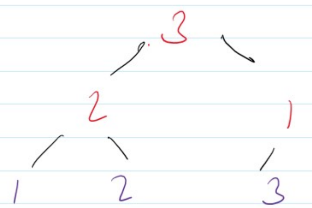
Insertion sort is stable. Given two elements a and b. where a precedes b in the unsorted collection. of. elements in an array [start . . . . . a . . . b . . . end], and a=b. when the algorithm. has. iterated. till b, and. when it has reached b, it will shift all the larger elements between a and b by 1 and place. b right after. a., thus making the. algorithm. stable.

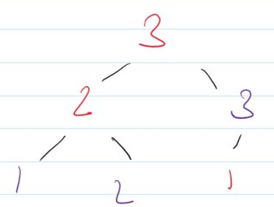
Merge-sort is also stable. Given two elements a and b where a precedes b in the unsorted collection of elements and, a = b. During a merge call, a will be on the left array and b will be on the right array. Since in the case of equality the element from the left array is placed first in the sorted, merged array and then the same element from the right array will be placed. So in the aforementioned array, a is placed first and then b is placed in the sorted array. After this the relative position of a and b is not changed during the sorting, thus making the algorithm stable.

Heap sort is not stable as the following counter-example will illustrate.

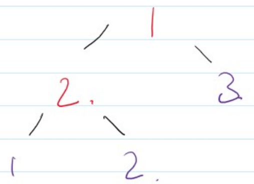
Unsorted array = 1, 2, 3, 1, 2, 3.





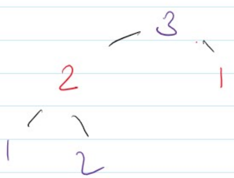


– making a max-heap.

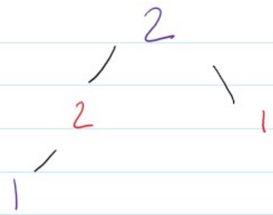


Sorted Array till now = 3

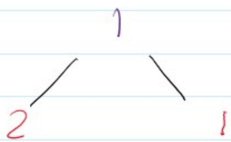
removing first element from the functional heap and putting it on the beginning of the sorted array

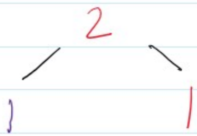


Sorted Array till now = 3 3



Sorted Array till now. = 2 3 3





Sorted Array till now = 2 2 3 3

Sorted Array till now = 1 2 2 3 3

1

Sorted Array till now = 1 1 2 2 3 3

Ans d.)

Insertion sort is adaptive algorithm. when the algorithm reaches an element x in the array, all the elements preceding x are already sorted. So only the elements which are larger than x need to be shifted by 1 to create the proper place for x. Thus, insertion sort is adaptive.

Merge sort is also adaptive as the two arrays being merged are already sorted. and this allows us to simultaneously iterate through left and the right arrays and placing them according to the desired order in the sorted order.

Heap sort is also an adaptive algorithm, as by utilizing the max-heap property, we can always obtain the largest elements on the array / sub-array as the root of the heap. Then we can exchange the root with the last element of the array, make a max heap again whilst disregarding the last element of the array and repeating the process over until the array is sorted.

Bubble sort in the other hand is not adaptive, as cannot discern whether the parts of its input sequence, is pre-sorted or not and thus cannot use such pre-sortedness to its advantage.