6.1)

a) in code

b) Best case: when array is already sorted, so it only executes inner loop ONCE, and then BREAK

statement executes.

Worst case: ∑N−1 i=1(∑N−i−1 j=1 (1))=∑N−1 i=1 (N−I) => N\*(N-1)/2 =>  
 T(N) = Θ(N^2).

Average case: also Θ(N^2), because in asymptotic perspective, it stays the same.  
T(N) = Θ(N^2)

c) 1. Insertion sort: stable ( it does not swap the equal values, only swaps when the one value is strictly larger than the other )

2. Merge sort: stable ( in the MERGE function if the elements are equal, it firstly takes the left element and places it to the left, so it does not change the order. )

3. Heap sort: unstable ( in the Heap-sort function if we have, for example tree like this:

16(1)

/ \

12(2) 12(3)

Firstly it takes 16, then we have

12(2)

/

12(3)

And it places 12(2) then to the sorted array: 12(2) 16

And then 12(3) : 12(3) 12(2) 16(1)

Thus, it is not stable, mainly because it firstly checks the left child-node and compares it with the parent-node

4. Bubble sort: stable, because there is no swaps of equal elements, because comparisons are strict between the previous element and the next one.

d) 1. Insertion sort: is adaptive, because when the array is sorted, it does not go to the inner while-loop. ( Arr[j] < Arr[i] always )

2. Merge sort: not adaptive, because it always goes all the way down to the array of 1 element in any case.

3. Heap sort: not adaptive, because at any case, it calls Max-Heapify to restore the HEAP property and the lowest element goes all the way down to the tree. ( I am not considering, when all the elements are identical, because it is an extremely rare case )

4. Bubble sort: adaptive, because if we pass the sorted array there, it will not execute the inner loop and time complexity is going to be Θ(N)

6.2)