Problem 6.1

a)

Pseudo code in bubbleSort.txt file

b)

For the worst case we would have to swap all items each time. Therefore, we would have an outer loop that repeats n times because we will have to swap with each iteration. The inner loop would repeat n – I times. Therefore, we calculate:

$$\Theta(n) \cdot \Theta(n-i) = \Theta(n) \cdot \Theta(n) = \Theta(n^2)$$

For the average case we would skip some iterations because they will already be sorted. This just mean that we calculate:

$$\Theta(n-j) \cdot \Theta(n-i-x) = \Theta(n) \cdot \Theta(n) = \Theta(n^2)$$

With j being the number of iterations that we did not have to check anymore because we didn't swap any item in the last iteration and x being the number of elements that we found were already sorted in the last iteration.

For the best case of an already sorted array we go through it once and see that we did not swap any items and therefore directly exit. Therefore, the time complexity is:

$$\Theta(1) \cdot \Theta(n) = \Theta(n)$$

c)

Insertion sort is stable since we take the first unsorted element that is unsorted and swap it with the item before it until the item in front of it is not larger or equal. Therefore, we do not swap anything if items are equal.
Merge sort is also stable since we also only swap the order of two elements in the recombining phase if the item from the right element is smaller than the element from the left. If that is not the case, we simply combine them in the same order they were before.

Heap sort is not stable since we essentially pick the first of the duplicates first but put it in the back. So if we pick two duplicates we put the first one to the very right and then pick the second one which we put to the left of it. Therefore, their order is reversed.

Bubble sort is of course stable since it only swaps to directly neighboring elements if the right one is smaller than the left one.

d)

Insertion sort is adaptive since it will not do anything to the array if all elements are already in the correct place. And it will also be faster to if the array is nearly sorted or the elements are close together already since the number of comparisons it needs to do to find the correct place to put the current element will be lower.

Merge sort is not adaptive because it will take the same amount of time in any case. It will always divide down to the smallest size and the recombining also always takes the same amount of comparisons.

Heap sort is not adaptive since the process of creating the heap and then removing nodes from the heap one by one will not change if the elements are already sorted.

Bubble sort is adaptive since it will only run through all elements once if it is already sorted. If elements are nearly sorted, the chance of not swapping some of the last elements is higher.

6.2

b) and c)

I did not get them to work so they are missing