Avik Bag
Professor Peysakhov
CS 260 – Assignment #10
28th August, 2015

Written Problems

Question 1

If we know where the start of the "random" number are in the array given to us, we can split the array up into two separate arrays, one ordered and the other unordered. Iterating through each of the values in the unordered set, we use each value from the unordered set, and then try to find the lower bound and upper bound within the ordered list by using the divide and conquer technique (binary search to find them). After the upper bound and lower bound is found, we insert this number into the ordered list. Rinse and repeat for each value from the unordered list. We could also use quicksort for this, where every element from the unordered list is a pivot.

If we do not know where the start of the "random" numbers are, we can simply use any sorting algorithm, because we cannot know for sure the extent to which it is arranged.

Question 2

Stable – Bubble Sort, Insertion Sort, Merge Sort, Radix Sort.
Unstable – Heap sort, Selection Sort, Shell Sort.

Question 3

In the worst case, insertion sort is $O(n^2)$. This is because the algorithm iterates through the entire list, and the compares one by one with the element to the left of it. This is done until the chosen key satisfies the condition. First it has to iterate through the entire list, and after that it has to compare with every element to the left of it. The worst case is when the entire list is reversed. Thus, the time complexity for this algorithm is $O(n^2)$. Even in an average case, where we assume that half the list is ordered, the last element might still have to compare with every element on the left of it. This leads to a time complexity of $O(n^2)$.

Question 4

```
def mode(input_array):
max = 0;
int val;
for x in input_array:
        int ctr = 1
        for y in input_array:
                if x == y then
                        ctr ++
        if ctr > max then
                max = ctr;
                val = x;

return val
```

The time complexity for this algorithm is **O(n^2)** because of the nested for loop.

Question 5

The reason it takes **O(n log n)** because this method iterates through every node in the tree, and uses this as a key to search through the decision tree. After finding it, it removes the node from the tree. Iterating through the tree takes **O(n)** and searching through the tree takes **O(log n)**. The entire function therefore takes **O(n log n).**

Question 6

```
def sumList(var sets: List of List) // k sets represented in list of lists
for set in sets:
        sort the set using any sorting algorithm
        for this example, selection sort will be used.
        time complexity = O(n^2), and returns sorted_list as result
        for elements in sorted_list:
                print elements + "+"
```

This function will be used to print out all the sum of number in a sorted manner using the given format. The time complexity for this given algorithm is **O(kn^2).**

Question 7

a) $n^{1.58}$
b) n^2
c) $n^3 \log n$

Question 8

a) $\log n$
b) $n^{\log n}$
c) $n \log n$
d) n^2

Question 9

The runtime for this algorithm could potentially take forever as it might not even reach all the elements of the array that needs to be sorted. Also, another point to be noted is that it swaps no matter what, which leads to the no order of any kind. Therefore, the running time in the worst case scenario is $O(\infty)$.