Ural Sarp Sipahi 28093

Problem 1

A) Since I need to sort elements first I chose Merge Sort algorithm in order to sort the numbers since it has one of the best asymptotic time complexity amongst all other comparison based sorting algorithms. The recurrence relation of Merge Sort is the following:

   $T(n)=2T(n/2) + \Theta(N) = T(n)=2T(n/2) + cn$

   We can solve this recurrence by using the master's theorem:

   A=2, B=2, C=1 -> log2 = 1 -> n^log2 = n = f(n) (CASE 2). Therefore $T(n)= \Theta(nlogn)$

   We found out that sorting the elements takes nlogn time. After sorting the numbers we have to return k smallest numbers which will take O(k) time. In total it takes O(k+nlogn) time for the whole process. Since k<=n the asymptotic time complexity for this operation is O(nlogn).

B) In order to find the k'th smallest number and partition around that number to get k smallest elements, I will use Select Algorithm. In Select algorithm:
   1-The numbers will be divided into 5 groups and the medians of each group will be found.
   This operation takes O(n) time.
   2-The median of the medians will be found recursively in order to make it the pivot.
   This operation takes O(n/5) time.
   3-Partition around the pivot will be done.
   This operation takes O(n) time.
   4-Lastly we will find how many recursive select has been done on the pivot.
   In order to find what this operation's time complexity we need to find out how many elements are smaller and how many elements are bigger than our pivot. In order to find how many elements are smaller than the pivot we divide the list into two. We will have 3n/10 elements that are smaller than our pivot. We will take the bigger array that the recursive select will be called on which has the size of 7n/10. Therefore this operation will take T(7n/10) time.

   Considering all these operations the recurrence relation of these processes are:
   T(n)+T(n/5)+T(7n/10)+O(n)
   To solve this recurrence we will use substitution method.
   Guess that T(n)=O(n).
   Prove that T(n)<=cn.
   Assume that T(k)<=ck for all k < n (Induction Hypothesis)
   T(n)= c n/5 + c 7n/10 + O(n) = c 9n/10 + O(n)

In order to bring this equation to the form of cn – residual we write it as:

= cn – (cn/10 – O(n)) <= cn

Therefore we can say that the Selection Algorithm takes O(n) time.

The last operation we need to do is to sort the k elements that we have. For that as explained in the previous question merge sort can be used therefore sorting will take O(k logk) time.

Combined this algorithm takes O(n+ klogk) time.

I would use the second method because the only time that the second algorithm wont be faster than the first algorithm is when k is equal to n which is the worse case. Second algorithm has the faster running time overall.

Problem 2

A) In radix sort we generally sort the numbers with the help of a auxiliary stable sort such as quicksort. In our case for sorting strings we could also use quicksort, however in our case the array that will be used will have the size of all uppercase letters in the alphabet (A-Z) which is 26 + 1. The reason for having the +1 in the array size is because of the fact that we are going to use a special character which has a higher priority in ASCII table than all the uppercase letters. The special character that we will use is "*". When we are dealing with numbers in radix sort, while comparing two numbers that are not the same length, we put "0" in front of the number which has the shorter length compared to the other number that we are comparing it to. In our case we will put the character "*" to the end of all the strings that are not equal length to the longest string, until their lengths are all equal to the longest string. We will keep an index which is equal to the length of the longest string - 1 at the initial step and we will decrement it as we compare all the i'th index of all strings that we have.

B) Initial step: "MERT**", "AYSU**", "SELIN*", "ERDEM*", "DILARA" i=5

Step 1: "MERT**", "AYSU**", "SELIN*", "ERDEM*", "DILARA" i=5
Step 2: "MERT**", "AYSU**", "ERDEM*", "SELIN*", "DILARA" i=4
Step 3: "DILARA", "ERDEM*", "SELIN*", "AYSU**", "MERT**" i=3
Step 4: "ERDEM*", "DILARA", "SELIN*", "MERT**", "AYSU**" i=2
Step 5: "SELIN*", "MERT**", "DILARA", "ERDEM*", "AYSU**" i=1
Step 6: "AYSU**", "DILARA", "ERDEM*", "MERT**", "SELIN*" i=0

Final List: [AYSU,DILARA,ERDEM,MERT,SELIN]

C) In the initial step of the algorithm, the algorithm finds the longest string which will take O(n) time. After this operation the algorithm needs to add special characters to the end of each string which is not as long as the longest string. This operation also takes O(n) time.

We can see from the algorithm that the number of comparisons is equal to the length of the longest string which is k. While sorting numbers with radix sort, the size of the array depends on how many numbers there are (0-9). In our case our array size will be 27 because we are using the alphabet + 1 special character.

Radix sort's running time while sorting numbers is O(mn) where m is the maximum number of digits in a number. In our case the running time is O(kn) where k is the length of the longest string.