CS301-A2

Yasemin Özkut

November 2022

1 Problem 1

1.1 a)First sort the numbers using a comparison-based sorting algorithm, and then return the k smallest numbers.

In the first part we have to sort the array. For comparison based sort, one of the best asymptotic worst-case running time algorithm belongs to merge sort. So that we can use merge sort. This algorithm recursively divides the array which has n elements into two parts (each part having n/2 elements), then compare the elements, and then merge the arrays. So, at each recursive iteration we have 2 subproblems that has the size n/2. Time complexity of merge sort is T(n) = 2T(n/2) + O(n). Two T(n/2) for 2 subproblems and O(n) for merging the array.

We can solve this time complexity with using Master theorem.

a=2, b=2, a $\geq$ 0, b $\geq$ 0, f(n) = O(n) = c·n
$n^{lg_b^a}$ = $n^{lg_2^2}$ = n (Case 2 applies)
f(n) = $\Theta(n^{lg_2^2})$ = Θ(n)
T(n) = $\Theta(n^{lg_2^2}$lgn) = Θ(n·lgn)

In the second part, we have to return the k smallest numbers. For this, we can iterate k times in the array to return all of the k smallest elements. This takes constant amount of time Θ(k). In the worst-case scenario, k will be equal to n. So, k $\leq$ n. This will make time complexity Θ(n).

Since k less than or equal to n, Θ(nlgn) always dominates Θ(k), so this makes this algorithm's time complexity Θ(n·lgn).

1.2 b) First use an order-statistics algorithm to find the k'th smallest number, then partition around that number to get the k smallest numbers, and then sort these k smallest numbers using a comparison-based sorting algorithm.

In the first part, we have to use order-statistics algorithm to find the k'th smallest number. We can use the following linear time order statistics algorithm that divides n elements into group size of 5 (from the CS301 slides):

```
SELECT(i, n)
1. Divide the n elements into groups of 5. Find the median of each 5-element
group by rote.

2. Recursively SELECT the median x of the ⌊n/5⌋ group medians to be the
pivot.

3. Partition around the pivot x. Let k = rank(x)

4. if i = k then return x
elseif i < k
    then recursively SELECT the $i^{th}$ smallest element in the lower part
    else recursively SELECT the $(i–k)^{th}$ smallest element in the upper part
```

If we divide n elements into groups of 5 in the first step, there will be $\lfloor n/5 \rfloor$ groups. This action is Θ(n). In the second step, we have to recursively recursively select the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot. So, the cost will be T(n/5) in this recursive step. The third step which is partition around the pivot x will be Θ(n). At least half the group medians are $\leq$ x and at least half the group medians are $\geq$ x, which both is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians. Therefore, at least $3 \cdot \lfloor n/10 \rfloor$ elements are $\leq$ x and at least $3 \cdot \lfloor n/10 \rfloor$ elements are $\geq$ x. For $n \geq 50, 3 \cdot \lfloor n/10 \rfloor \geq$ n/4. n-n/4 = 3n/4. So, the last step will be executed recursively on $\leq$ 3n/4 elements. So, the cost will be T(3n/4) in this recursive step.

So, if we add all these complexities up, the complexity of algorithm will become T(n) = T(n/5) + T(3n/4) + Θ(n)

T(n) $\leq$ 1/5·c·n + 3/4·c·n + Θ(n)
T(n) $\leq$ 19/20·c·n + Θ(n)
T(n) $\leq$ c·n - (1/20·c·n - Θ(n)) $\leq$ cn
cn is our desired part and (1/20·c·n - Θ(n)) is our residual part. So, we can choose a c that is large enough to handle the equality.
This makes our T(n) = O(n).

In the second part, we have to partition around the k^{th} smallest element to find the k smallest elements. This will be O(n) because we have to check all element so partition.

In the third part, we have to sort these k amount of elements. So if we use merge sort like in the a part, the complexity will be O(k·lgk).

If we consider all of the steps in our algorithm, our overall complexity will become O(n + k·lgk). There can be cases that k·lgk can dominate n and there can be cases that n can dominate k·lgk. So, complexity stays O(n + k·lgk).

I would prefer the second algorithm because for bigger k values the complexity will be close to the first algorithm's complexity n·lgn but for smaller k values, the complexity will be less than the first algorithm's complexity since it will be linear.

2 Problem 2

2.1 a) How can you modify the radix sort algorithm for integers, to sort strings? Please explain the modifications.

Since the question does not specify, for this question I will assume that the strings will only contain upper case letters because of the example that is given in the question. In our list, we can have shorter and longer strings. So, we have to modify radix sort by equalizing shorter strings' lengths' with the longest strings' length. To do this, we have to select an ascii character which has a lower value than 'A' because shorter strings. We can choose '.' character for this. So, for example if our longest string in the list has 5 characters, and one of the shorter string has 3 characters (ex: "AAA"), we have to modify it as "AAA.." to equalize the lengths to compare all strings to each other. The reason we add it at the end of the string is in radix sort, we start from the last index of each element.

Each radix sort iteration uses counting sort to compare the current indexes of the string starting from the last index of each string to the first at each iteration. So, to assign each character to the idexes of the array C, we will use 0th index to store the '.' character and then for other characters, since 'A' character's ascii value is 65, we have to subtract 64 from the character's ascii value. The C array size will be constructed like in the original counting sort. The character that has the highest ascii value will be found first and when we assign the index values to them as I explained, the array size will be (ascii value of the highest character - 64 +1) since we started our array from the index 0. For example, if we have the characters A, B, G for the counting sort, we can find that G has the highest ascii value as 71. So, our C array will have indexes starting from 0 to 71-64=7. This makes the array C to have a size of 8. Since we start from the index 0, in the last step below, we have to write B's index as

B[C[A[j]]-1]

Counting sort algorithm:

```
for i ← 1 to k
        do C[i] ← 0
for j ← 1 to n
        do C[A[ j]] ← C[A[ j]] + 1
for i ← 2 to k
        do C[i] ← C[i] + C[i–1]
for j ← n downto 1
        do B[C[A[ j]]-1] ← A[ j]
        C[A[ j]] ← C[A[ j]] – 1
```

2.2 b) Illustrate how your algorithm sorts the following list of strings [“BATURAY”, “GORKEM”, “GIRAY”, “TAHIR”, “BARIS”]. Please show every step of your algorithm.

Firstly, we have to modify our strings according to the longest string. We found that the longest string is “BATURAY” which has 7 characters. So our list will become [“BATURAY”, “GORKEM.”, “GIRAY..”, “TAHIR..”, “BARIS..”].

First step: for index 6 (last character)
Our input array is A = [Y,.,.,.,.].
Since our highest value is Y and its ascii value is 89, our Auxiliary storage array C will have indexes 0 to 89-64=25 (C array will have size 26)

1) Firstly we have to initialize each index of our array C as 0. (for i ← 1 to k do C[i] ← 0)
C = [0,0,0,...,0]
2) Then we will proceed with increasing the indexes of C (for j ← 1 to n do C[A[j]] ← C[A[j]] + 1)
C = [4, 0, , ..., 1]
3) Then we will proceed with (for i ← 2 to k do C[i] ← C[i] + C[i–1])
C’ = [4, 4, 4 ...., 5]
4) Lastly we will proceed with

```
for j ← n downto 1
        do B[C[A[ j]]-1] ← A[ j]
        C[A[ j]] ← C[A[ j]] – 1
```

we can show each step of this for loop as follows:
B = [, , ,.,] $\longrightarrow$ C’ = [3, 4, 4, ..., 5]
B = [, ,.,.,] $\longrightarrow$ C’ = [2, 4, 4, ..., 5]

B = [,.,.,.,] $\longrightarrow$ C' = [1, 4, 4, ..., 5]
B = [.,.,.,.,] $\longrightarrow$ C' = [0, 4, 4, ..., 5]
B = [.,.,.,.,Y] $\longrightarrow$ C' = [1, 4, 4, ..., 4]
So, since radix sort is stable our list becomes [“GORKEM.”, “GIRAY..”, “TAHIR..”, “BARIS..”, ”BATURAY”]

From now on the steps will be same as the one I explained in the first step.

Second step: for index 5
Input array A = [M,.,.,.,Y]
Since our highest value is M and its ascii value is 89, our Auxiliary storage array C will have indexes 0 to 77-64=13 (C array will have size 14)
C = [3,1,0,0...,1]
C' = [3,4,4,4...,5]
B = [, ,., ,] $\longrightarrow$ C' = [2,4,4,4...,5]
B = [,.,., ,] $\longrightarrow$ C' = [1, 4, 4, ..., 5]
B = [.,.,., ,] $\longrightarrow$ C' = [0, 4, 4, ..., 5]
B = [.,.,.,A,] $\longrightarrow$ C' = [0, 3, 4, ..., 5]
B = [.,.,.,A,M] $\longrightarrow$ C' = [1, 3, 4, ..., 4]
So, since radix sort is stable our list becomes [“GIRAY..”, “TAHIR..”, “BARIS..”, ”BATURAY”, “GORKEM.”]

Third step: for index 4
Input array A = [Y,R,S,R,E]
Since our highest value is Y and its ascii value is 89, our Auxiliary storage array C will have indexes 0 to 89-64=25 (C array will have size 26)

C = [0,0,...,1,0,0,...2,1...,1]
C' = [0,0,...,1,1,1,...3,4...,5]
B = [E, , , ,] $\longrightarrow$ C' = [0,0,...,0,1,1,...3,4...,5]
B = [E, ,R, ,] $\longrightarrow$ C' = [0,0,...,0,1,1,...2,4...,5]
B = [E,.,R,S,] $\longrightarrow$ C' = [0,0,...,0,1,1,...2,3...,5]
B = [E,R,R,S,] $\longrightarrow$ C' = [0,0,...,0,1,1,...1,3...,5]
B = [E,R,R,S,Y] $\longrightarrow$ C' = [0,0,...,0,1,1,...1,3...,4]
So, since radix sort is stable our list becomes [“GORKEM.”, “TAHIR..”, ”BATURAY”, “BARIS..”, “GIRAY..”]

Forth step: for index 3
Input array A = [K,I,U,I,A]
Since our highest value is M and its ascii value is 85, our Auxiliary storage array C will have indexes 0 to 85-64=21 (C array will have size 22)
C = [0,1,0,...,2,0,0,...1,0...,1]
C' = [0,1,1,...,3,3,3,...4,4...,5]
B = [A, , , ,] $\longrightarrow$ C' = [0,0,1,...,3,3,3,...4,4...,5]
B = [A, ,I, ,] $\longrightarrow$ C' = [0,1,1,...,2,3,3,...4,4...,5]
B = [A, ,I, ,U] $\longrightarrow$ C' = [0,1,1,...,2,3,3,...4,4...,4]

B = [A,I,I, ,U] $\longrightarrow$ C' = [0,1,1,...,1,3,3,...4,4...,4]
B = [A,I,I,K,U] $\longrightarrow$ C' = [0,1,1,...,1,3,3,...3,4...,4]
So, since radix sort is stable our list becomes [“GIRAY..”, “TAHIR..”, “BARIS..”, “GORKEM.”, ”BATURAY”]

Fifth step: for index 2
Input array A = [R,H,R,R,T]
Since our highest value is T and its ascii value is 84, our Auxiliary storage array C will have indexes 0 to 84-64=20 (C array will have size 21)
C = [0,0,...,1,0,0,...3,0...,1]
C' = [0,0,...,1,1,1,...4,4...,5]
B = [, , , ,T] $\longrightarrow$ C' = [0,0,...,1,1,1,...4,4...,4]
B = [, , ,R,T] $\longrightarrow$ C' = [0,0,...,1,1,1,...3,4...,4]
B = [, ,R,R,T] $\longrightarrow$ C' = [0,0,...,1,1,1,...2,4...,4]
B = [H, ,R,R,T] $\longrightarrow$ C' = [0,0,...,0,1,1,...3,4...,4]
B = [H,R,R,R,T] $\longrightarrow$ C' = [0,0,...,0,1,1,...2,4...,4]
So, since radix sort is stable our list becomes [“TAHIR..”, “GIRAY..”, “BARIS..”, “GORKEM.”, ”BATURAY”]

Sixth step: for index 1
Input array A = [A,I,A,O,A]
Since our highest value is O and its ascii value is 79, our Auxiliary storage array C will have indexes 0 to 79-64=15 (C array will have size 16)
C = [3,0,0,...,1,0,0,...,1]
C' = [3,3,3,...,4,4,4,...,5]
B = [, ,A, ,] $\longrightarrow$ C' = [2,3,3,...,4,4,4,...,5]
B = [, ,A,I,] $\longrightarrow$ C' = [2,3,3,...,3,4,4,...,5]
B = [,A,A,I,] $\longrightarrow$ C' = [1,3,3,...,3,4,4,...,5]
B = [,A,A,I,O] $\longrightarrow$ C' = [1,3,3,...,3,4,4,...,4]
B = [A,A,A,I,O] $\longrightarrow$ C' = [0,3,3,...,3,4,4,...,5]
So, since radix sort is stable our list becomes [“TAHIR..”, “BARIS..”, ”BATURAY”, “GIRAY..”, “GORKEM.”]

Seventh step: for index 0
Input array A = [T,B,B,G,G]
Since our highest value is T and its ascii value is 84, our Auxiliary storage array C will have indexes 0 to 84-64=20 (C array will have size 21)
C = [0,2,0,...,2,0,0,...,1]
C' = [0,2,2,...,4,4,4,...,5]
B = [, , ,G,] $\longrightarrow$ C' = [0,2,2,...,3,4,4,...,5]
B = [, ,G,G,] $\longrightarrow$ C' = [0,2,2,...,2,4,4,...,5]
B = [,B,G,G,] $\longrightarrow$ C' = [0,1,2,...,2,4,4,...,5]
B = [B,B,G,G,] $\longrightarrow$ C' = [0,0,2,...,2,4,4,...,5]
B = [B,B,G,G,T] $\longrightarrow$ C' = [0,0,2,...,2,4,4,...,4]
So, since radix sort is stable our list becomes [“BARIS..”, ”BATURAY”, “GIRAY..”, “GORKEM.”, “TAHIR..”]

2.3 c) Analyze the running time of the modified algorithm.

Finding the longest string in n elements takes O(n) times since we have to iterate each element. To modify each shorter elements we have to iterate the list again and add '.' character to the end of the each shorter strings. If we have m characters in the longest string, Modifying the strings takes O(n·m) amount of time. In the worst case m=n. So that our time complexity becomes O(n·n).

For radix sort, we have to use counting sort. To generate the C array for counting sort, we have to look at the character that has the maximum ascii value. This takes O(n) amount of time since we have to iterate each character in our input array A. To initialize C array with 0, we also have to iterate C array, and since it has k elements, this step takes O(k) amount of time. After that we have to iterate A array that has n elements to assign each character's count in the C array. This takes O(n) amount of time. Then we have to iterate C array to sum up consecutive 2 elements. This step takes O(k) amount of time. Lastlyi we have to iterate the A array from backwards and place the characters in the B array. This step takes O(n) amount of time. So, in total we have O(n+k) and in the worst case k=n and we get O(2n) = O(n) for counting sort. For radix sort, in the worst case, our maximum element has n characters. So we have to do counting sort for n characters. This takes O(n·n) amount of time.

So, our modified algorithm's runnning time is O(n·n).