CS301-HW2

Gülsen Görkem Köse

February 2020

1 Question 1

(a) *For part a, I could use merge sort to sort the array of n numbers first, and then return the first k elements in that sorted array. The recurrence relation for merge sort is* $T(n) = 2T(n/2) + \theta(n)$ *and this is case 2 for the Master Method since* $f(n) = \theta(n) = \theta(n^{log_2 2})$ *where* $a = b = 2$*. Then the asymptotic running time complexity for the merge sort is* $\theta(nlogn)$ *in the best-case, average-case and worst-case. The reason why I choose the merge sort is that it is asymptotically optimal sorting algorithm. After I sort the elements using merge sort in* $\theta(nlogn)$ *time, returning the first k sorted elements in the array takes* $\theta(k)$ *time. Then the total complexity becomes* $\theta(nlogn + k)$ *but since we know that* $k <= n$ *always, we can omit the lower order term k and at the end, from an array that consists of n elements, returning the first k items in sorted order takes* $\theta(nlogn)$ *time in the worst-case.*

(b) *For part b, I could use Blum, Floyd, Pratt, Rivest and Tarjan's algorithm to find the k'th smallest number in the array of n. I would apply what they suggested:*
1. Divide the n elements into group of 5. And then find the median of each 5-element group by rote.
2. Recursively select the median x of the $\lfloor n/5 \rfloor$ *group medians to be pivot.*
3. Partition around the pivot.
4. If $k = rank(x)$ *then return x*
Elseif $k < rank(x)$ *then recursively select the k'th smallest element in the lower part*
Else recursively select the $(k - rank(x))$*'th smallest element in the upper part*

Then by observation, we can see that at least $3\lfloor n/10 \rfloor$ *elements are* $<= x$*. And at least* $3\lfloor n/10 \rfloor$ *elements are* $>= x$*. We can observe that* $3\lfloor n/10 \rfloor >= n/4$ *for* $n > 50$*. Since we are focused on the worst-case, let's say the k'th smallest element is in the other part which would take is* $T(3n/4)$ *time in the worst case. For the algorithm that is described above, each step takes* $\theta(n)$*,* $T(n/5)$*,* $\theta(n)$*, and* $T(3n/4)$ *respectively. When we add them up, the final recurrence relation will look like:*

$$T(n) = T(n/5) + T(3n/4) + \theta(n)$$

We can solve this recurrence relation using the substitution method, and the claim is that this recurrence is linear, which is $\theta(n)$. What we want to prove is that $T(n) <= cn$ for some constant c. Then;

$$T(n) <= \frac{1}{5}cn + \frac{3}{4}cn + \theta(n)$$

$$= \frac{19}{20}cn + \theta(n)$$

$$= cn - (\frac{1}{20}cn - \theta(n))$$

and if c is chosen large enough to dominate the coefficient of $\theta(n)$ and also meet the initial conditions. Therefore this selection algorithm runs in linear time which is $\theta(n)$ in the worst case. After we find the k'th smallest element using this selection algorithm, we could sort the smaller part and the k'th item together, again using merge sort. Then this will take $\theta(klogk)$. The total running time becomes $\theta(n + klogk)$. Since we do not know k and n, we cannot omit any of these terms.

Conclusion: For part (a) the best asymptotic worst-case running time is $\theta(k + nlogn)$ while for part (b) it is $\theta(n + klogk)$. As I said, k could be omitted from $\theta(k + nlogn)$ and the total running time complexity becomes $\theta(nlogn)$ for part (a). But we cannot omit any term for the part (b) since we do not know the values of k and n, we cannot decide which one would be the dominating term over the other. But the point here is that whatever the dominating term in the $\theta(n+klogk)$, nlogn dominates both of them. Therefore, we can say that the best asymptotic worst-case complexity of method (b) is more efficient than method (a). Hence, I would use method (b).

2 Question 2

(a)

Radix sort uses counting sort and sorts integers digit-by-digit. To sort the strings in the alphabetical order, still we should start to sort from right-most character in the radix sort but we need to make modifications on counting sort first.
*The counting sort would be based on characters this time, not digits. So the array A would store the current characters of the strings that we are sorting at that step. For this case, since we are sorting 5 strings, the array A has 5 elements, each element is a character from a different strings. Here the important part is the fact that not all the strings have equal length. Since while sorting the strings, most-significant character is always the left-most ones, first we find the longest string among all strings to be sorted, then we fill the right sides of the strings that is shorter than the longest one with another character, for example *. For instance, here in this problem the longest string has 6 characters, but*

*"Ali" has 3, so while radix sort proceed on these strings, Ali wouldn't be Ali but "Ali***" to complete the length of Ali to 6 like the others.*
There would be an output array B, which will store the characters in the current step of sorting in the sorted order. For this example, array B would contain 5 characters.
*And there should be an auxiliary storage, the array C to keep the counts of occurences of the characters. Since we have 26 letters in English alphabet and use 1 additional character * to fill the shorter strings right sides in order to make them comparable, the array C has size 27. All values in C are 0 initially, then while the counting proceed, we first increment the count of each element when we encounter it and after we proceed all characters, change the C such that each element in the array C is the count of that particular character itself + the counts of the characters which is left-hand-side of that current character.*

After we make these modifications on the counting sort, we can use the radix sort to sort these strings character by character, starting from the right-most characters.

(b) When we add the *'s, the strings would look like as follows:

$$["VEYSEL","ALI***","SELIN*","YASIN*","ZEYNEP"]$$

For all the steps that are explained detailly below, the format is as follows:
Changes made on the array B $\longrightarrow$ How the array C looks like afterwards

Step 1: Sorting 6'th characters

$$A = [\ L\ ,\ *\ ,\ *\ ,\ *\ ,\ P\ ]$$

Then C becomes:

$$C = [\ 0\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ \ ,\ \ ,\ \ ,\ \ ,\ P\ ] \quad \longrightarrow \quad C = [\ 3\ ,\ .....,\ 4\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ \ ,\ \ ,\ *\ ,\ \ ,\ P\ ] \quad \longrightarrow \quad C = [\ 2\ ,\ .....,\ 4\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ \ ,\ *\ ,\ *\ ,\ \ ,\ P\ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 4\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ *\ ,\ *\ ,\ *\ ,\ \ ,\ P\ ] \quad \longrightarrow \quad C = [\ 0\ ,\ .....,\ 4\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ *\ ,\ *\ ,\ *\ ,\ L\ ,\ P\ ] \quad \longrightarrow \quad C = [\ 0\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....]$$

At the end of the first step, the order of the strings is as follows:

$$["ALI***","SELIN*","YASIN*","VEYSEL","ZEYNEP"]$$

Step 2: Sorting 5'th characters

$$A = [\ *\ ,\ N\ ,\ N\ ,\ E\ ,\ E\ ]$$

Then C becomes:

$$C = [\ 1\ ,\ .....,\ 3\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ \ ,\ E\ ,\ \ ,\ \ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 2\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ E\ ,\ E\ ,\ \ ,\ \ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ E\ ,\ E\ ,\ \ ,\ N\ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ \ ,\ E\ ,\ E\ ,\ N\ ,\ N\ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 3\ ,\ .....]$$

$$B = [\ *\ ,\ E\ ,\ E\ ,\ N\ ,\ N\ ] \quad \longrightarrow \quad C = [\ 0\ ,\ .....,\ 1\ ,\ .....,\ 3\ ,\ .....]$$

At the end of the second step, the order of the strings is as follows:

$$["ALI***","VEYSEL","ZEYNEP","SELIN*","YASIN*"]$$

Step 3: Sorting 4'th characters

$$A = [\ *\ ,\ S\ ,\ N\ ,\ I\ ,\ I\ ]$$

Then C becomes:

$$C = [\ 1\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ \ ,\ I\ ,\ \ ,\ \ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 2\ ,\ .....,\ 4\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ I\ ,\ I\ ,\ \ ,\ \ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 4\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ I\ ,\ I\ ,\ N\ ,\ \ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 3\ ,\ .....,\ 5\ ,\ .....]$$

$$B = [\ \ ,\ I\ ,\ I\ ,\ N\ ,\ S\ ] \quad \longrightarrow \quad C = [\ 1\ ,\ .....,\ 1\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....]$$

$$B = [\ *\ ,\ I\ ,\ I\ ,\ N\ ,\ S\ ] \quad \longrightarrow \quad C = [\ 0\ ,\ .....,\ 1\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....]$$

At the end of the third step, the order of the strings is as follows:

$$["ALI***","SELIN*","YASIN*","ZEYNEP","VEYSEL"]$$

Step 4: Sorting 3'rd characters

$$A = [\ I\ ,\ L\ ,\ S\ ,\ Y\ ,\ Y\ ]$$

Then C becomes:

$$C = [....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ ,\ ,\ ,\ ,\ Y\ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ ....,\ 4\ ,\ .....]$$

$$B = [\ ,\ ,\ ,\ Y\ ,\ Y\ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ ....,\ 3\ ,\ .....]$$

$$B = [\ ,\ ,\ S\ ,\ Y\ ,\ Y\ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ .....]$$

$$B = [\ ,\ L\ ,\ S\ ,\ Y\ ,\ Y\ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ .....]$$

$$B = [\ I\ ,\ L\ ,\ S\ ,\ Y\ ,\ Y\ ] \quad \longrightarrow \quad C = [....,\ 0\ ,\ ....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 3\ ,\ .....]$$

At the end of the fourth step, the order of the strings is as follows:

$$[\text{"}ALI***\text{"},\text{"}SELIN*\text{"},\text{"}YASIN*\text{"},\text{"}ZEYNEP\text{"},\text{"}VEYSEL\text{"}]$$

Step 5: Sorting 2'nd characters

$$A = [\ L\ ,\ E\ ,\ A\ ,\ E\ ,\ E\ ]$$

Then C becomes:

$$C = [....,\ 1\ ,\ ....,\ 4\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ ,\ ,\ ,\ E\ ,\ \ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 3\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ ,\ ,\ E\ ,\ E\ ,\ \ ] \quad \longrightarrow \quad C = [....,\ 1\ ,\ ....,\ 2\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ A\ ,\ ,\ E\ ,\ E\ ,\ \ ] \quad \longrightarrow \quad C = [....,\ 0\ ,\ ....,\ 2\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ A\ ,\ E\ ,\ E\ ,\ E\ ,\ \ ] \quad \longrightarrow \quad C = [....,\ 0\ ,\ ....,\ 1\ ,\ ....,\ 5\ ,\ .....]$$

$$B = [\ A\ ,\ E\ ,\ E\ ,\ E\ ,\ L\ ] \quad \longrightarrow \quad C = [....,\ 0\ ,\ ....,\ 1\ ,\ ....,\ 4\ ,\ .....]$$

At the end of the fifth step, the order of the strings is as follows:

$$[\text{"}YASIN*\text{"},\text{"}SELIN*\text{"},\text{"}ZEYNEP\text{"},\text{"}VEYSEL\text{"},\text{"}ALI***\text{"},]$$

Step 6: Sorting 1'st characters

$$A = [\ Y\ ,\ S\ ,\ Z\ ,\ V\ ,\ A\ ]$$

Then C becomes:

$$C = [.....,\ 1\ ,\ .....,\ 2\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....,\ 5\ ]$$

$B = [\ A\ ,\ \ ,\ \ ,\ \ ,\ \ ]\quad \longrightarrow \quad C = [.....,\ 0\ ,\ .....,\ 2\ ,\ .....,\ 3\ ,\ .....,\ 4\ ,\ .....,\ 5\ ]$

$B = [\ A\ ,\ \ ,\ V\ ,\ \ ,\ \ ]\quad \longrightarrow \quad C = [.....,\ 0\ ,\ .....,\ 2\ ,\ .....,\ 2\ ,\ .....,\ 4\ ,\ .....,\ 5\ ]$

$B = [\ A\ ,\ \ ,\ V\ ,\ \ ,\ Z\ ]\quad \longrightarrow \quad C = [.....,\ 0\ ,\ .....,\ 2\ ,\ .....,\ 2\ ,\ .....,\ 4\ ,\ .....,\ 4\ ]$

$B = [\ A\ ,\ S\ ,\ V\ ,\ \ ,\ Z\ ]\quad \longrightarrow \quad C = [.....,\ 0\ ,\ .....,\ 1\ ,\ .....,\ 2\ ,\ .....,\ 4\ ,\ .....,\ 4\ ]$

$B = [\ A\ ,\ S\ ,\ V\ ,\ Y\ ,\ Z\ ]\quad \longrightarrow \quad C = [.....,\ 0\ ,\ .....,\ 1\ ,\ .....,\ 2\ ,\ .....,\ 3\ ,\ .....,\ 4\ ]$

At the end of the sixth step, the order of the strings is as follows:

$$[\text{"}ALI***\text{"},\text{"}SELIN*\text{"},\text{"}VEYSEL\text{"},\text{"}YASIN*\text{"},\text{"}ZEYNEP\text{"}]$$

Now the radix sort is done since it has sorted the strings character-by-character and the sorted result is:

["ALI*","SELIN*","VEYSEL","YASIN*", "ZEYNEP"]**

(c) *In counting sort:*

1) Initializing the auxilary array C takes $\theta(k)$ time where k is the value of maximum element in the array to be sorted.
2) Counting the elements in array A and making respective changes in array C takes $\theta(n)$ time where n is the number of items to be sorted.
3) Changing the C such that each element in the array C is the count of that particular character itself + the counts of the characters which is located at left-hand-side of that current character takes $\theta(k)$ time.
4) Constructing the output array B by going through the array A and using array C takes $\theta(n)$ time.

Therefore, the running complexity of the counting sort algorithm is $\theta(n + k)$. And we are using the counting sort for ,let's say, d times where d is the length of the longest string. Moreover, in radix sort, we have additional costs like finding the longest string and filling the shorter ones with *'s, which will take an additional $\theta(n + n) = \theta(n)$. Hence, the running complexity of radix sort is $\theta(n + d * (k + n)) = \theta(d * (k + n))$.

Here in this problem, we are using the counting sort $d = 6$ times since the longest strings "VEYSEL" and "ZEYNEP" consist of 6 characters.
The number k, which is the value of maximum element in the array is 27 since there are 26 letters in English alphabet and we also used the * character. We could sort any number of strings using this method. To sort strings that consists of only uppercase or lowercase English letters, we will always need an array C of size 27. Since 27 is a constant value but d and n may change, so the resulting running time complexity of the algorithm is $\theta(dn)$.

For this homework, I collaborated with Deniz Cangı and got help from the lecture slides. Yet the solutions and the explanations reflect my own understanding.