CS301 - Assignment 2

Ceren Arkac - 28269

November 9, 2022

1 Problem 1 (Order statistics)

Suppose that you are given a set of n numbers. The goal is to find the k smallest numbers in this set, in sorted order. For each method below, identify relevant algorithms with the best asymptotic worst-case running time (e.g., which sorting algorithm? which order-statistics algorithm?), and analyze the running time of the overall algorithm in terms of n and k.

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

The steps that will be tracked are these: 1. Select a sorting algorithms and sort the elements in ascending order with this algorithm. 2. Then find the first k elements of the sorted set.

I will choose merge sort as my sorting algorithm because I can solve the recurrence easily with the master method.

The recurrence relation of the merge sort algorithm is $T(n) = T(n) = 2T(n/2) + \Theta(n)$, where $2T(n/2)$ comes from recursively sorting the left subrarray + the right subarray;
$\Theta(n)$ comes from merging these two subarrays.

To solve the recurrence, I will apply master theorem. The appropriate form of recurrence relation to apply the master theorem is: $T(n) = aT(n/b) + f(n)$ whereas $a \geq 1$, $b > 1$ and $f(n)$ is asymptotically positive.

Here $a = 2 \geq 1$, $b = 2 > 1$ and $f(n) = \Theta(n)$ is asymptotically positive.

$n^{\log_b a} = n$

$f(n) = n = \Theta(n^{\log_b a})$ Then case 2 applies:
$T(n) = \Theta(n^{\log_b a} logn) = \Theta(nlogn)$

The cost for the first step is $\Theta(nlogn)$.

The cost for finding the first k elements of a sorted array will be related only to k since the array is already sorted and there will be no operation rather than reaching the elements from the array. Reaching an element from an array will take constant time. Therefore, unit cost will be $\Theta(1)$. The same operation will be repeated k times to reach the first k elements of the array. Then the cost for the step 2 will be $k\Theta(1)$, which is $\Theta(k)$.

The total cost for the algorithm will be $\Theta(nlogn) + \Theta(k)$

$\Theta(nlogn) + \Theta(k) = \Theta(nlogn + k)$

Since $k \leq n$ always and in the worst case $k = n$, $k = \Theta(n)$, the time complexity for the worst scenario will be $\Theta(nlogn+n)$ Because n is dominated by $nlogn$, $\Theta(nlogn+n)$ can be reduced to $\Theta(nlogn)$

Answer: $\Theta(nlogn)$

1.2 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.

The time complexity is $T(n)$ and the steps that will be tracked are these:

1. Apply an order-statistics algorithm to find the kth smallest number and partition around the kth smallest number: Worst case linear time order statistics algorithm

2. Sort the k smallest numbers by using a sorting algorithm: merge sort algorithm: $T(n) = O(klogk)$

3. Then find the first k elements of the array.

The time complexity of the problem will be the sum of the time complexities of the all steps.

The explanation for the cost of the first step:

This order-statistics algorithm is based on dividing the elements into groups (let's say groups of m). Because I want a linear time complexity in the worst case, I have to choose a m greater than or equal to 5. Because, m = 1,2,3,4 cause $\Theta(nlogn)$ time complexity.

In this homework, I will choose m as 5.
The Worst case linear time order statistics algorithm is based on Selecting the ith order element in an array in size of n. Select(i,n) procedure is:

1.Divide the n elements into groups of m = 5. Sort those groups. Find the median of each sorted groups. Store these medians.
2.Recursively call Select to find the median of medians. The median of the medians will be the pivot.

3.Partition around the pivot. Thus, return the right position of the pivot (pos) from the partition function which is in the Select function.

4.If pos = k, then return kth element of the array. Else if pos is greater than k, call Select recursively for the left subarray. Else if pos is smaller than k, call Select recursively for the right subarray.

The time complexity for the Select algorithm (which is the first step I mentioned at the top of my answer for this question.): The time complexity for the first step of Select: $\Theta(n)$
For the second step: $T(n/5)$, because the Select is called recursively for the groups of size $\lfloor n/5 \rfloor$.
For the third step: $\Theta(n)$ since all the elements of the array should be checked and placed in the right of left of the pivot according to its relation with the pivot.

For the fourth step: $\Theta(3n/4)$.

Here is the explanation: Since we sorted the groups, found their medians, and found the median of the group medians,say x, the half of the group medians are $\leq x$, that is $\lfloor n/10 \rfloor$ group medians. And at least half of the groups are $\geq x$. Then $3\lfloor n/10 \rfloor$ elements are smaller than x, and $3\lfloor n/10 \rfloor$ elements are greater than x. But here we should discount the the group containing x and the last group which has less than 5 elements.

3 ($\lfloor n/10 \rfloor - 2$ Because 3 $\lfloor n/10 \rfloor - 2 \geq 3n/10 - 6$,

Then the number of elements than x is at least $3n/10 - 6$,

The number of elements that are greater than x is at least: $3n/10 - 6$.

Since as we see in the lecture, I will some modifications: 3 $\lfloor n/10 \rfloor \geq n/4$, I will swap 3 $\lfloor n/10$ with $n/4$. Ignore 6 because it is a constant.

In the worst case, pos is not equal to k at the first time and we need to call Select recursively in either the left subarray or the right subarray.

The number of recursive calls in the worst case is at most n - $3n/10 - 6 = 7n/10 + 6$. With the modifications: $n - n/4 = 3n/4$ Then the time complexity for this step is: $T(3n/4)$

Then the time complexity for the order statistic algorithms is: $T\Theta(n) + T(n/5) + \Theta(n) + T(3n/4)$
$T(n) = T(n/5) + T(3n/4) + \Theta(n)$
I will solve the recurrence by substitution method:
$T(n) = T(n/5) + T(3n/4) + an$
Claim that $T(n) = O(n)$. Then use induction to prove it.
if $T(n) = O(n)$, then for some $c, n0 >= 0$ such that for all $n >= n0 : T(n) <= cn$
Here is proof: Inuduction base step: $T(1) <= c.1 <= c$ There are some c's to satisfy.
Induction proof step: Assume for all $k < n$, $T(k) <= ck$ Show that $T(n) < cn$ $T(n) = T(n/5) + T(3n/4)+an$ $T(n) <= c(n/5)+c(3n/4)+an$ $T(n) <= (19cn)/20+an$ $T(n) <= cn-($ something$>= 0)$ should hold to prove the claim.
$T(n) <= cn - (cn/20 - an) <= cn$ should be true some c and n.
c value can be found with below steps: $(cn/20 - an) >= 0$ $cn/20 >= an$ $c >= (an20)/n$ $c >= 20a$
for n value: if $c = 20a$, to make the residual 0, n becomes $1(n = 1)$, which is a nonnegative value. There are some c and n values to make true the inequality.
$T(n) <= cn$ for $c, n_0 >= 0$ and for all $n >= n_0$ which means $T(n) = O(n)$.

The running time complexity of the algorithm is also $\Omega(n)$ by substitution method because there are also some c and n values to satisfy $T(n) >= cn+$ something nonnegative

Then T(n) = $\Theta(n)$

Now, I will sum all the time complexities to find the time complexity of the whole problem:

As I explained in the first question, sorting k element and returning them will take $\Theta(klogk)$.
$T(n) = \Theta(n) + \Theta(klogk) = \Theta(n + klogk)$

2 Problem 2 (Linear-time sorting)

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

The radix sort uses counting sort to sort by the digits in the 1's place, then 10's places if applicable, then 100's place if applicable and so on.

As we discussed in the lecture, radix sort for number uses an array of C in the size of the number of digits in the number system that the numbers in the array are involves in.

For example, if I have an array something like this:
$my_list = [45, 1213, 1, 1011, 789]$
I say that the numbers are integers. Those integers are in decimal number system whose digits changes from 0 to 9. Then the size of C array should be 10.
In our case:
The array C should be in the size of 26 + 1 = 27:
26 is the number of letters in English alphabet.
One additional character is needed. The reason for this is: Let's say we will sort my_list. Here, we first look at the LSD's. For example, [5,3,1,1,9]. Then [4,1,0,1,8]. The third element of my_list, contains 1 digit only and to continue the algorithm, for the next steps, we consider this element as if 0001. Because the element with the highest number of digits contains 4 digits, all elements should be considered as four digit integers by adding 0's in front of the actual digits of that that integer.

For string, similarly, I should use a character like 0 in integers case. I will choose this character as *. However, to sort strings, there should be a modification. * should be come after the actual characters of the strings. For example, instead of ***Ali, it should be Ali***.

In array C, the 0th index (first element) is *, and the rest is letters.

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

Find the longest string: BATURAY (7 letters)
Modify the array with *'s: ["BATURAY", "GORKEM*", "GIRAY**", "TAHIR**", "BARIS**"]
Create an array A in the size of number of input strings, which is 5. A will store the letters we are comparing for that specific step and will change at every step. Create an array B to store the elements of A after sorting them based on counting sort. Sort the kth element where k starts from 7 and down to 1.

Sorting 7th elements:
A becomes [Y,*,*,*,*] Then C becomes [4,0,0, ... , 0,0,1,0] Then C becomes [4,4,4, ... , 4,4,5,5]

Find the correct position in B of the compared elements. Make B[C'[A[j]]] = A[j] where j starts from 5 (the number of input elements) and downs to 1.

Loop 1: B[C[A[5]]] = A[5]
5th element of A is *.
The number for * in C is 4.
Make the 4rd element of B *.
B becomes [null,null,null,*,null]
C[A[5]] = C[A[5]] - 1
Then C = [3,4,4, ... , 4,4,5,5]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is *.
The number for * in C is 3.
Make the 3rd element of B *.
B becomes [null,null,*,*,null]
C[A[4]] = C[A[4]] - 1
Then C = [2,4,4, ... , 4,4,5,5]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is *.
The number for * in C is 2.
Make the 2nd element of B *.
B becomes [null,*,*,*,null]
C[A[3]] = C[A[3]] - 1
Then C = [1,4,4, ... , 4,4,5,5]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is *.
The number for * in C is 1.
Make the 1st element of B *.
B becomes [*,*,*,*,null]
C[A[2]] = C[A[2]] - 1
Then C = [0,4,4, ... , 4,4,5,5]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is Y.
The number for Y in C is 5.
Make the 5th element of B Y.
B becomes [*,*,*,*,Y]
C[A[1]] = C[A[1]] - 1

Then C = [0,4,4, ... , 4,4,4,5]
The result for sorting 7th elements:
Array becomes [$\backslash GORKEM*$", $\backslash GIRAY**$", $\backslash TAHIR**$", $\backslash BARIS**$", $\backslash BATURAY$"]

Sorting 6th elements:
A becomes [M,*,*,*,A] Then C becomes [3,1,0, ... , 0,1,0, ... ,0] Then C becomes [3,4,4, ... , 4,5,5, ... ,5]
Loop 1: B[C[A[5]]] = A[5]
5th element of A is A.
The number for A in C is 4.
Make the 4rd element of B A.
B becomes [null,null,null,A,null]
C[A[5]] = C[A[5]] - 1
Then C = [3,3,4, ... , 4,5,5, ... ,5]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is *.
The number for * in C is 3.
Make the 3rd element of B *.
B becomes [null,null,*,A,null]
C[A[4]] = C[A[4]] - 1
Then C = [2,3,4, ... , 4,5,5, ... ,5]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is *.
The number for * in C is 2.
Make the 2nd element of B *.
B becomes [null,*,*,A,null]
C[A[3]] = C[A[3]] - 1
Then C = [1,3,4, ... , 4,5,5, ... ,5]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is *.
The number for * in C is 1.
Make the 1st element of B *.
B becomes [*,*,*,A,null]
C[A[2]] = C[A[2]] - 1
Then C = [0,3,4, ... , 4,5,5, ... ,5]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is M.
The number for M in C is 5.
Make the 5th element of B M.
B becomes [*,*,*,A,M]
C[A[1]] = C[A[1]] - 1
Then C = [0,3,4, ... , 4,4,5, ... ,5]
The result for sorting 6th elements:
Array becomes [$\backslash GIRAY**$", $\backslash TAHIR**$", $\backslash BARIS**$", $\backslash BATURAY$", $\backslash GORKEM*$"]

Sorting 5th elements:
A becomes [Y,R,S,R,E] Then C becomes [0, ... ,0,1,0, ... ,0,0,2,0, ... 0,1,0 ... ,1,0] Then C becomes [0,0, ... ,0,1,1, ... ,1,1,3,3, ... 3,4,4 ... ,5,5]
Loop 1: B[C[A[5]]] = A[5]

5th element of A is E.
The number for E in C is 1.
Make the 1st element of B E.
B becomes [E,null,null,null,null]
C[A[5]] = C[A[5]] - 1
Then C = [0,0, ... ,0,0,1, ... ,1,1,3,3, ... 3,4,4 ... ,5,5]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is R.
The number for R in C is 3.
Make the 3rd element of B R.
B becomes [E,null,R,null,null]
C[A[4]] = C[A[4]] - 1
Then C = [0,0, ... ,0,0,1, ... ,1,1,2,3, ... 3,4,4 ... ,5,5]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is S.
The number for S in C is 4.
Make the 4th element of B S.
B becomes [E,null,R,S,null]
C[A[3]] = C[A[3]] - 1
Then C = [0,0, ... ,0,0,1, ... ,1,1,2,3, ... 3,3,4 ... ,5,5]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is R.
The number for R in C is 2.
Make the 2nd element of B R.
B becomes [E,R,R,S,null]
C[A[2]] = C[A[2]] - 1
Then C = [0,0, ... ,0,0,1, ... ,1,1,1,3, ... 3,3,4 ... ,5,5]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is Y.
The number for Y in C is 5.
Make the 5th element of B Y.
B becomes [E,R,R,S,Y]
C[A[1]] = C[A[1]] - 1
Then C = [0,0, ... ,0,0,1, ... ,1,1,1,3, ... 3,3,4 ... ,4,5]
The result for sorting 5th elements:
Array becomes [$\backslash GORKEM*$", $\backslash TAHIR**$", $\backslash BATURAY$", $\backslash BARIS**$", $\backslash GIRAY**$"]

Sorting 4th elements:
A becomes [K,I,U,I,A] Then C becomes [0,1,0, ... ,2,0 ... , 1,0, ... ,1,0 ...] Then C becomes [0,1, ... ,3, ... , 4, ... ,5, ...]
Loop 1: B[C[A[5]]] = A[5]
5th element of A is A.
The number for A in C is 1.
Make the 1st element of B A.
B becomes [A,null,null,null,null]
C[A[5]] = C[A[5]] - 1
Then C = [0,0, ... ,3, ... , 4, ... ,5, ...]

Loop 2:
B[C[A[4]]] = A[4]

4th element of A is I.
The number for I in C is 3.
Make the 3rd element of B I.
B becomes [A,null,I,null,null]
C[A[4]] = C[A[4]] - 1
Then C = [0,0, ... ,2, ... , 4, ... ,5, ...]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is U.
The number for U in C is 5.
Make the 5th element of B U.
B becomes [A,null,I,null,U]
C[A[3]] = C[A[3]] - 1
Then C = [0,0, ... ,2, ... , 4, ... ,4, ...]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is I.
The number for I in C is 2.
Make the 2nd element of B I.
B becomes [A,I,I,null,U]
C[A[2]] = C[A[2]] - 1
Then C = [0,0, ... ,1, ... , 4, ... ,4, ...]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is K.
The number for K in C is 4.
Make the 4th element of B K.
B becomes [A,I,I,K,U]
C[A[1]] = C[A[1]] - 1
Then C = [0,0, ... ,1, ... , 3, ... ,4, ...]
The result for sorting 4th elements:
Array becomes ["$GIRAY**$","$TAHIR**$","$BARIS**$","$GORKEM*$","$BATURAY$"]

Sorting 3th elements:
A becomes [R,H,R,R,T] Then C becomes [0, ... ,0,1,0 ... 0,3,0 ... 0,1,0...] Then C becomes [0, ... ,0,1, ... ,4, ... ,5 ...]
Loop 1: B[C[A[5]]] = A[5]
5th element of A is T.
The number for T in C is 5.
Make the 5th element of B T.
B becomes [null,null,null,null,T]
C[A[5]] = C[A[5]] - 1
Then C = [0, ... ,1, ... ,4, ... ,4 ...]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is R.
The number for R in C is 4.
Make the 4th element of B R.
B becomes [null,null,null,R,T]
C[A[4]] = C[A[4]] - 1
Then C = [0, ... ,1, ... ,3, ... ,4,...]

Loop 3:
B[C[A[3]]] = A[3]

3rd element of A is R.
The number for R in C is 3.
Make the 3rd element of B R.
B becomes [null,null,R,R,T]
C[A[3]] = C[A[3]] - 1
Then C = [0, ... ,1, ... ,2, ... ,4, ...]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is H.
The number for H in C is 1.
Make the 1st element of B H.
B becomes [H,null,R,R,T]
C[A[2]] = C[A[2]] - 1
Then C = [0, ... ,0, ... ,2, ... ,4 ...]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is R.
The number for R in C is 2.
Make the 4th element of B K.
B becomes [H,R,R,R,T]
C[A[1]] = C[A[1]] - 1
Then C = [0, ... ,0, ... ,1, ... ,4 ...]
The result for sorting 3rd elements:
Array becomes [$\backslash TAHIR**$", $\backslash GIRAY**$", $\backslash BARIS**$", $\backslash GORKEM*$ ", $\backslash BATURAY$"]
Sorting 2nd elements:
A becomes [A,I,A,O,A] Then C becomes [0,3,0 ... 0,1,0 ... 0,1,0 ... 0] Then C becomes [0,3 ... ,4, ... ,5, ...]
Loop 1: B[C[A[5]]] = A[5]
5th element of A is A.
The number for A in C is 3.
Make the 3rd element of B A.
B becomes [null,null,A,null,null]
C[A[5]] = C[A[5]] - 1
Then C = [0,2 ... ,4, ... ,5, ...]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is O.
The number for O in C is 5.
Make the 5th element of B 0.
B becomes [null,null,A,null,O]
C[A[4]] = C[A[4]] - 1
Then C = [0,2 ... ,4, ... ,4, ...]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is A.
The number for A in C is 2.
Make the 3rd element of B A.
B becomes [null,A,A,null,O]
C[A[3]] = C[A[3]] - 1
Then C = [0,1 ... ,4, ... ,4, ...]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is I.
The number for I in C is 4.

Make the 4th element of B I.
B becomes [null,A,A,I,O]
C[A[2]] = C[A[2]] - 1
Then C = [0,1 ... ,3, ... ,4, ...]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is A.
The number for A in C is 1.
Make the 1st element of B A.
B becomes [A,A,A,I,O]
C[A[1]] = C[A[1]] - 1
Then C = [0,0 ... ,3, ... ,4, ...]
The result for sorting 2nd elements:
Array becomes [$\backslash TAHIR**$", $\backslash BARIS**$", $\backslash BATURAY$", $\backslash GIRAY**$", $\backslash GORKEM*$"]
Sorting 1st elements:
A becomes [T,B,B,G,G] Then C becomes [0,0,2,0 ... 0,2,0 ... 0,1,0 ... 0] Then C becomes [0,0,2, ... ,4, ... ,5, ...]
Loop 1: B[C[A[5]]] = A[5]
5th element of A is G.
The number for G in C is 4.
Make the 4th element of B G.
B becomes [null,null,null,G,null]
C[A[5]] = C[A[5]] - 1
Then C = [0,0,2, ... ,3, ... ,5, ...]

Loop 2:
B[C[A[4]]] = A[4]
4th element of A is G.
The number for G in C is 3.
Make the 3rd element of B G.
B becomes [null,null,G,G,null]
C[A[4]] = C[A[4]] - 1
Then C = [0,0,2, ... ,2, ... ,5, ...]

Loop 3:
B[C[A[3]]] = A[3]
3rd element of A is B.
The number for B in C is 2.
Make the 2nd element of B B.
B becomes [null,B,G,G,null]
C[A[3]] = C[A[3]] - 1
Then C = [0,0,1, ... ,2, ... ,5, ...]
Loop 4:
B[C[A[2]]] = A[2]
2nd element of A is B.
The number for B in C is 1.
Make the 1st element of B B.
B becomes [B,B,G,G,null]
C[A[2]] = C[A[2]] - 1
Then C = [0,0,0, ... ,2, ... ,5, ...]
Loop 5:
B[C[A[1]]] = A[1]
1st element of A is T.
The number for T in C is 5.
Make the 5th element of B T.
B becomes [B,B,G,G,T]

C[A[1]] = C[A[1]] - 1
Then C = [0,0,0, ... ,2, ... ,4, ...]
The result for sorting 1st elements:
Array becomes [$\backslash BARIS**$", $\backslash BATURAY$", $\backslash GIRAY**$", $\backslash GORKEM*$ ", $\backslash TAHIR**$"]

2.3 Analyze the running time of the modified algorithm.

In radix sort, we use counting sort m times, where m is the number of digits in in the longest input string. In radix sort for strings, we also take the cost for finding the longest string and filling out the shorter ones with * into consideration.

Counting sort: Creating and initializing the Array C will take O(k), where k is the number of letters in alpahet and one additional character. Filling out the array C according to the array A in each step will take $\Theta(n)$, where n is the number of strings to be sorted. Concatenating the C array will take $\Theta(k)$ Modifying the array B will take O(n).

Sum them up: $\Theta(n+k)$

Run counting sort d times: $\Theta(d*(n+k))$
Cost for finding the longest string: $\Theta(n)$
filling out the shorter ones with *: $\Theta(n)$

Sum them up: $\Theta(d*(n+k))$ since $\Theta(n)$'s are dominated.

Note: Ignore slashes in the string arrays above.