Birla Institute of Technology & Science - Pilani, Hyderabad Campus
Second Semester 2022-23
CS F211 – Data Structures & Algorithms
Comprehensive Examination

Type: Closed **Time: 180 mins** **Max Marks: 120** **Date: 10.05.2023**

All parts of the same question should be answered together.

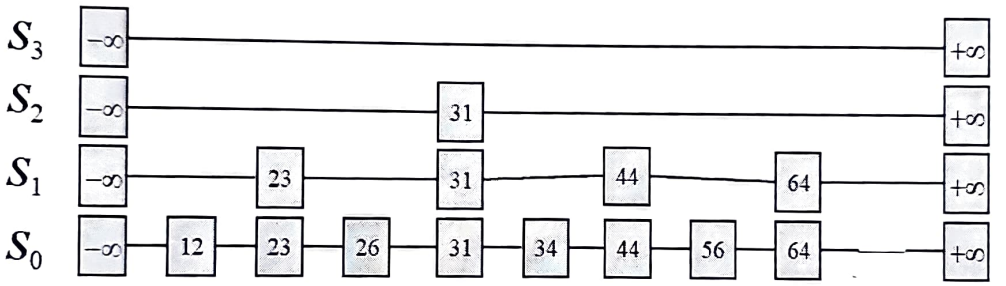
1.a. Derive the best, worst case complexity of heap sort to put up n numbers in the ascending order. [8 Marks]

Note: You need to write down all the necessary details.

Sol: Refer to class notes

1.b. Suppose there is a skip list with the elements 64, 56, 44, 34, 31, 26, 23, 12. The approach taken to build towers for each element is tossing the coin and build one tower if the toss is tail and building towers go on till first head is encountered. If the first head encountered for each of these numbers (64, 56, 44, 34, 31, 26, 23, 12) are at trial numbers: 2, 1, 2, 1, 3, 1, 2, 1 respectively then depict the skip list thus constructed. The skip list should contain all possible details. [10 Marks]

Sol: Rough sketch of the solution



2. An array T contains n elements. You want to find the m smallest elements, where m is much smaller than n and in fact $m \in o(\log n)$. Find the complexity of each of the following procedures and suggest the best of these algorithms:

(a) Sort T (using comparison-based sorting algorithms) and picks the first m,

(b) call select (T, i) for i = 1, 2, … . m.

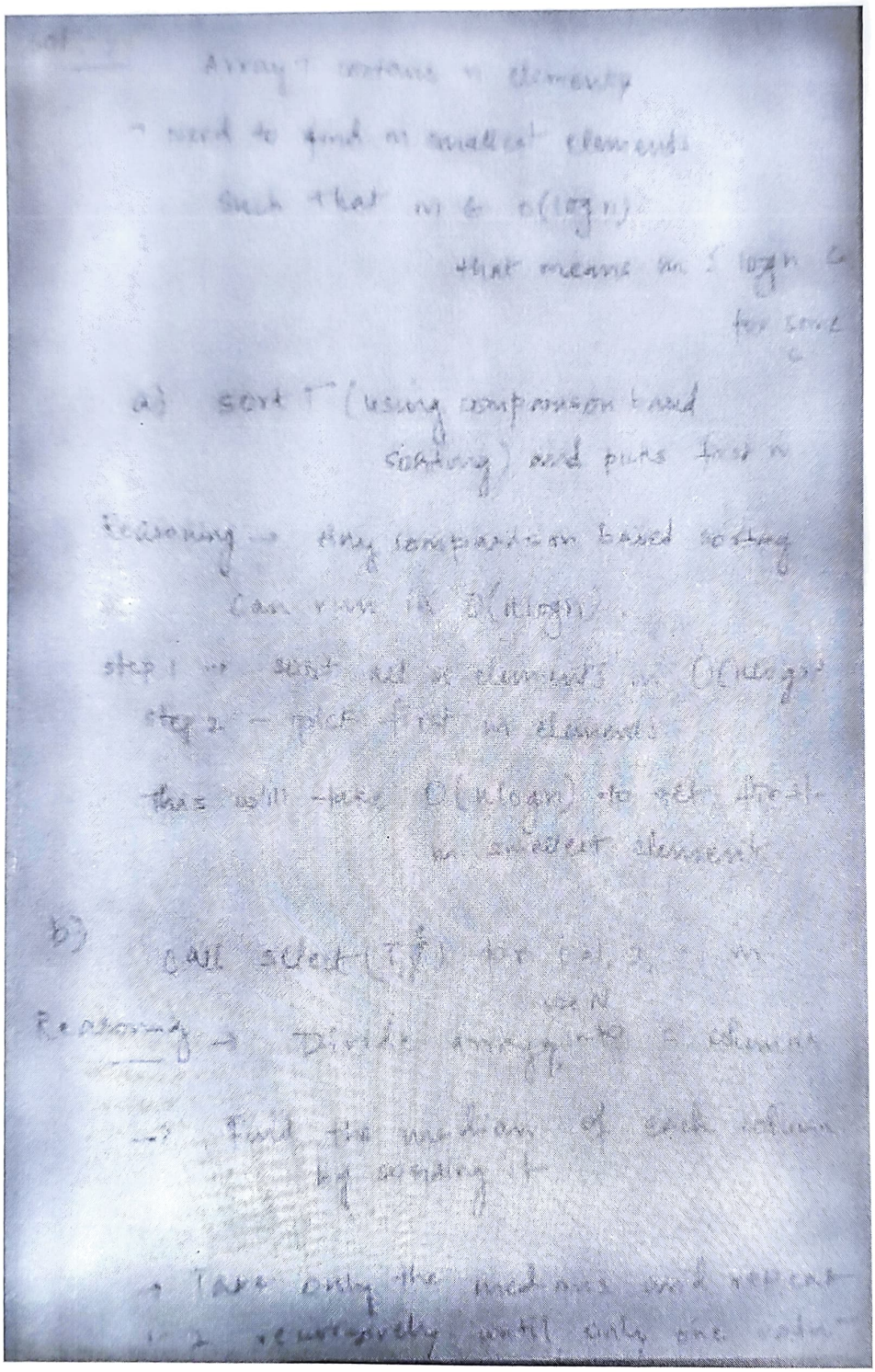
(c) Algorithm that works as follow:

```
//inputList is a list of n input elements
//outputList is a list of m elements
for i = 1 to m {
Find out the smallest element from the inputList
Add it to the outputList
Delete the smallest element from the input list}
```

Note: Select (T,i) is same as what we discussed in the class – finding i^{th} smallest number in the array T of an elements – using Median of Median method.

Notations: $m \in o(\log n)$ refers to m belongs to small o of log n. [10 Marks]

Sol:



sol :- 2

Array T contains n elements

→ Need to find m smallest elements such that $m \in O(\log n)$

that means $m \leq \log n \cdot c$ for some c

a.) sort T (using comparison based sorting) and picks first m

Reasoning → Any comparison based sorting can run in $O(n \log n)$.

step 1 → sort all n elements in $O(n \log n)$

step 2 - pick first m elements

this will take $O(n \log n)$ to get first m smallest element

b) call select (T, i, c) for $i = 1, 2, \ldots m$

Reasoning → Divide array of size N into c columns

→ find the median of each column by sorting it

→ Take only the medians and repeat 1-2 recursively until only one value

remains. That value pick as pivot

→ iterate through array and count no. of elements strictly smaller than pivot (S) larger than pivot (L) and equal to pivot (E = N - S - L)

→ If N > K, move all values smaller than pivot to the beginning and recursively run whole algo

→ If N + E > K, conclude K^{th} element is equal to the current pivot

→ Otherwise, move all values larger than pivot to beginning and recursively run the whole algorithm.

This algorithm gives K^{th} smallest in O(n) time for m smallest element it will take O(m·n) where m ≤ log n · c that means this will run faster than a

1) Finding smallest element from an array takes O(n) time. Delete the smallest and again finding smallest will take (n-1) ie again O(n)

In total it takes same but is O(m·n) but works

remains that value given as pivot

→ Iterate through array and count no. of elements strictly smaller than pivot (S), larger than pivot (L) and equal to pivot ($E = N - S - L$)

→ If $N > k$, move all values smaller than pivot to the beginning and recursively run whole algo

→ If $N + E > k$, conclude k^{th} element is equal to the current pivot

→ Otherwise move all values larger than pivot to beginning and recursively run the whole algorithm

This algorithm gives k^{th} smallest in $O(n)$ time for m smallest element it will take $O(m \cdot n)$ where $m \leq \log n$. [illegible] that means this will run faster than [illegible]

(2) Finding smallest element from an array takes $O(n)$ time. Take the [illegible] and again finding smallest will take $(n-1)$ [illegible]

In [illegible] it [illegible] same only $O(mn)$ but [illegible] better in strict comparison [illegible]

3. Derive the best, average and worst-case complexity of successful and unsuccessful search in a hash table with m slots having n elements. [10 Marks]

Sol: Refer to class notes.

4.a. Let C(n,r) or equivalently nCr represent selecting r objects from n objects, for example C(4,2) is 6. But computationally it becomes extremely difficult to find C(2000, 800) due to the number of multiplications involved. For given N, using dynamic programming, devise a $O(N^2)$ algorithm to find out C(n,r) for any n ($0 \le n \le N$), r ($0 \le r \le N$) and $r \le n$. Your algorithm should not involve multiplication of r consecutive integers starting from 1. [12 Marks]

Hint: You may think of using combinatorial identities like C(n,r) = C(n-1, r) + C(n-1, r-1).

Sol: Based on the given hint, i.e., C(n,r) = C(n-1,r)+C(n-1,r-1), store the elements in matrix, and perform computations. The complexity analysis is as follows:

****Store C(n,1) = n*****

```
for (int n=1; n<=N; n++)
        C(n,1) = n;
```

Complexity: O(n)

****Store C(n,r) = 1 when r=n*****

```
for (int n=1; n<=N; n++)
        C(n,n) = 1;
```

Complexity: O(n)

****Store C(n,r) = 0 when r>n*****

```
for (int n=1; n<=N; n++)
        for (int r=n+1; r<=N; r++)
                C(n,r) = 0;
```

Complexity: $O(n^2)$

****Compute C(n,r) = C(n-1,r)+C(n-1,r-1)*****

```
for (int n=2; n<=N; n++)
        for (int r=2; r<=n-1; r++)
                C(n,r) = C(n-1,r)+C(n-1,r-1);
```

Complexity: $O(n^2)$

Overall complexity = $O(n) + O(n) + O(n^2) + O(n^2) = O(n^2)$

4.b. Let T[1. . n] be a sorted array of distinct integers, some of which may be negative. Give an algorithm that can find an index i such that $1 \le i \le n$ and T[i] = i, provided such an index exists. Your algorithm should take a time in O(log n) in the worst case. [8 Marks]

Sol: This problem can be solved using binary search.

Check the middle element of the array. If T[j] = j, the solution exists where j is the index of the middle element.

- If T[j] < j, continue the search on the left part of the array (from 1 to j-1)
- If T[j] > j, continue the search on the right part of the array (from j+1 to n)

Given a binary search problem, you need to search O(log n) times, and hence the overall complexity is O(log n) in worst case.

5.a. Bring out scenario(s) where an array implementation is preferred to a linked list. Also discuss few advantages of using linked list implementation over an array implementation. [6 Marks]

Sol: The advantages of array over linked list are:

- Arrays provide direct or random access to the elements via the indices of the elements.
- Arrays allow only to store a single type of element like int, float, char etc.
- Arrays take lesser amount of space since pointers are not required to be stored.
- Sorting, performing binary search on sorted elements, swapping of two elements are easier in arrays.

The advantages of linked list over array are:

- Linked lists can grow or shrink dynamically in size as and when required.
- Linked lists do not require contiguous blocks of memory. Hence, creating a large number of nodes is not a problem as long as memory is available.
- Insertion/deletion of elements do not require shifting of the elements which is required in array.
- A single node of a linked list can store different types of data

Note – You are required to write one advantage of array and two advantages of linked list. 2 marks for array advantage and 4 marks for linked list advantages. If you have written that an array/linked list provides better implementation for a specific type of data structure, that point has not been considered and awarded marks.

5.b. Let $x_1, x_2, \ldots$ be a sequence of integers. Let *sum(i,j)* = $x_i + x_{i+1} + \ldots + x_j$, $i \leq j$. Using dynamic programming, devise an algorithm of $O(n^2)$ complexity to find i and j for which *sum(i,j)* is maximum. [10 Marks]

Sol:

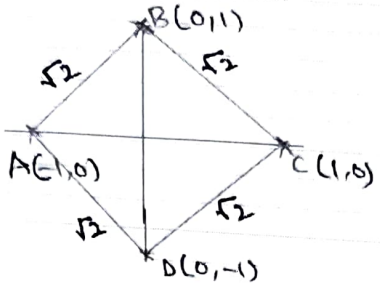
```
max = -∞
for (i = 1, i <= n, i++){
        for (j = i + 1, j <= n,  j++){
                sum(i, j) = sum(i, j + 1) + xj
                if sum(i, j) > max then
                        max = sum(i, j)
        }
}
```

Note – If you have not used dynamic programming, only 2 marks have been awarded.

6.a. Suppose the cost of laying a telephone cable from point a to point b is proportional to the Euclidean distance from a to b. A certain number of towns are to be connected at minimum cost. Find an example where it costs less to lay the cables via an exchange situated in between the towns than to use only direct links. [10 Marks]

Sol:

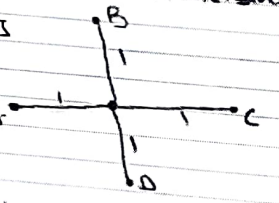
(a) consider four towns A, B,C, and D with positions A at (-1, 0), B at (0, 1) , C at (1, 0) , and D at (0, -1) as shown below.



Clearly, $|AB|=|BC|=|CD|=|DA|=\sqrt{2}$

∴ The cost connecting these towns by using the links shown in the figure is the same as the cost of MST for the graph, which is $= 3\sqrt{2}$

However, by placing an exchange at (0,0). we can connect these towns A, B, C and D with the cost 4 which is less than $3\sqrt{2}$



⇒ Hence, for this example, we have better solution by placing a new exchange.

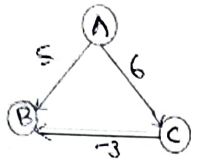
6.b. Will Dijkstra's algorithm (to find the shortest path between a source node and the remaining nodes) works for all weighted graphs. If so prove it otherwise give a counter example with all the details.

[8 Marks]

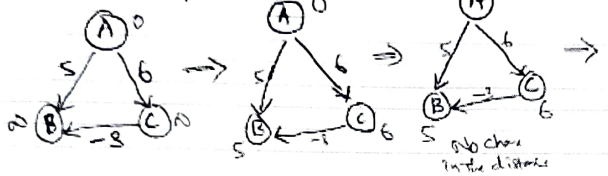
b) Dijkstra's algorithms fails to return the optimal distance between two nodes in a weighted graph with negative weights.

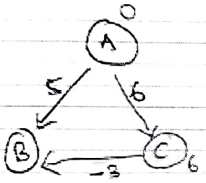
Consider a weighted graph as shown below:

Source = A
destination = B



In the final step,



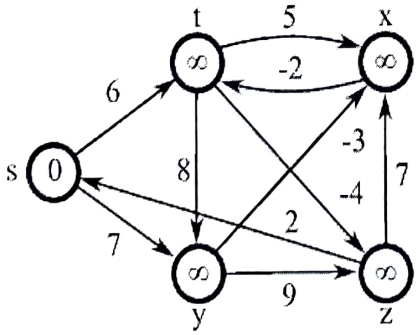


In this case, B is adjacent to C, but it's distance will not be updated as B is already explored.

Hence, algorithm stops and returns the shortest distance from A to B as 5, but the actual optimum is 3 (i.e) A → C → B with distance 6−3=3

⇒ In this case, Dijkstra algo. is failed

7. Apply Bellman-Ford algorithm to find the shortest algorithm between s and all other nodes. You need show all steps involved with all the details. [8 Marks]



Sol: Refer to class notes.

8.a. A ship arrives at a port, and the 40 sailors on board go ashore for revelry. Later at night, the 40 sailors return to the ship and, in their state of inebriation, each chooses a random cabin to sleep in. If the ship has 40 cabins and if the sailors never share a cabin what is the expected number of sailors sleeping in their own cabins? [8 Marks]

Sol: Let X_i be a random variable with value 1 when the i^{th} pirate sleeps in his own cabin and 0 otherwise. The probability of a pirate sleeping in his own cabin is 1/40.

X_i	1	0
$P[X_i]$	1/40	39/40

The expected value of X_i is

$E[X_i] = 1 \times 1/40 + 0 \times 39/40 = 1/40$

The number of sailors sleeping in their own cabin is $X = X_1 + X_2 + \ldots X_{40}$. So the expected number of sailors sleeping in their own cabin is

$E[X] = E[X = X_1 + X_2 + \ldots X_{40}]$

By linearity of expectation,

$E[X] = E[X_1] + E[X_2] + \ldots\ldots E[X_{40}] = 40 \times 1/40 = 1$

8.b. Derive average case complexity of successful sequential search in a list of 'n' elements. [12 Marks]

Sol: Let X_i be a random variable denoting number of comparisons required if the element we are searching is present in the i^{th} index of the array (assuming 1 indexing). That is $X_i=i$ if element is present and 0 otherwise. The probability of the element we are searching is present in i^{th} index is 1/n.

X_i	i	0
$P[X_i]$	1/n	1- 1/n

The expected value of X_i is

$E[X_i] = i \times 1/n + 0 \times (1-1/n) = i/n$

So the expected number of comparisons required for linear search is

$E[X] = E[X = X_1 + X_2 + \ldots X_n]$

By linearity of expectation,

$E[X] = E[X_1] + E[X_2] + \ldots\ldots E[X_n] = 1/n+2/n+3/n+\ldots\ldots 1 = (1/n)(1+2+\ldots n)$

$= n(n+1)/2n = (n+1)2.$