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

Type: Part Open **Time: 90 mins** **Max Marks: 75** **Date: 14.03.2023**

All parts of the same question should be answered together.

Part A (Closed Book)

1. Suppose $f(n) \in O(n)$ then prove or disprove the following: [2 + 2 Marks]
(i). $2^{f(n)} \in O(2^n)$

Let $f(n) \in O(n)$
We prove $2^{f(n)} \notin O(2^n)$
Suppose $2^{f(n)} \in O(2^n)$ for all $f(n) \in O(n)$.
Let $f(n) = 10n+5$
$2^{10n+5} \in O(2^n)$
$\Rightarrow \exists\ c_1 > 0$ and $n_1 \in N$ s.t.
$2^{10n+5} \le c_1 2^n \ \forall n \ge n_1$
$\Rightarrow 2^{9n+5} \le c_1 \ \forall n \ge n_1$
which is a contradiction.
Hence $2^{f(n)} \notin O(2^n)$.

(ii). $[f(n)]^2 \in O(n^2)$

$f(n) \in O(n)$
$\Rightarrow f(n) \le c_1 n \ \forall n \ge n_1$ for $c_1 > 0$ and $n_1 \in N$.
$(f(n))^2 \le c_1^2 n^2 \ \forall n \ge n_1$
$\Rightarrow (f(n))^2 \le c_2 n^2 \ \forall n \ge n$ where $c_2 = c_1^2$
$\Rightarrow (f(n))^2 \in O(n^2)$.

2. Let S = {f: f: N $\rightarrow$R$^+$}. Define a relation R on S as follows: S = {(f, g) $\in$ S X S: f $\in$ O(g)}. Prove or disprove the following statements: [2 + 2 + 2 + 2 Marks]
(i). S is reflexive.
(ii). S is symmetric

(iii). S is anti-symmetric
(iv). S is transitive
Notation: R^+ is a set of nonnegative real numbers.
Sol:

(2). $S = \{ f \mid f : N \to R^+ \}$

$T = \{ (f, g) \in S \times S \mid f \in O(g) \}$

(i) $f(n) \leq 1 \, f(n) \quad \forall \, n \geq 1$

$\Rightarrow f(n) \in O(f(n)) \Rightarrow (f(n), f(n)) \in T.$

$\Rightarrow$ T is reflexive.

(ii). Let $f(n) = n$ and $g(n) = n^2$

$f(n) \in O(g(n)) \Rightarrow (f, g) \in T$

but $g(n) \notin O(f(n)) \Rightarrow (g, f) \notin T$

Hence T is not symmetric

(iii) $f(n) = 10n + 5$

$g(n) = 17n + 100$

clearly $f(n) \neq g(n) \quad \forall \, n$

$(f, g) \in T$ since $f(n) \leq 1 \cdot g(n) \quad \forall \, n \geq 1.$

$(g, f) \in T$ since $g(n) \leq 50 \, f(n) \quad \forall \, n \geq 1$

Hence T is not anti-symmetric.

(iv).

$(f, g) \in T \Rightarrow f(n) \le c_1 g(n) \ \forall n \ge n_1$

$(g, h) \in T \Rightarrow g(n) \le c_2 h(n) \ \forall n \ge n_2$

$\forall n \ge \max\{n_1, n_2\}$

$f(n) \le c_1 g(n)$

$\le c_1 c_2 h(n)$

$\Rightarrow f(n) \le c\, h(n) \ \forall n \ge \max\{n_1, n_2\}$

where $c = c_1 c_2$

$\Rightarrow (f, h) \in T$

$\Rightarrow T$ is transitive.

3. Prove or disprove that the lower bound for comparison based and non-comparison based sorting algorithms is O(n log n)and O(n) respectively. [8 Marks]
Sol: Refer to class notes

4. Is merge sort a stable sort or not. If so, what portion of algorithm makes it stable. Otherwise what needs to be done to make it stable. [4 Marks]
Sol: Refer to class notes

5. The sequences X1, X2, . . , Xk are sorted sequences such that the sum of cardinalities of the sequences X1, X2, . . Xk is n. Show how to merge these k sequences in time O(n log k). [6 Marks]
Sol:
Let, S be the total number of elements in the array. An efficient solution will be to take pairs of arrays at each step. Then merge the pairs using the two-pointer technique of merging two sorted arrays. Thus, after merging all the pairs, the number of arrays will reduce by half. We will continue this till the number of remaining arrays doesn't become 1. Thus, the number of steps required will be of the order log(k), and since at each step, we are taking O(S) time to perform the merge operations, the total time complexity of this approach becomes O(S * log(k)).

Part B (Open Book)

1. Do you agree with the following statement: "There exists functions f, g such that f(n) $\notin$O(g(n)) and g(n) $\notin$O(f(n)) where f and g are functions from N to Set of non-negative real numbers".
Justify your answer with all mathematical reasoning or a counter example. [9 Marks]
Sol:
f(n) = 1 if n is even
= 0 if n is odd
g(n) = 0 if n is even
= 1 if n is odd

2. We have understood good number of data structures in our class. Is it possible to make use of any of these data structures (other than heap) with some changes to implement a priority queue where in the

ENQUEUE and DEQUEUE operations can be performed in O(log n) time. If so discuss the details of the data structure and the complexity. [9 Marks]

Sol:

Let us construct a Red-Black tree with priority as the key [but not the value]
The left most leaf will have the least priority and getting to the left most node ~~and~~ takes O(log n) time. Deleting the node takes O(1). Hence deque takes O(log n) time.

~~Enqu~~
~~Enqueue~~
Enque operation: Place element with priority as the key ~~[illegible]~~ as ~~[illegible]~~ as per the ADD operation in a red-black tree. This operation takes O(log n) time.

3. Input is a sequence X of n keys with many duplications such that the number of distinct keys is d (<n). Present an O(n log d) time sorting algorithm for this input. (For example, if X = 5, 6, 1, 18, 6, 4, 4, 1, 5, 17>, the number of distinct keys in X is six.) [9 Marks]

Note: You are not supposed to use non-comparison-based sorting algorithms.

Sol: There are a total number of n keys.

Let us consider d is the number of distinct keys. Therefore, n-d is the number of duplicate keys. Now, do the following:

- Insert d elements to form a BST. Time complexity O(dlog(d)) (This is by assuming that first d keys are distinct)
- There are n-d duplicate elements (on top of the existing d elements).
- For each n-d element, find a duplicate present in BST. Time complexity for search for one element is O(log d). Add a counter to the element found in BST and increase it by 1. Time complexity is O(1).
- For n-d elements, the time complexity will be O((n-d)log d).
- The summation is O(dlog(d))+ O((n-d) log d) = O(n log d).

4. Is it possible to build a binary search tree in O(n) time complexity. If so provide an algorithm and prove that the building binary search tree will take O(n) time. If not provide a proper justification. [9 Marks]

Sol: Let a_1, a_2,….a_n be n keys to be sorted. We can sort these keys as follows

1. Create a binary search tree T with these elements as keys
2. Do an inorder traversal of T

Suppose there is an algorithm to create a binary search tree in O(n) time. Then the step 1 and step 2 of the above algorithm can be done in O(n) time. Consequently, we have a comparison based sorting algorithm with running time O(n) which is contradicting Theorem 8.1 in CLRS stating lower bound for the worst case running time of any comparison based sorting algorithms.

5. Write an algorithm of complexity O(k+h) to find and delete the kth smallest in the binary search tree of height h. [9 Marks]

Sol:

→ Inorder traversal of BST will give sorted order and we can k^{th} smallest but the solution will be O(N) where N is No. of element. But for this problem we don't need to traverse all element.

To put tight bound on our solution, the algo traverse down to leftmost node in O(h) and then traverse K element. this put tight bound with overall time complexity O(h+K), where O(h) is (max height of tree)

→ If K=1, the algo go to leftmost in O(h) and 1 element will take constant time search.

→ As algo will run in similar inorder traversal fashion, it use recursion then we need to maintain a counter to get k^{th} smallest element, the recursion will implicitly use stack and stack will maximum grow to k elements. Once the counter matches K, the elements get pop out from stack to get final answer, which takes again O(h) time.

Explanation through psuedocode.

```
C = 0
getKthelement ( root, k)
{
    if root or c>=k
    return
    getKthelement (root->left, k)
    {
     if c<k
     { c++
       if c==k
       { Kthelement = root->value
         return }
       getKthelement ( root->right, k)
     }
}
```

recursive call -> to reach leftmost node O(h) = heigh of tree)

This condition bounds inorder traversal to run only k times

This condition makes algo to not run for all elements.

After match pop element from stack O(h)

Further Deletion is O(h) time in BST.

so Total its O(2h+k) = O(h+k complexity)