Comprehensive Exams January 2023

Algorithms and Data Structures

The numbers at the end of each question denote the maximum marks awarded for the correct answer of the respective question.

1. Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child. 3

2. Describe an algorithm to build a max-heap from an unordered array in linear time. 4

3. Solve the following recurrences. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answers.

   (a) $T(n) = T(n-2) + \frac{1}{\lg n}$

   (b) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

   (c) $T(n) = 7T(n/3) + n^2$

   6

4. Suppose that we toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses? 5

5. Show how to sort n integers in the range 0 to $n^3 - 1$ in $O(n)$ time. 5

6. Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue the correctness of your algorithm as well. 5

7. We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose, we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T. Prove that $G = T$. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.) 5

8. Let $G(V, E)$ be a weighted, directed graph with nonnegative weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time. 5

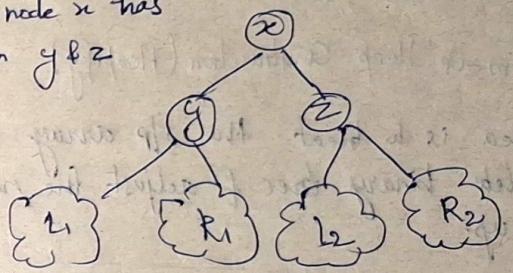
Comprehensive Exams Jan 2023

Answer ① Since a has 2 children, its right subtree is non-empty. Its successor therefore is the leftmost node in the right subtree. But by definition, the leftmost node has no left child.

A similar argument shows that the predecessor of a has no right child.

OR

Suppose that node x has 2 children y & z



Case 1: Predecessor

The predecessor of x is the maximum element, say M, in the subtree R_1. If M has a right child, then that child will be greater than M, which violates the stipulation that M is largest in R_1, showing that M can not have a right child. Note that if R_1 is empty, then y is the predecessor & we have nothing to prove.

Case 2: Successor

The successor of x is the minimum element, say m, in the subtree L_2. (If L_2 is empty, z is successor of

9. A subset of the nodes of a graph G is a dominating set if every other node of G is adjacent to some node in the subset.

   Let DOMINATING-SET $= \{< G, k > | G$ has a dominating set with k nodes$\}$.

   Show that it is NP-complete by giving a reduction from VERTEX-COVER. 6

10. Let RELPRIME be the problem of testing whether two numbers are relatively prime. Show that RELPRIME $\in P$. 6

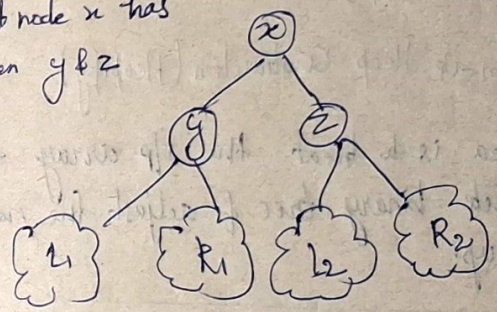
Comprehensive Exams Jan 2023

Answer ① Since a has 2 children, its right subtree is non-empty. The successor therefore is the leftmost node in the right subtree. But by definition, the leftmost node has no left child.

A similar argument shows that the predecessor of a has no right child.

OR

Suppose that node x has 2 children y & z



Case 1: Predecessor

The predecessor of x is the maximum element, say M, in the subtree R_1. If M has a right child, then that child will be greater than M, which violates the stipulation that M is largest in R_1, showing that M can not have a right child. Note that if R_1 is empty, then y is the predecessor & we have nothing to prove.

Case 2: Successor

The successor of x is the minimum element, say m, in the subtree L_2. (If L_2 is empty, z is successor of

16:09 PM

DSA - QP 2021

(1) Prove or disprove the following:

(a) $n! = O(n^n)$

(b) If $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then $h(n) = \Omega(f(n))$

(c) $f(n) = \log_2 n$ is $O(n^\alpha)$ for any $\alpha > 0$

Sol^n (a) $n! = O(n^n)$

We need to determine whether there exists a constant C & N such that $\forall\ n \geq N$, $n! \leq C \cdot n^n$.

Let's analyze the exp. $n!$ & n^n seperately:

(1) $n! = n \cdot (n-1) \cdot (n-2) \cdot \text{---} \cdot 3 \cdot 2 \cdot 1$

(2) $n^n = n \cdot n \cdot n \cdot \text{---} \cdot n$ (with n factors).

Let's compare $n!$ & n^n by considering the ratio $\frac{n!}{n^n}$:

$$\frac{n!}{n^n} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \text{---} \cdot 3 \cdot 2 \cdot 1}{n \cdot n \cdot n \cdot \text{---} \cdot n}$$

$$= \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \text{---} \cdot \frac{3}{n} \cdot \frac{2}{n} \cdot \frac{1}{n}$$

$$\Rightarrow \frac{n!}{n^n} = \frac{1}{n} \cdot \frac{2}{n} \cdot \frac{3}{n} \cdot \cdots \cdot \frac{(n-2)}{n} \cdot \frac{n-1}{n} \cdot \frac{n}{n}$$

$$= \frac{1}{n} \cdot \frac{2}{n} \cdot \frac{3}{n} \cdot \cdots \cdot \frac{n-2}{n} \cdot \frac{n-1}{n} \cdot 1$$

$$\boxed{\frac{n!}{n^n} = \frac{1.2.3.\cdots.(n-2)(n-1)}{n^n}}$$

Now, as n grows larger, each term in the numerator becomes less than or equal to 1.

Thus, $\frac{n!}{n^n} \to O(n)$ approaches infinity.

$\therefore$ $\boxed{n! = O(n^n) \text{ is true}}$

(b) To prove $h(n) = \Omega(f(n))$

Given; $f(n) = O(g(n))$ & $g(n) = O(h(n))$

Sol^n: we need to show that there exist positive constants k & N such that $\forall\ n \geq N$ & then $f(n)$

Since, $f(n) = O(g(n))$, $\exists$ positive constants c_1 & N_1 s.t. $\forall\ n \geq N_1$, $f(n) \leq c_1 \cdot g(n)$.

Similarly, $g(n) = O(h(n))$, $\exists$ +ve c_2 & N_2 s.t. $n \geq N_2$ & $g(n) \leq c_2 \cdot h(n)$

Now, substituting the inequality $f(n) \leq c_1 \cdot g(n)$ into 2nd inequality, we have:

$$f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n)$$

$f(n) = O(h(n))$

This holds $\forall\ n \geq \max(N_1, N_2)$

Let $K = c_1$ & $N = \max(N_1, N_2)$, $\forall\ n \geq N$

$$f(n) \leq k \cdot h(n)$$

Hence, $\boxed{h(n) = \Omega(f(n))}$

(c) To prove that $f(n) = \log_2 n$ is $O(n^\alpha)$ $\forall\ \alpha > 0$

We need to find constant c & N s.t. $f(n) \leq c \cdot n^\alpha$ $\forall\ n \geq N$

Let's choose $\alpha > 0$ & consider n^α, we have:

$$n^\alpha = e^{\alpha \log n}$$

$\boxed{\log n \leq n^\alpha}$ taking exponentials of both sides.

$$2^{\log_2 n} \leq 2^{n^\alpha} \Rightarrow \boxed{n \leq 2^{n^\alpha}}$$

If $N = 1$ & $c = 2$ $\forall\ n \geq N = 1$ $\Rightarrow$ $\boxed{n \leq 2^{n^\alpha}}$

This holds true $\forall\ n$, as exponential fnctn faster than the polynomial fnctn 'n' for any positive value α.

Hence, $f(n) = \log_2 n$ is $o(n^\alpha)$ for any α.

② Using Mathematical induction show that when 'n' is an exact power of 2, the recurrence for best case of M.S is

$$T(n) = \begin{cases} 2 & , \text{if } n=2 \\ 2T(n/2)+n & , \text{if } n=2^i \text{ for } i>0 \end{cases}$$

① proof $T(n) = n\log n$ → Base Case.

② Inductive step: Assume that $T(k) = k\log k$ holds $\forall\ k<n$ & then prove that $T(n) = n\log n$

Proof ① Base case

$\forall\ n=2$, $T(2) = 2$ (given)

Now, $2\log 2 = 2\times 1 = 2$

Hence, base case holds.

② Inductive Step: Assume $T(k) = k\log k$ $\forall\ k<n$ then prove $T(n) = n\log n$

$$T(n) = 2T\left(\frac{n}{2}\right)+n$$

Let's substitute $T(n/2) = \frac{n}{2}\log\left(\frac{n}{2}\right)$ → by inductive hypothesis

$$T(n) = 2\times\frac{n}{2}\log\left(\frac{n}{2}\right)+n$$

$$= n\log\left(\frac{n}{2}\right)+n$$

$$= n(\log n - \log 2)+n$$

$$= n\log n - n + n$$

$$T(n) = n\log n$$ Hence proved.

$T(2^k) = 2^k \log 2^k$

$n = 2^k$

$T(2^{k+1}) = 2T\left(\frac{2^{k+1}}{2}\right) + 2^{k+1}$

$= 2(2^k \log 2^k) + 2^{k+1}$

$= k2^{k+1} + 2^{k+1}$

$= (k+1)2^{k+1} = 2^{k+1}\log 2^{k+1}$

$= n\log n$

③ Given an array A with distinct element & sorted in decreasing order, a Quick-Sort is applied to it. Show the running time of Q.S is $\theta(n^2)$. What will be running time of Q.S when all elements of array A have same value.

Sol^n $T(n) = T(n-1) + \theta(n) + T(0)$

① Base case: when the array size 0 or 1. $T(0) = T(1) = \theta(1)$ → sorted array size.

② Inductive Step: When all elements are the same & the pivot is choosen as the 1st element, in each partition step, one subarray has size $(n-1)$ & other has size 0.

Then, $T(n) = T(n-1) + T(0) + \Theta(n)$ Equ

$T(n) = T(n-1) + \Theta(n)$ Distinct.

$= T(n-1) + cn$

$= T(n-2) + c(n-1) + cn$

$= T(n-3) + c(n-2) + c(n-1) + cn$

$\vdots$

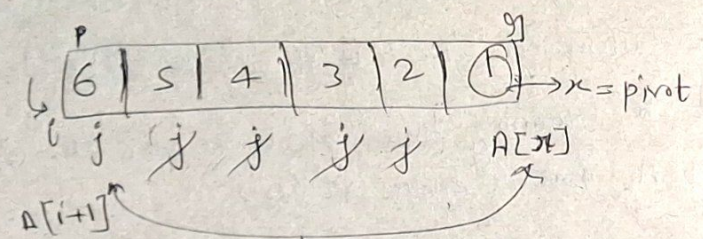
$= c + c2 + c3 + \cdots + cn$

$= c(1 + 2 + \cdots + n)$

$= \frac{c\,(n)(n+1)}{2}$

$T(n) = \Theta(n^2)$ } Worst case

Ex:-



$T(n) = \Theta(n) + T(n-1)$

④ If a queue Q is given, how many stack are needed to replicate the queue Q implement it & Algo. & Algo give the running time of implement.

Ans: To replicate a queue Q using 2-stacks. → S_1, → S_2

① Enqueue operation: → To enqueue an element, simply push it onto stack1 → S_1

② Dequeue operation: → If S_2 is empty, pop all elements from S_1 & push them onto S_2

→ Then, pop the top element from S_2 to dequeue.

Pseudo code

```
QueueUsingStacks:
    Data:  S1, S2 stacks.

Method:

    Enque(x);
        push x onto S1

    Dequeue();
        If S2 is empty:
            While S1 is not empty:
                push S1.pop() onto S2
        If S2 is not empty:
            Return S2.pop()
    Else
```

Else:
Return Null (indicating queue is empty)

IsEmpty():

Return True if both S_1 & S_2 are empty
Else
Return False.

Time Complexity

* Enqueue operation: O(1) (amortized) → Avg. Case
* Dequeue operation: O(n) → worst case (when S_2 is empty & needs to be refilled)

* Overall, the amortized running time for both enqueue & dequeue operation is O(1).

(5) Mr. B.C. Dull claims to have developed a new data structure for priority (Heap) queue that supports the operations Insert, Maximum & Extract-Max all in O(1) worst-case. Prove that he is mistaken.

Ans: Mr. B.C. Dull's seems to contradict the fundamental properties of priority queue & the limitations imposed by the comparison-based model of computation. Here's why?

(1) Insert Operation: Achieving O(1) time complexity for insert operation in a comparison-based model is impossible. Because, in order to maintain the heap property (Ex:- If implementing using max-heap or min-heap) we typically need to compare the inserted elements with its parent & potentially perform swaps to maintain the heap structure. This requires atleast O(log n) → worst case.

(2) Maximum Operation: If max operation (finding max element) then true O(1).
- but only for best Not for amortized & worst-case.

(3) Extracting-Max operation: If Extract-Max (Removing max. element) were truly O(1).
It would imply that removing the max. element does not require any restructuring of the data structure.

∴ Mr. B.C. Dull's claim of supporting Insert, Max & Extract-Max operation in O(1) worst-case time cannot be correct within the bounds of comparison-based model of computation.

⑥ Give asymptotic upper & lower bounds for T(n) in each of following recurrences.

ⓐ $T(n) = 4T(n/2) + n^2\sqrt{n}$

ⓑ $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Sol'n ⓐ $T(n) = 4T(n/2) + n^2 \cdot n^{1/2}$

$T(n) = 4T(n/2) + n^{5/2}$

$a = 4, \; b = 2, \; k = 5/2 \; \& \; p = 0$

$a \; b^k$

$4 < 2^{2.5}$ } Case ③ ⓐ $p \geq 0$

$T(n) = \theta(n^k \log^p n)$

$T(n) = \theta(n^{5/2} \log^0 n)$

$\boxed{T(n) = \theta(n^2\sqrt{n})}$

Sol'n ⓑ $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Each term in the recurrence represents a subproblem of size $n/2^k$ for $k = 1, 2, 3 \dots$

The size each subproblem decreases geometrically & there are such subproblem at each level of recursion.

Lower Bound: At each of recursion, the sum of the subproblem sizes is n. Therefore, total work done at each level is at least n. Since the No. of levels O(logn).

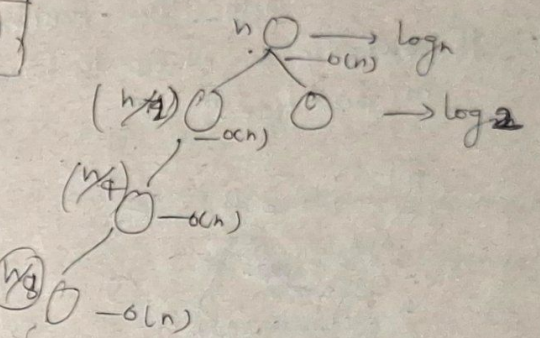
& Total work done is at least O(nlogn)

Upper Case:

At each level of recursion, the work done is O(n) because there are O(1) subproblem & each subproblem has size O(n). Since, there are O(logn) level of recursion, the total work done is O(nlogn)

Amortized case → Avg. case

$\boxed{T(n) = \theta(n \log n)}$



$n = 2^k$

$\log n = k$

∴ O(n) → work done

∴ $T(n) = \theta(n) * \theta(\log n)$

$\boxed{T(n) = \theta(n \log n)}$

⑦ A minimum Bottleneck Spanning Tree (MBST) of an Undirected graph G(V, E) is a spanning tree whose maximum weight edge is minimized.

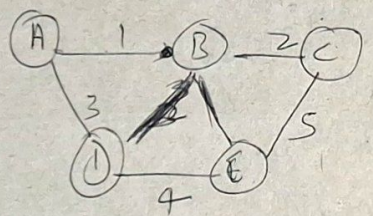
* MST is always a MBST

An MST of a graph is a spanning tree that connects all vertices with the minimum possible total edge weight. Here's why an MST is always MBST:

① MST property: An MST is constructed by selecting edges with the minimum weight that connect all vertices without forming any cycles.

② Observation: In any tree, the weight of the maximum edge is minimized when other edges are also minimized.

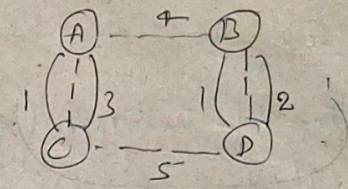
③ Implication: Since, an MST connects all vertices with the minimum total edge weight, it naturally minimizes the weight of the maximum edge in the tree.



MBST has edge weights (1, 2, 3, 4, 5).
However the MST may have a different set of edges such as (1, 2, 3, 4) while MST still minimizes the maximum edge weight (bottleneck), it does not include all edges necessary to connect all vertices with the minimum total weight. ~~Hence~~, ~~MST~~ MBST. (Not necessary)

In Summary, an MST is always MBST because it inherently minimize the weight of the maximum edge in the tree. However, the converse is not always true, as an MBST may not necessarily include all edges that form the minimum total weight spanning

MBST →



* MBST edge set {(A,B), (A,C), (C,D), (B,D)}
where, the max. edge weight is 4.

MST edge set {(A,B), (A,C), (B,D)} ⇒ Total Cost = 6
OR
{(A,B), (B,D), (C,D)} ⇒ 10 = Total Cost

④ To check if 2 BST represent the same set of elements, we can perform an In-Order traversal on both trees & compare the resulting sequences of elements. If the sequence are identical then BSTs represent the same set of elements.

① Perform an In-order traversal on the first BST & store the elements in a list or array L_1

② Perform an In-order traversal on the ~~first~~ 2nd BST & store the elements in $list_2$ or array L_2

③ Compare the lists L_1 & L_2. If they are identical, then BSTs represents the same set of elements.

Pseudo Code

```
Inorder_BST_Traversal (root, result):
    If (root ≠ Null)
        Inorder_BST_Traversal (root.left, results)
        result.append (root.val)
        Inorder_BST_Traversal (root.right, results)

Same_BST_element (root1, root2):
    list1 = empty list
    list2 = empty list.
    Inorder_BST_Traversal (root1, list1)
    Inorder_BST_Traversal (root2, list2)
    If (list1 == list2)
        Return True
    else
        Return False.
```

(9) $\boxed{\sqrt{n} < O(n)}$ without using Median of Medians Algo.

Sol" Using of Quick Selection Algorithm.

$\boxed{I = L-1}$

$N = 2I+1$

$\boxed{N = 2(L-1)+1}$

Row major

$A[I][J] = \text{Base add. of } A + W[I \times M + J]$

Column major

$A[I][J] = \text{Base add. of } A + W[I + J \times M]$

DSA – 2024 QP

Q1 Priority queue → Ans: heap.

Q2 Bipartite graph is ~~not~~ contain → cycle

Q3 $\theta(m+n)$ → Connected graph → m → edge, n → vertex.

Q4 Undirected graph: n → vertex & m → edge varies b/w $(n-1)$ and $\frac{n(n-1)}{2}$

Q5 K – level → has exactly $2^k - 1$ node complete binary tree → proof.

Q6 Inorder & pre-order. Example Reconstruct of

Q7 MST, spanning shortest path → undirected graph example & how they are diff.?

Q8 ~~Example~~ → Result is same.

Q9 $f = \theta(g(n))$ $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ is equal any constant(c)

Q10 $g = O(f)$ then proof $\boxed{f+g = \theta(f)}$

Q11 $T(n) = 2T(n/2) + O(n^2)$ → ?

Q12 Strongly connected digram → + one edge effect? → Q.P 2021 (Reference)

Q13 Bellman ford can detect cycle T/F

Q14 u & v mutually related, u, w mutually related then u & w mutually related T/F

Q16 $Y \leq_p X$ → X is 2^n is polynomial time? → T

Q17 ~~X~~ $\leq_p X$ then Y is otherwise as X. T/F

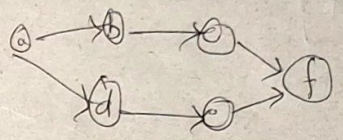
Q18 Max-flow ≠ Min cut → multi-community. T/F

Q19 What is bipartite matching & "Network flow" approach & its time complexity.

Q20 In ~~place~~ binary ~~heap~~, we have d-ary tree then what is index of parent & children & Algo height of d-ary ~~heap~~.

Q21 No. Topological order?

No. of Node = No. of order



6

Q. To prove $f+g = \Theta(f)$ given $g = O(f)$
Then prove that $f+g = \Theta(f)$

We need to show two things:
① $f+g = O(f)$
② $f+g = \Omega(f)$

① $f+g = O(f) \Rightarrow g(n) \le c \cdot f(n) \quad \forall n \ge N$
add $f(n)$ both side.
$f(n)+g(n) \le (c+1)\cdot f(n) \quad \forall n \ge N$

Thus, $f+g = O(f)$

② $f+g = \Omega(f)$

$f(n)+g(n) \ge k \quad \forall n \ge N'$
$g = O(f) \Rightarrow g(n) \le c' f(n) \quad \forall n \ge N'$

add $f(n)$ both side

$f(n)+g(n) \le (c'+1)\cdot f(n) \quad \forall n \ge N'$

Choosing $k=1$ & N' as given above, $f(n)+g(n) \ge f(n) \quad \forall n \ge N$

Thus $f+g = \Omega(f)$

Since, $f+g = O(f)$ & $f+g = \Omega(f)$ then

$f+g = \Theta(f)$ Hence proved

Q. To prove that $f(n) = \Theta(g(n))$. We need to show that there exist positive constants k_1 & $k_2 \in N$ s.t. $n \ge N$

$k_1 \cdot g(n) \le f(n) \le k_2 \cdot g(n)$

Given that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$

Then, by the definition of a limit, we know that $\epsilon > 0$

$\exists\ N$ s.t. $\forall n > N$.

$\left| \frac{f(n)}{g(n)} - c \right| < \epsilon$

Let's first choose $\epsilon = \frac{c}{2}$

For this, there exists an N_1 s.t. $\forall n > N_1$

$\left| \frac{f(n)}{g(n)} - c \right| < \frac{c}{2}$

$\Rightarrow -\frac{c}{2} < \frac{f(n)}{g(n)} < \frac{3c}{2} \quad ; c > 0$

$\Rightarrow \frac{c}{2} \cdot g(n) < f(n) < \frac{3c}{2} \cdot g(n) \Rightarrow k_1 \cdot g(n) < f(n) < k_2 g(n)$

($\frac{c}{2} \to k_1$, $\frac{3c}{2} \to k_2$)

Hence, $f(n) = \Theta(g(n)) \quad \forall n > N$ with constant k_1 & k_2.