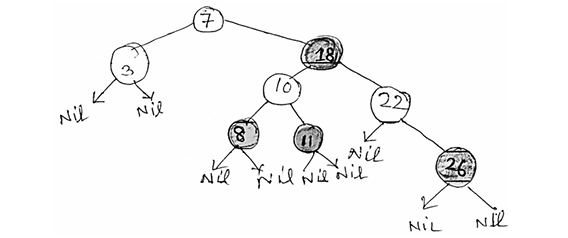
**PYQ of (Analysis and Design of Algorithms) 5 Mark**

**2019**

a) Compare between dynamic programming approach and divide and conquer approach. Write the basic steps to develop a dynamic programming algorithm. Write the name of a problem that can be solved using dynamic programming algorithm. 2+2+1=5

b) Calculate the time complexities of quick sort in case of worst case partitioning and best case partitioning. What is the best case running time of merge sort? 2+2+1=5

c) Write the properties of a red-black tree. Insert 2, 6, 13 in the following red-black tree. [Shaded nodes are red] 2+3=5



d) What is heap? Write an algorithm to construct a heap from an array of data elements. 1+4=5

**2021**

a) Describe depth-first search in brief with a suitable example.

b) Describe decision tree in brief with a suitable example.

c) Write a short note on Divide and Conquer technique for designing an algorithm.

d) Write an algorithm to generate the minimum spanning tree of a graph. Explain the algorithm with an example. 2.5x2=5

**2022**

a. Describe any one technique used for algorithmic complexity analysis in brief.

b. Analyze the average case time complexity of quick sort algorithm.

c. Describe AVL-tree formation technique in brief with at least four nodes.

d. Describe depth first search algorithm for a graph having at least four vertices.

**2022-23**

a) Show that log (1+1/i) = 1/I – 1/2i2 + 1/3i3 – 1/4i4 + 1/5i5+ ………………….. is θ (log n).

b) Why do we use amortised analysis? Distinguish between amortised and average case analysis.

c) State some important properties of red-black tree?

d) Discuss breath first search algorithm in brief with a suitable example.

**ANSWERS**

**2019**

**a) Compare dynamic programming and divide and conquer, and describe steps to develop a dynamic programming algorithm. Mention a problem solvable by dynamic programming.**

**Comparison:**

* **Dynamic Programming (DP):**
  + Solves problems by breaking them into overlapping subproblems.
  + Stores solutions to subproblems to avoid redundant computations (memoization).
  + Suitable for optimization problems with optimal substructure and overlapping subproblems.
* **Divide and Conquer:**
  + Divides the problem into non-overlapping subproblems.
  + Solves each subproblem independently and combines their solutions.
  + Examples: Merge Sort, Quick Sort.

**Steps to Develop a DP Algorithm:**

1. **Characterize the structure of an optimal solution.**
2. **Define the value of an optimal solution recursively in terms of the values of smaller subproblems.**
3. **Compute the value of an optimal solution (typically in a bottom-up manner).**
4. **Construct an optimal solution to the problem.**

**Example Problem:**

* **Fibonacci Sequence:**
  + Recursive formula:

F(n)=F(n−1)+F(n−2)

DP approach involves storing previously computed Fibonacci numbers to avoid redundant calculations.

**b) Calculate time complexities of quicksort in worst and best cases. What is the best case running time of mergesort?**

**Quick Sort:**

* **Worst Case:**
  + Occurs when the pivot divides the array into highly unbalanced partitions (e.g., when the smallest or largest element is always chosen as the pivot).
  + Time Complexity: **O(n²)**.
* **Best Case:**
  + Occurs when the pivot divides the array into two nearly equal parts.
  + Time Complexity: **O(n log n)**.

**Merge Sort:**

* **Best Case:**
  + Merge Sort always divides the array into two halves and merges them, regardless of the initial order.
  + Time Complexity: **O(n log n)**.

**c) Write properties of a red-black tree. Insert nodes (2, 6, 13) into a given red-black tree.**

**Properties of a Red-Black Tree:**

1. **Node Colors:**
   * Each node is either red or black.
2. **Root Property:**
   * The root is always black.
3. **Leaf Property:**
   * All leaves (NIL nodes) are black.
4. **Red Node Property:**
   * Red nodes cannot have red children (no two red nodes can be adjacent).
5. **Black Height Property:**
   * Every path from a node to its descendant NIL nodes has the same number of black nodes.
6. **Balanced Height:**
   * The path from the root to the farthest leaf is no more than twice as long as the path to the nearest leaf, ensuring logarithmic height.

**Insertion of Nodes:**

* **Insert 2:**
  + Inserted as a red node. If it violates any red-black properties, perform rotations and recoloring to restore properties.
* **Insert 6:**
  + Inserted as a red node. Adjust the tree to maintain red-black properties.
* **Insert 13:**
  + Inserted as a red node. Necessary rotations and recoloring are performed to maintain balance.

*Note: The exact tree structure after each insertion depends on the initial configuration and may require specific rotations and recoloring steps.*

**d) What is a heap? Write an algorithm to construct a heap from an array of elements.**

**Heap:**

* A **heap** is a complete binary tree that satisfies the **heap property**:
  + **Max Heap:** The key of each node is greater than or equal to the keys of its children.
  + **Min Heap:** The key of each node is less than or equal to the keys of its children.

**Algorithm to Construct a Max Heap:**

1. **Input:** An array of elements.
2. **Build the heap from the bottom up:**
   * Start from the last non-leaf node and apply the **heapify** operation to each node.
3. **Heapify Operation:**
   * Compare the node with its children.
   * If the node is smaller than its largest child, swap them.
   * Recursively apply heapify to the affected subtree.

*Note: The heapify operation ensures that the subtree rooted at a given node satisfies the heap property.*

**2021**

**a) Describe depth-first search (DFS) with an example.**

**Depth-First Search (DFS):**

* **Definition:**
  + DFS is an algorithm for traversing or searching tree or graph data structures. It starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.
* **Algorithm:**
  + Start from the root node.
  + Mark the node as visited.
  + Visit all the adjacent nodes that have not been visited.
  + Repeat the process for each unvisited adjacent node.
* **Example:**
  + Consider the graph:

A -- B -- D

| |

C -- E

* + DFS traversal starting from node A: **A → B → D → E → C**.

**b) Describe a decision tree with an example.**

**Decision Tree:**

* **Definition:**
  + A decision tree is a flowchart-like structure where each internal node represents a "test" or "decision" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label or decision outcome.
* **Example:**
  + In a decision tree for loan approval:

Is credit score > 700?

/ \

Yes No  
| |

Approved Denied

* This tree helps in making decisions based on the credit score.

**c) Write a short note on the divide and conquer technique.**

**Divide and Conquer:**

* **Definition:**
  + Divide and Conquer is an algorithm design paradigm that involves three steps:
    1. **Divide:** Break the problem into smaller subproblems.
    2. **Conquer:** Solve the subproblems recursively.
    3. **Combine:** Merge the solutions of the subproblems to solve the original problem.
* **Examples:**
  + Merge Sort, Quick Sort, Binary Search.

**d) Write an algorithm to generate a minimum spanning tree of a graph and explain with an example.**

**Prim’s Algorithm (MST):**

1. Start with any node (say A) and mark it as visited.
2. Find the edge with the minimum weight that connects a visited node to an unvisited node.
3. Add the selected edge and the unvisited node to the MST.
4. Repeat until all nodes are visited.

**Example:**

Graph:

A --(1)-- B

| |

(3) (2)

| |

C --(4)-- D

Steps:

* Start with A → Pick edge A-B (1)
* From A or B, pick edge B-D (2)
* From A, B, D → pick edge A-C (3)
* MST Edges: A-B, B-D, A-C
* Total weight = 1 + 2 + 3 = **6**

**2022**

**a) Describe one technique for algorithmic complexity analysis.**

**Asymptotic Analysis:**

* Analyzes the *growth of running time* with respect to input size (n).
* Ignores machine-specific constants.
* Uses notations like:
  + **Big-O (O)**: Upper bound (worst case)
  + **Theta (Θ)**: Tight bound (average case)
  + **Omega (Ω)**: Lower bound (best case)
* Example: For a loop running n times, complexity is O(n).

**b) Analyze the average case time complexity of quicksort.**

**Quick Sort Average Case:**

* In average case, the pivot divides the array into two nearly equal halves.
* Time Complexity:

T(n)=2T(n/2)+O(n)

* Solving the recurrence:

T(n)=O(nlogn)

* So, **average case time complexity is O(n log n)**.

**c) Describe AVL-tree formation with at least four nodes.**

**AVL Tree:**

* A self-balancing binary search tree where the height difference (balance factor) of left and right subtrees is at most 1 for every node.

**Example Insertion:**  
Insert nodes: 30, 20, 10, 25

* Insert 30 → root
* Insert 20 → left of 30
* Insert 10 → left of 20 → imbalance at 30 (Left-Left case)
* Perform **Right Rotation** on 30
* Insert 25 → right of 20 → tree remains balanced

**Final Tree:**

20

/ \

10 30

/

25

**d) Describe the depth-first search algorithm for a graph with at least four vertices.**

**DFS Algorithm:**

1. Start at a node and mark it visited.
2. Recursively visit each unvisited neighbor.
3. Backtrack when no unvisited neighbors are left.

**Example Graph:**

A -- B

| |

C -- D

**DFS starting from A:**

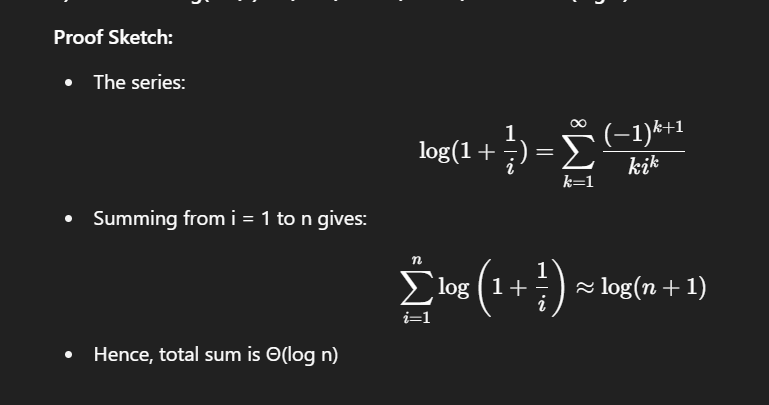
* A → B → D → C

Visited Order: A, B, D, C

**2022–23**

**a) Show that log(1+1/i) = 1/i – 1/2i² + 1/3i³ – 1/4i⁴ + ... is θ(log n).**

**Proof Sketch:**



**b) Why do we use amortized analysis? Distinguish between amortized and average case analysis.**

**Amortized Analysis:**

* Used to analyze the average cost per operation over a sequence of operations.

**Difference:**

| **Feature** | **Amortized Analysis** | **Average Case Analysis** |
| --- | --- | --- |
| Input Dependence | Worst-case sequence | Average over all inputs |
| Use Case | Stack with dynamic array | Random input behavior |
| Cost per op | Over a sequence | Expected over distribution |

**c) State some important properties of a red-black tree.**

**Red-Black Tree Properties:**

1. Each node is either red or black.
2. The root is always black.
3. No two red nodes appear consecutively on a path.
4. Every path from a node to descendant leaves has same number of black nodes.
5. Red-black tree maintains a balanced height (O(log n)).

**d) Discuss breadth-first search (BFS) algorithm with an example.**

**BFS (Breadth-First Search):**

* **Definition:**  
  A graph traversal method that explores all neighbors of a node before moving to the next level.
* **Steps:**
  1. Start from a chosen source node.
  2. Enqueue it and mark as visited.
  3. While the queue is not empty:
     + Dequeue a node.
     + Visit all its unvisited neighbors.
     + Enqueue each neighbor and mark it as visited.
* **Example Graph:**

A

/ \

B C

/

D

* **BFS starting from A:**
  1. Queue: [A] → Visit A
  2. Queue: [B, C] → Visit B
  3. Queue: [C] → Visit C
  4. Queue: [D] → Visit D
* **Traversal Order:** A → B → C → D