ECHELON INSTITUTE OF TECHNOLOGY

Roll No. …….

First Sessional Test, March- 2024
MCA 2nd Semester
ANALYSIS & DESIGN OF ALGORITHMS (MCA-20-102)

Time: 90 Minutes **[Max Marks: 30]**

Instructions:
Note: 1. Question *No. 1. It is compulsory.*
2. Answer any two questions from PART-B.
3. Different sub-parts of a question are to be attempted adjacent to each other.

PART-A
1.

- Explain binary search tree? [CO-1, BTL2] (1)

A binary search tree (BST) is a data structure composed of nodes, where each node has at most two children: a left child and a right child. The key property of a BST is that for any node, all values in its left subtree are less than its value, and all values in its right subtree are greater. This allows for efficient searching, insertion, and deletion operations, with time complexity of O(log n) on average, where n is the number of nodes. However, if the tree is unbalanced, worst-case time complexity can degrade to O(n), making balance maintenance crucial for optimal performance

- What is a circular queue? [CO-1, BTL1] (1)

A circular queue, also known as a ring buffer, is a data structure that operates like a regular queue but with a fixed size. Instead of having a fixed start and end, a circular queue wraps around, so when the end is reached, it loops back to the beginning, effectively creating a circular structure. This allows for efficient use of memory and enables continuous insertion and deletion operations without the need to shift elements when the end or start of the queue is reached. Circular queues are commonly used in situations where a fixed-size buffer is required, such as in operating systems, networking, and memory management.

- Define graphs. [CO-1, BTL2] (1)

Graphs are mathematical structures consisting of a set of vertices (or nodes) and a set of edges (or links) connecting pairs of vertices. They are used to model pairwise relationships between objects. Graphs can be directed, where edges have a direction from one vertex to another, or undirected, where edges have no direction. They can also be weighted, with numerical values assigned to edges to represent properties such as distance or cost. Graphs are widely used in computer science, algorithms, network analysis, social network analysis, transportation systems, and various other fields to model and analyze relationships and connections between entities.

- What is space complexity of an algorithm? [CO-1, BTL1] (1)

The space complexity of an algorithm refers to the amount of memory space required by the algorithm to solve a problem as a function of the input size. It measures the total amount of memory space consumed by the algorithm during its execution, including both auxiliary space (extra space) and input space.
Space complexity is often expressed in terms of Big O notation, just like time complexity.

- What is time complexity of an algorithm? [CO-1, BTL1] (1)

The time complexity of an algorithm describes the amount of time it takes to run as a function of the input size. It quantifies the number of operations or steps performed by the algorithm relative to the size of the input.Time complexity is typically expressed using Big O notation, which provides an upper bound on the growth rate of the algorithm's runtime as the input size increases.

- Explain Divide and Conquer strategy. [CO-2, BTL2] (1)

The Divide and Conquer strategy is a problem-solving approach in computer science and mathematics that involves breaking down a problem into smaller, more manageable subproblems, solving them independently, and then combining the solutions to obtain the final solution to the original problem.

- Explain Greedy Methods. [CO-2, BTL2] (1)

Greedy methods, also known as greedy algorithms, are problem-solving strategies in computer science and mathematics that make locally optimal choices at each step with the hope of finding a globally optimal solution. The greedy approach selects the best available option at each stage of the algorithm, without reconsidering choices made earlier or looking ahead to potential future consequences.

- What is a minimum spanning tree? [CO-2, BTL1] (1)

A minimum spanning tree (MST) is a subset of edges of an undirected weighted graph that connects all the vertices together without forming any cycles and has the minimum possible total edge weight. In simpler terms, it's a tree that spans (covers) all the vertices of the graph with the least total weight.

- What is a pivot in quick sort? [CO-2, BTL1] (1)

In quicksort, a pivot is an element chosen from the array being sorted. The choice of the pivot significantly influences the efficiency of the algorithm. The basic idea behind quicksort is to partition the array around the pivot such that elements smaller than the pivot are placed to its left, and elements greater than the pivot are placed to its right. After partitioning, the pivot element is in its final sorted position.

- What is a single source path? [CO-2, BTL1] (1)

A single-source path refers to a problem in graph theory and algorithms where the objective is to find the shortest path from a single source vertex to all other vertices in a graph. The "source" vertex is the starting point from which paths are calculated.

This problem is commonly encountered in various applications, such as network routing, transportation planning, and shortest distance calculations in maps or social networks.

PART-B

2. (a) Discuss the use of Big and small O notations. [CO-1, BTL4] (5)

Big O and small o notations are mathematical tools used in computer science and mathematics to analyze and describe the behavior of functions or algorithms in terms of their growth rates.

Big O Notation (O):

- Big O notation provides an upper bound on the growth rate of a function or algorithm. It represents the worst-case scenario for the function's behavior.
- Formally, $f(n)$ is said to be $O(g(n))$ if there exist constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- In simpler terms, if $f(n)$ is $O(g(n))$, it means that $f(n)$ grows no faster than $g(n)$ for sufficiently large n.
- Big O notation is commonly used to analyze time complexity and space complexity of algorithms.

Small o Notation (o):

- Small o notation is similar to Big O notation but provides a stricter upper bound. It represents a tighter bound on the growth rate of a function.
- Formally, $f(n)$ is said to be $o(g(n))$ if for any positive constant $c > 0$, there exists a constant n_0 such that for all $n \geq n_0$, $f(n) < c \cdot g(n)$.
- In simpler terms, if $f(n)$ is $o(g(n))$, it means that $f(n)$ grows strictly slower than $g(n)$ for sufficiently large n.
- Small o notation is particularly useful when we want to indicate that the upper bound provided by Big O notation is not tight and can be improved.

Both notations are vital for analyzing the performance and efficiency of algorithms, as they provide insights into how the resource requirements (such as time or space) of an algorithm scale with input size. By understanding the growth rates of functions and algorithms, we can make informed decisions about algorithm selection, optimization strategies, and system design.

(b) Analyze and solve the Knapsack problem for the table A given below. [CO-2, BTL3] (5)

This question was already solved by students.

3. (a) Explain set and disjoints set union. [CO-1, BTL2] (5)

Set:

A set is a collection of distinct elements with no particular order. In mathematics, sets are denoted by curly braces "{}" and contain unique elements. Sets are used to represent groups of objects where each element is unique and has no specific order. For example, the set of all prime numbers less than 10 can be represented as: {2, 3, 5, 7}.

In computer science, sets are also commonly used data structures that allow for efficient insertion, deletion, and retrieval operations. Set data structures typically enforce uniqueness, meaning they do not allow duplicate elements. Sets are fundamental in various algorithms and data structures for tasks like membership testing, eliminating duplicates, and representing relationships between elements.

Disjoint Set Union (Union-Find Data Structure):
Disjoint Set Union, also known as the Union-Find data structure, is a data structure used to efficiently represent a collection of disjoint sets and perform operations such as union (merging sets) and find (determining the set to which an element belongs).

Operations:
MakeSet(x): Creates a new set containing the element x.
Find(x): Returns the representative (also called the root or leader) of the set containing element x.
Union(x, y): Merges the sets containing elements x and y into a single set.
Implementation:
Disjoint Set Union can be implemented using various techniques such as:

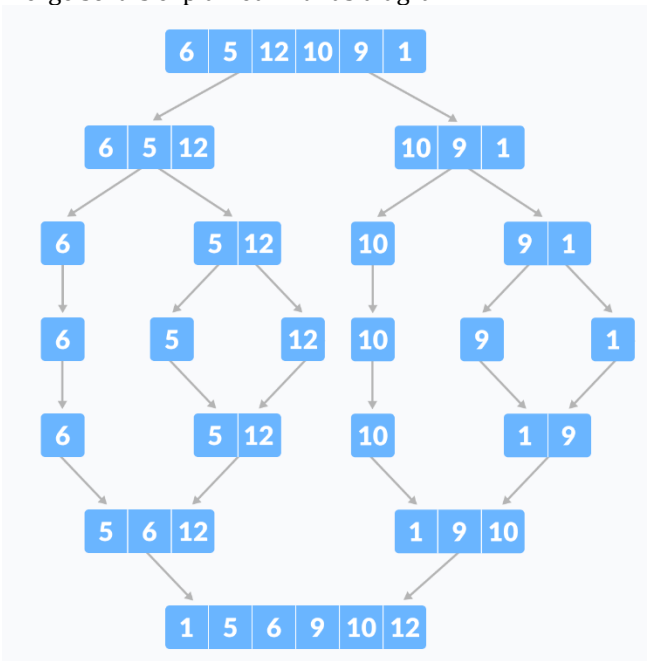
Array Representation: Represent each set as a tree or forest, where each element points to its parent. Path compression and union by rank heuristics can be used to optimize the operations.
Optimized Data Structures: Use balanced trees or other data structures to represent sets and perform operations efficiently.
Applications:
Disjoint Set Union is used in various algorithms and applications, including:

Kruskal's Minimum Spanning Tree Algorithm: Used to find the minimum spanning tree of a graph.
Connected Components: Used to determine the connected components of an undirected graph.
Dynamic Connectivity: Used in dynamic connectivity problems where the set of connections between elements changes over time.
Disjoint Set Union provides an efficient way to manage and manipulate disjoint sets, making it a valuable tool in algorithm design and problem-solving.

(b) Describe merge sort with an example. [CO-2, BTL2] (5)

Merge Sort :

Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm. Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution. For example, here the sorting of an array using merge sort is explained with as diagram.



4. (a) Explain the analyze algorithms.. [CO-1, BTL2] (5)

Analyzing algorithms in terms of time and space complexity involves assessing how their performance scales with respect to the size of the input. Here's how you can analyze algorithms in terms of time and space complexity:

Time Complexity Analysis:

Definition: Time complexity measures the amount of time an algorithm takes to execute as a function of the input size. It quantifies the number of basic operations or steps performed by the algorithm.
Techniques:
Counting Operations: Identify the fundamental operations (e.g., comparisons, assignments, arithmetic operations) executed by the algorithm.
Asymptotic Analysis: Focus on the dominant term or terms in the function that represents the number of operations. Express the time complexity using Big O notation to provide an upper bound on the growth rate of the function.
Example: For a simple linear search algorithm, the time complexity is O(n), indicating a linear relationship between the number of elements in the input and the number of operations performed.

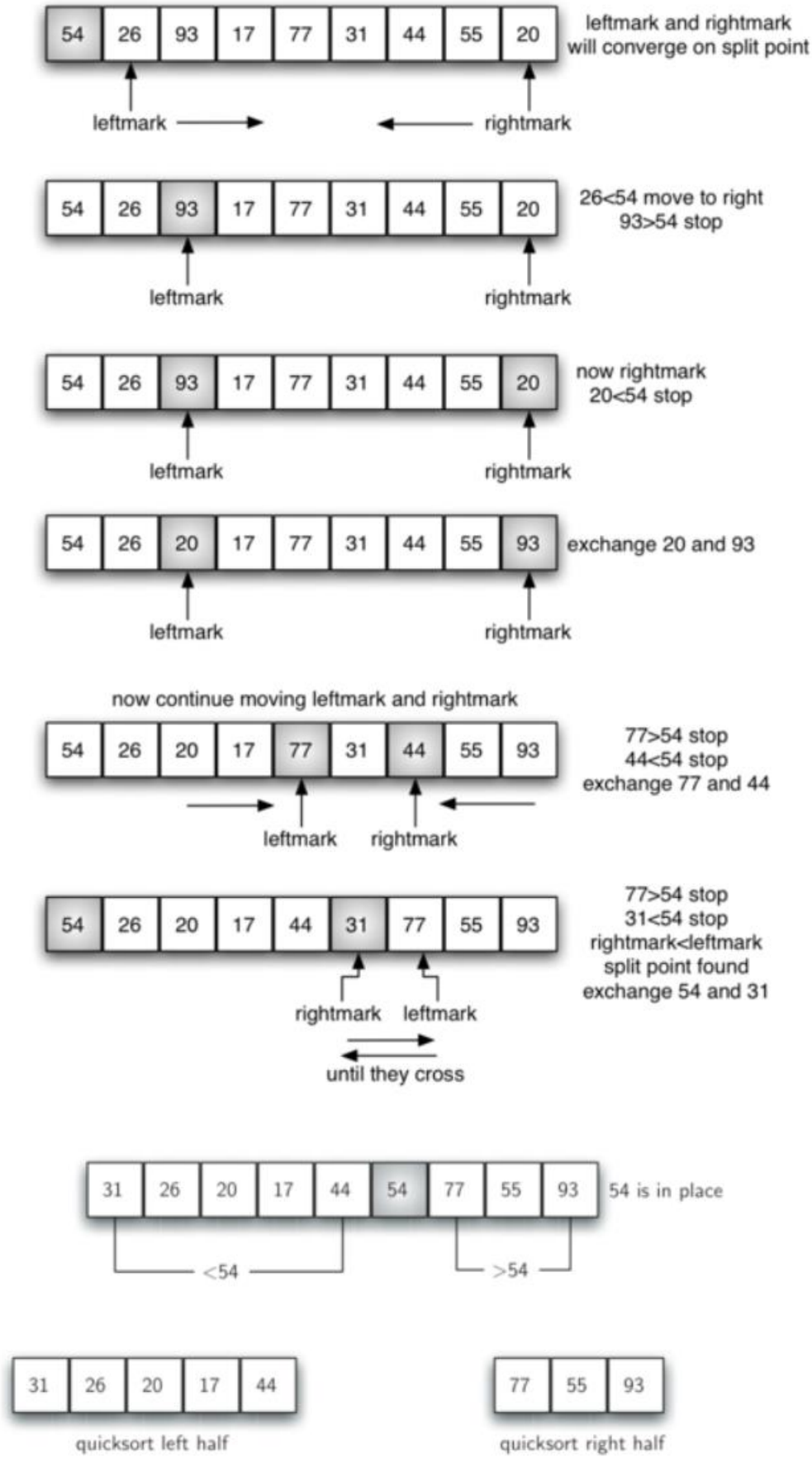
Space Complexity Analysis:

Definition: Space complexity measures the amount of memory space required by an algorithm as a function of the input size. It includes both auxiliary space (extra space) and input space.
Techniques:
Counting Memory Usage: Identify the variables, data structures, and recursive calls that consume memory during the execution of the algorithm.
Asymptotic Analysis: Express the maximum amount of memory used by the algorithm in terms of the input size using Big O notation.
Example: For a recursive algorithm like binary search on an array, the space complexity is O(log n) due to the maximum depth of the recursion stack, which grows logarithmically with the input size.
Comparison and Trade-offs:

Compare the time and space complexity of different algorithms for solving the same problem to identify the most efficient solution.
Consider trade-offs between time and space complexity. Some algorithms may have lower time complexity but higher space complexity, and vice versa. Choose the algorithm that best suits the requirements of the problem and the available computational resources.

(b) Describe quick sort with an example. [CO-2, BTL1] (5)
The quick sort uses divide and conquer to gain the same advantages as the merge sort, while not using additional storage. As a trade-off, however, it is possible that the list may not be divided in half. When this happens, we will see that performance is diminished.
A quick sort first selects a value, which is called the pivot value. Although there are many different ways to choose the pivot value, we will simply use the first item in the list. The role of the pivot value is to assist with splitting the list. The actual position where the pivot value belongs in the final sorted list, commonly called the split point, will be used to divide the list for subsequent calls to the quick sort.
Partitioning begins by locating two position markers—let's call them leftmark and rightmark—at the beginning and end of the remaining items in the list. The goal of the partition process is to move items that are on the wrong side with respect to the pivot value while also converging on the split point.

Consider 54 as the pivot.



All the items to the left of the split point are less than the pivot value, and all the items to the right of the split point are greater than the pivot value. The list can now be divided at the split point and the quick sort can be invoked recursively on the two halves.