|  |  |
| --- | --- |
| Program | BCA |
| Semester | 2 |
| Course | Data Structures and Algorithms |

UNIT 1

## Part B

1. What is an Abstract Data Type (ADT)?  
An Abstract Data Type (ADT) is a theoretical concept in computer science that defines a data type based on its behavior, independent of its implementation. It specifies a set of operations that can be performed on the data, without detailing how these operations are carried out. Examples of ADTs include lists, stacks, queues, trees, and graphs.

2. How are data types classified?

Data types are classified based on their nature and behavior. They can be categorized as primitive data types, which include integers, floating-point numbers, characters, etc., and abstract data types (ADTs), which define data based on their operations rather than their internal representation. ADTs can further be categorized as linear and nonlinear data types, depending on the organization of their elements.

3. What is recursion in programming?  
Recursion is a programming technique where a function calls itself in order to solve smaller instances of the same problem. It involves breaking down a problem into smaller, similar subproblems and solving each subproblem recursively until a base case is reached. Recursion is commonly used to solve problems involving data structures like trees and graphs, as well as mathematical and combinatorial problems.

4. How is dynamic memory allocation used in data structures?  
Dynamic memory allocation is a feature in programming languages that allows the allocation and deallocation of memory at runtime. In data structures, dynamic memory allocation is often used to create flexible structures such as linked lists, trees, and graphs, where the size of the structure may vary during program execution. It enables efficient memory management and allocation of memory as needed, optimizing resource utilization.

5. What are the differences between arrays and linked lists?  
Arrays and linked lists are both linear data structures, but they differ in their implementation and characteristics. Arrays offer constant-time access to elements using indexing but have a fixed size, making it inefficient to insert or delete elements in the middle. Linked lists, on the other hand, provide dynamic memory allocation, allowing for efficient insertion and deletion of elements but have linear-time access and higher memory overhead due to pointers. Arrays have better cache locality, while linked lists offer better flexibility in memory management.

UNIT 2

## Part B

1. What is a Linked List?  
A linked list is a linear data structure consisting of a sequence of elements called nodes, where each node contains a data element and a reference (or pointer) to the next node in the sequence. Unlike arrays, linked lists do not have a fixed size and allow for dynamic memory allocation.

2. What is a Stack?  
A stack is an abstract data type (ADT) that follows the Last In, First Out (LIFO) principle. It supports two main operations: push, which adds an element to the top of the stack, and pop, which removes the top element from the stack.  
3. How is a stack represented using an array?  
A stack can be represented using an array by defining a fixed-size array and maintaining a variable to keep track of the top element's index. When an element is pushed onto the stack, it is added at the top index, and when an element is popped, the top index is decremented.

4. How is a stack represented using a linked list?  
A stack can be represented using a linked list by using the head of the linked list as the top of the stack. Each push operation adds a new node at the head of the list, and each pop operation removes the head node.

5. What are the operations performed on a stack?  
The main operations performed on a stack are push (to add an element to the top of the stack), pop (to remove the top element from the stack), peek (to view the top element without removing it), and isEmpty (to check if the stack is empty).

# Unit 3

## Part B

1. What is a Binary Search Tree (BST)?  
A Binary Search Tree (BST) is a binary tree data structure where each node has at most two children, referred to as the left child and the right child. In a BST, for every node, all elements in the left subtree are less than the node's value, and all elements in the right subtree are greater than the node's value.

2. What are Tree Traversals?  
Tree traversals are methods used to visit and process all nodes in a tree data structure. The three main types of tree traversals are:

* Inorder traversal: Visits the left subtree, then the root node, and finally the right subtree.
* Preorder traversal: Visits the root node, then the left subtree, and finally the right subtree.
* Postorder traversal: Visits the left subtree, then the right subtree, and finally the root node.

3. What are the applications of trees?

* Representing hierarchical data structures like file systems and organizational charts.
* Implementing data storage and retrieval structures like binary search trees (BSTs) and AVL trees.
* Enabling efficient searching, sorting, and traversal algorithms.
* Powering decision-making processes in artificial intelligence and decision trees.
* Facilitating efficient routing algorithms in network design and computer networks.

4. What are the applications of Binary Search Trees (BSTs)?

* Implementing dictionaries and associative arrays for efficient data storage and retrieval.
* Enabling quick insertion, deletion, and searching operations, with average-case time complexity of O(log n) for balanced trees.
* Supporting range queries and ordered iteration due to the sorted nature of BSTs.
* Facilitating symbol tables in compilers and interpreters for programming languages.

5. How is a Heap Data Structure defined?  
A heap is a specialized tree-based data structure that satisfies the heap property. Heaps are commonly implemented as binary trees, where the key stored in each node is either greater than or equal to (max heap) or less than or equal to (min heap) the keys of its children. In a max heap, the root node has the highest key, while in a min heap, the root node has the lowest key.

# Unit 4

## Part B

1. What is sorting, and why is it important in data structures and algorithms?

Sorting is the process of arranging elements in a specific order, typically ascending or descending, according to a defined criterion. It is crucial in data structures and algorithms as it enables efficient searching, retrieval, and manipulation of data. Sorting allows for easier identification of patterns, faster access to desired elements, and streamlined processing of large datasets.

2. Explain Bubble Sort and its basic operation.  
Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, indicating that the list is sorted. Despite its simplicity, Bubble Sort is inefficient for large datasets due to its O(n^2) time complexity.

3. Describe the process of Insertion Sort.  
Insertion Sort is a straightforward sorting algorithm that builds the final sorted array (or list) one item at a time. It iterates through the input list, removing one element at a time and inserting it into the correct position in the sorted part of the list. Insertion Sort is efficient for small datasets or nearly sorted lists but becomes inefficient for larger datasets due to its O(n^2) time complexity.

4. What is Selection Sort, and how does it work?  
Selection Sort is a simple sorting algorithm that divides the input list into two parts: the sorted and unsorted sublists. It repeatedly selects the smallest (or largest) element from the unsorted sublist and swaps it with the first unsorted element. This process continues until the entire list is sorted. Selection Sort has a time complexity of O(n^2) and is not suitable for large datasets.

5. What is the key difference between Linear Search and Binary Search?  
Linear Search is a simple search algorithm that sequentially checks each element in a list until the desired element is found or the end of the list is reached. It has a time complexity of O(n) and is suitable for unordered or small datasets. Binary Search, on the other hand, is a more efficient search algorithm that requires the list to be sorted. It repeatedly divides the search interval in half and narrows down the search space based on the comparison of the target value with the middle element. Binary Search has a time complexity of O(log n) and is ideal for large, sorted datasets.

# Unit 5

## Part B

1. What is a graph in the context of data structures?

In data structures, a graph is a non-linear data structure that consists of a set of vertices (nodes) and a set of edges (connections) that link pairs of vertices. Graphs are used to represent relationships between entities, such as networks, social connections, or transportation routes.

2. How is a graph represented using arrays?  
Graphs can be represented using arrays by creating an adjacency matrix. In an adjacency matrix, rows and columns represent vertices, and each cell indicates whether there is an edge between the corresponding vertices. This representation is suitable for dense graphs but can be memory-intensive for sparse graphs.

3. What is Breadth-First Search (BFS) in graph traversal?  
Breadth-First Search (BFS) is a graph traversal algorithm that systematically explores all vertices in a graph by visiting vertices in layers, starting from the source vertex and moving outward. BFS uses a queue data structure to keep track of the vertices to be visited next and ensures that all vertices at the current layer are visited before moving to the next layer.

4. Explain the concept of Topological Sorting.  
Topological Sorting is an ordering of the vertices in a directed acyclic graph (DAG) such that for every directed edge u -> v, vertex u comes before vertex v in the ordering. Topological Sorting is used in various applications, such as task scheduling, dependency resolution, and job sequencing.

5. Define Hashing and its importance in data structures.  
Hashing is a technique used to map data of arbitrary size to fixed-size values, typically integers, known as hash codes or hash values. Hashing is essential in data structures for efficient storage and retrieval of data, particularly in hash tables. It allows for constant-time average-case access to elements, making it ideal for applications where quick access to data is required.

UNIT 1

## Part C

1. Discuss the importance of algorithm analysis in program design.  
Algorithm analysis is crucial in program design as it helps in evaluating the efficiency and performance of algorithms. By analyzing algorithms, programmers can understand their time and space complexity, enabling them to make informed decisions about algorithm selection and optimization. Efficient algorithms lead to faster execution, reduced resource consumption, and scalability, which are essential for developing high-performance software systems. Moreover, algorithm analysis facilitates the comparison of different algorithms for solving the same problem, aiding in the selection of the most suitable one based on the specific requirements and constraints of the application.

2. Explain the types of linked lists and their applications.  
Linked lists come in various forms, including singly linked lists, doubly linked lists, and circular linked lists. Singly linked lists consist of nodes where each node contains a data element and a reference (or pointer) to the next node in the sequence. Doubly linked lists have nodes with references to both the next and previous nodes, allowing traversal in both directions. Circular linked lists form a closed loop where the last node points back to the first node. Linked lists are widely used in various applications such as implementing dynamic data structures like stacks, queues, and hash tables. They are also used in memory management systems, file systems, and as building blocks for more complex data structures like trees and graphs. Singly linked lists are efficient for applications where forward traversal is predominant, while doubly linked lists offer bidirectional traversal, suitable for operations requiring backward traversal as well. Circular linked lists are useful in applications where continuous looping through the list is required, such as in scheduling algorithms or round-robin processes.

UNIT 2

## Part C

1. Explain the concept of a Queue and its applications.  
A queue is an abstract data type (ADT) that follows the First In, First Out (FIFO) principle. It supports two primary operations: enqueue, which adds an element to the rear of the queue, and dequeue, which removes an element from the front of the queue. Queues are widely used in various applications such as process scheduling, task management, and resource allocation. For example, in computer science, queues are used in network packet scheduling, printer job scheduling, and task scheduling in operating systems. Queues are also used in simulation software, where entities are processed in the order they arrive, mimicking real-world scenarios like customer service queues or traffic flow.

2. Discuss the representation and properties of binary trees.  
A binary tree is a hierarchical data structure consisting of nodes, where each node has at most two children, referred to as the left child and the right child. The topmost node of a binary tree is called the root node. Binary trees exhibit various properties, including:

* Height: The height of a binary tree is the length of the longest path from the root to a leaf node. It represents the tree's maximum depth.
* Depth: The depth of a node is the length of the path from the root to that node.
* Degree: The degree of a node is the number of children it has.
* Balance: A binary tree is balanced if the heights of its left and right subtrees differ by at most one for every node.
* Complete Binary Tree: A binary tree is complete if all levels except possibly the last are completely filled, and the last level is filled from left to right.
* Full Binary Tree: A binary tree is full if every node has either zero or two children.
* Perfect Binary Tree: A binary tree is perfect if it is both full and complete.

Binary trees can be represented using arrays, where each node is assigned an index, and the relationship between parent and child nodes is determined by their indices. This representation enables efficient memory usage and traversal operations. Additionally, binary trees can be implemented using dynamic memory allocation and pointers in programming languages, allowing for flexible and dynamic tree structures.

# Unit 3

## Part C

1. Discuss the construction and deletion operations in a Binary Search Tree (BST).  
Construction: To construct a BST, elements are inserted one by one according to the BST property. When inserting a new element, it is compared with the current node's value. If the new element is less than the node's value, it is inserted in the left subtree; if it is greater, it is inserted in the right subtree. This process continues recursively until a suitable leaf node is found, where the new element is inserted.  
Deletion: Deleting a node from a BST involves three cases:

1. Deleting a leaf node: Simply remove the node from the tree.
2. Deleting a node with one child: Replace the node with its child.
3. Deleting a node with two children: Find the node's in-order successor (or predecessor), copy its value to the node to be deleted, and then recursively delete the successor node.

2. Explain the implementation and applications of Heap Data Structure.  
Implementation: Heaps are commonly implemented as binary trees, usually stored in arrays for efficient memory usage. In a binary heap, the elements are stored in a complete binary tree, where each node's value is greater than or equal to (max heap) or less than or equal to (min heap) its children's values. Heap operations such as insertion, deletion, and heapify (restructuring the heap to maintain the heap property) are performed to maintain the heap structure.

Applications: Heaps have various applications, including:

* + Priority queues: Heaps are used to implement priority queues, where elements with higher priorities are dequeued first.
  + Heap sort: Heapsort is an efficient sorting algorithm that utilizes the heap data structure to achieve a time complexity of O(n log n).
  + Memory allocation: Heaps are used in memory management systems to allocate and deallocate memory dynamically.
  + Graph algorithms: Heaps are used in algorithms like Dijkstra's shortest path algorithm and Prim's minimum spanning tree algorithm for efficient selection of minimum-weight edges or vertices.

# Unit 4

## Part C

1. Compare and contrast Quick Sort and Merge Sort.  
Quick Sort: Quick Sort is a divide-and-conquer sorting algorithm that selects a pivot element and partitions the input list into two sublists: elements less than the pivot and elements greater than the pivot. It recursively sorts the sublists and combines them to form the final sorted list. Quick Sort has an average-case time complexity of O(n log n) and is efficient for large datasets. However, its worst-case time complexity is O(n^2) when the pivot selection is poor.

Merge Sort: Merge Sort is also a divide-and-conquer sorting algorithm that divides the input list into smaller sublists until each sublist contains only one element. It then merges the sublists in a sorted manner to produce a sorted output. Merge Sort has a guaranteed time complexity of O(n log n) for all cases and is stable, meaning it preserves the order of equal elements. However, Merge Sort requires additional space for merging sublists, making it less memory-efficient than Quick Sort.

2.Discuss the comparison between different search algorithms, including Linear Search and Binary Search.  
Linear Search: Linear Search sequentially checks each element in a list until the target element is found or the end of the list is reached. It has a time complexity of O(n) and is suitable for unordered or small datasets. Linear Search is simple to implement and does not require the list to be sorted.

Binary Search: Binary Search is a more efficient search algorithm that requires the list to be sorted. It repeatedly divides the search interval in half and narrows down the search space based on the comparison of the target value with the middle element. Binary Search has a time complexity of O(log n) and is ideal for large, sorted datasets. However, it requires additional preprocessing to sort the list initially. Binary Search is not suitable for unordered or dynamically changing datasets.

Comparison:

* Binary Search is more efficient than Linear Search for large datasets, as it has a time complexity of O(log n) compared to O(n) for Linear Search.
* Linear Search is simpler to implement and does not require the list to be sorted, making it suitable for unordered or small datasets.
* Binary Search requires the list to be sorted initially, while Linear Search does not have this requirement.
* Binary Search is not suitable for dynamically changing datasets, as the list must remain sorted for the algorithm to work correctly.

# Unit 5

## Part C

1. Discuss the concept of Minimum Spanning Tree (MST) and its algorithms: Prim's and Kruskal's.  
Minimum Spanning Tree (MST): A Minimum Spanning Tree is a subset of edges of a connected, undirected graph that connects all vertices together with the minimum possible total edge weight. MSTs are used in network design, clustering, and circuit design, among other applications.

Prim's Algorithm: Prim's algorithm is a greedy algorithm that starts with an arbitrary vertex and grows the MST by adding the shortest edge that connects a vertex in the MST to a vertex outside the MST. This process continues until all vertices are included in the MST.

Kruskal's Algorithm: Kruskal's algorithm is also a greedy algorithm that builds the MST by repeatedly adding the shortest edge that does not form a cycle. It starts with the smallest edge and adds edges in increasing order of weight until all vertices are connected.

2. Explain the concept of Hashing in detail, including hash functions and collision avoidance techniques.  
Hashing: Hashing is a technique used to convert a given key into a smaller range of values, typically an index in an array, known as the hash code or hash value. Hashing is used in data structures to quickly locate a data record given its search key.

Hash Functions: Hash functions are algorithms that take an input (or key) and produce a fixed-size output, the hash code. An ideal hash function distributes keys uniformly across the hash table, minimizing collisions.

Collision Avoidance: Collisions occur when two keys hash to the same index. Collision avoidance techniques include:

* Open Addressing: In open addressing, collisions are resolved by finding an alternative location within the hash table for the colliding key.
* Separate Chaining: In separate chaining, each bucket of the hash table is associated with a linked list, allowing multiple keys to hash to the same index. Collisions are resolved by appending the colliding key to the linked list.

Advantages of Hashing: Hashing provides constant-time average-case access to elements, making it efficient for storing and retrieving data. It also allows for fast insertion and deletion operations, making it suitable for dynamic data structures like hash tables. Additionally, hashing can be used to check for data integrity and ensure uniqueness of keys.

# Unit 1

## Part A

1. What is a fundamental characteristic of an Abstract Data Type (ADT)?

A) It only represents physical data structures.

B) It exposes the implementation details to the user.

C) It defines operations on data without specifying the implementation.

D) It can only be used with linear data structures.

Correct Answer: C) It defines operations on data without specifying the implementation.

2. Which notation is used to describe the efficiency of an algorithm as the size of the input grows?

A) Asymptotic Notation

B) Concrete Notation

C) Explicit Notation

D) Descriptive Notation

Correct Answer: A) Asymptotic Notation

3. In the context of data structures, what is a benefit of using dynamic memory allocation?

A) It leads to static memory management.

B) It allows for the allocation of memory during compile-time.

C) It enables the allocation of memory at runtime.

D) It restricts the size of data structures.

Correct Answer: C) It enables the allocation of memory at runtime.

4. Which of the following is a non-linear data structure?

A) Array

B) Linked List

C) Queue

D) Tree

Correct Answer: D) Tree

5. What is the primary difference between a linked list and an array?

A) Arrays allow for dynamic memory allocation.

B) Linked lists have constant-time access to elements.

C) Arrays store elements in contiguous memory locations.

D) Linked lists require a fixed size declaration.

Correct Answer: C) Arrays store elements in contiguous memory locations.

6. Which of the following operations is not typically performed on data structures?

a) Insertion

b) Deletion

c) Sorting

d) Multiplication

Correct Answer: Multiplication

7. What is a key feature of recursion in programming?

A) It avoids using function calls.

B) It relies on iterative loops for execution.

C) It involves a function calling itself.

D) It can only be applied to linear data structures.

Correct Answer: C) It involves a function calling itself.

8. Which data structure is best suited for implementing a stack?

A) Linked List

B) Queue

C) Array

D) Tree

Correct Answer: C) Array

9. What does Algorithm Analysis primarily focus on?

A) The physical representation of data structures.

B) The design and implementation of algorithms.

C) The classification of data types.

D) The efficiency and performance of algorithms.

Correct Answer: D) The efficiency and performance of algorithms.

10. Which data structure utilizes pointers to connect elements?

A) Stack

B) Array

C) Queue

D) Linked List

Correct Answer: D) Linked List

# Unit 2

## Part A

1. Which data structure follows the Last-In-First-Out (LIFO) principle?

A) Queue

B) Linked List

C) Stack

D) Tree

Correct Answer: C) Stack

2. What is the purpose of recursion in problem-solving?

a) To make algorithms more complex

b) To make algorithms easier to understand

c) To break down problems into smaller subproblems

d) To avoid using loops

Correct Answer: To break down problems into smaller subproblems

3. What is a fundamental characteristic of a stack data structure?

A) Allows random access to elements

B) Follows FIFO (First-In, First-Out) order

C) Supports dynamic resizing of elements

D) Follows LIFO (Last-In, First-Out) order

Correct Answer: D) Follows LIFO (Last-In, First-Out) order

4. Which data structure is typically used to implement undo functionality in text editors?

A) Stack

B) Queue

C) Tree

D) Linked list

Correct Answer: A) Stack

5. How is a circular queue different from a regular queue?

A) Circular queue has a fixed size.

B) Circular queue does not support enqueue operation.

C) Circular queue allows elements to be removed from anywhere.

D) Circular queue has no front or rear end.

Correct Answer: A) Circular queue has a fixed size.

6. What is a characteristic of a priority queue?

A) Elements are stored based on their insertion order.

B) Elements are removed based on their priority level.

C) Elements can only be inserted at the rear end.

D) Elements are removed randomly.

Correct Answer: B) Elements are removed based on their priority level.

7. Which tree traversal visits the root node after traversing both left and right subtrees?

A) Inorder traversal

B) Preorder traversal

C) Postorder traversal

D) Level order traversal

Correct Answer: B) Preorder traversal

8. What is a property of a complete binary tree?

A) All nodes have at most one child.

B) All levels are completely filled except possibly the last level.

C) All nodes have two children.

D) All leaf nodes are at the same level.

Correct Answer: B) All levels are completely filled except possibly the last level.

9. How is a binary tree typically represented using arrays?

A) Each node has two pointers to its children.

B) Each node's data is stored in an array.

C) Each node's children are stored in separate arrays.

D) Each node's parent index is stored in the array.

Correct Answer: B) Each node's data is stored in an array.

10. What is a disadvantage of using an array to represent a stack?

A) Limited capacity

B) Dynamic memory allocation

C) Complex operations

D) Variable size

Correct Answer: A) Limited capacity

# Unit 3

## Part A

1. Which traversal of a binary tree visits the nodes in the order: left subtree, root, right subtree?

A) Inorder traversal

B) Preorder traversal

C) Postorder traversal

D) Level-order traversal

Correct Answer: A) Inorder traversal

2. In which tree traversal does a node's value get processed after visiting its left and right subtrees?

A) Preorder traversal

B) Inorder traversal

C) Postorder traversal

D) Level-order traversal

Correct Answer: C) Postorder traversal

3. Which type of binary tree ensures that for every node, the value of all nodes in its left subtree is lesser and the value of all nodes in its right subtree is greater?

A) Binary Search Tree (BST)

B) AVL Tree

C) Threaded Binary Search Tree

D) Expression Tree

Correct Answer: A) Binary Search Tree (BST)

4. What is the purpose of threading in a Threaded Binary Search Tree (TBST)?

A) To add additional threads to increase concurrency

B) To reduce the time complexity of tree traversals

C) To allow traversal without using recursion or a stack

D) To improve the space efficiency of the tree

Correct Answer: C) To allow traversal without using recursion or a stack

5. How is a binary search tree typically constructed from a given set of elements?

A) By randomly inserting elements

B) By arranging elements in a sorted order

C) By inserting elements in a zigzag pattern

D) By swapping elements to maintain a balanced structure

Correct Answer: B) By arranging elements in a sorted order

6. In a binary search tree, what is the time complexity of searching for a specific element?

A) O(1)

B) O(log n)

C) O(n)

D) O(n log n)

Correct Answer: B) O(log n)

7. What is an example of an application of trees in computer science?

A) Sorting algorithms

B) Database indexing

C) Cryptography

D) Digital image processing

Correct Answer: B) Database indexing

8. What is a characteristic feature of an AVL tree?

A) It is always a complete binary tree

B) It guarantees constant-time search operations

C) It automatically rebalances itself to maintain height balance

D) It allows duplicate elements in the tree

Correct Answer: C) It automatically rebalances itself to maintain height balance

9. Which operation is typically used to maintain the heap property after insertion in a minimum heap?

A) Heapify Up

B) Heapify Down

C) Bubble Up

D) Bubble Down

Correct Answer: A) Heapify Up

10. What is a common application of heaps in computer science?

A) Symbol tables

B) Graph traversal algorithms

C) Text processing

D) Disk scheduling algorithms

Correct Answer: B) Graph traversal algorithms

# Unit 4

## Part A

1. What is the primary objective of sorting algorithms?

A) To rearrange elements in descending order

B) To rearrange elements in ascending order

C) To remove duplicate elements from a list

D) To perform mathematical operations on elements

Correct Answer: B) To rearrange elements in ascending order

2. What does the efficiency of an algorithm primarily refer to?

A) The number of steps required to complete the algorithm

B) The amount of memory consumed by the algorithm

C) The number of comparisons made by the algorithm

D) The degree of parallelism in the algorithm

Correct Answer: A) The number of steps required to complete the algorithm

3. Which complexity measure assesses the amount of memory an algorithm consumes?

A) Time complexity

B) Space complexity

C) Computational complexity

D) Efficiency complexity

Correct Answer: B) Space complexity

4. In which sorting algorithm do elements "bubble" to their correct positions by repeatedly comparing adjacent elements?

A) Insertion Sort

B) Selection Sort

C) Quick Sort

D) Merge Sort

Correct Answer: A) Insertion Sort

5. What is a key characteristic of the Selection Sort algorithm?

A) It divides the array into smaller subarrays

B) It repeatedly selects the smallest element and swaps it with the current element

C) It recursively divides and conquers the array

D) It combines sorted subarrays to produce the final sorted array

Correct Answer: B) It repeatedly selects the smallest element and swaps it with the current element

6. What is a key feature of Merge Sort?

A) It works efficiently on small datasets

B) It uses a pivot element to partition the array

C) It divides the array into two halves, sorts them, and then merges them

D) It relies on swapping adjacent elements to sort the array

Correct Answer: C) It divides the array into two halves, sorts them, and then merges them

7. Which sorting algorithm is well-suited for sorting integers or strings with fixed lengths?

A) Bubble Sort

B) Insertion Sort

C) Radix Sort

D) Shell Sort

Correct Answer: C) Radix Sort

8. What is a key property of Heap Sort?

A) It is an unstable sorting algorithm

B) It relies on comparing adjacent elements to sort the array

C) It builds a binary heap from the array and repeatedly extracts the maximum element

D) It recursively divides the array into smaller subarrays

Correct Answer: C) It builds a binary heap from the array and repeatedly extracts the maximum element

9. In which search algorithm does the array need to be sorted beforehand for efficient searching?

A) Linear Search

B) Binary Search

C) Quick Search

D) Merge Search

Correct Answer: B) Binary Search

10. What is a primary advantage of Binary Search over Linear Search?

A) Binary Search is more straightforward to implement

B) Binary Search has a better worst-case time complexity

C) Binary Search works efficiently on unsorted arrays

D) Binary Search is suitable for searching in linked lists

Correct Answer: B) Binary Search has a better worst-case time complexity

# Unit 5

## Part A

1. What is a cycle in a graph?

A) A path that visits each node exactly once

B) A path that visits the same node twice

C) A path that connects two distinct nodes

D) A path with only one node

Correct Answer: B) A path that visits the same node twice

2. How is a graph typically represented using arrays?

A) Each node stores a list of adjacent nodes

B) Each node stores its position in the graph

C) Each node stores a list of incoming edges

D) Each node stores its degree

Correct Answer: A) Each node stores a list of adjacent nodes

3. What does a topological sort of a directed acyclic graph (DAG) ensure?

A) All nodes are visited exactly once

B) There are no cycles in the graph

C) The graph is connected

D) The graph contains the maximum number of edges

Correct Answer: B) There are no cycles in the graph

4. Which data structure is commonly used to implement a graph using arrays?

A) Array of linked lists

B) Array of stacks

C) Array of queues

D) Array of trees

Correct Answer: A) Array of linked lists

5. What does a minimum spanning tree of a graph represent?

A) The shortest path between any two nodes in the graph

B) The tree that connects all nodes with the minimum edge weight sum

C) The longest path between any two nodes in the graph

D) The tree that connects only a subset of nodes in the graph

Correct Answer: B) The tree that connects all nodes with the minimum edge weight sum

6. What is the primary goal of Kruskal's Algorithm?

A) To find the shortest path between two nodes in a graph

B) To find the maximum spanning tree of a graph

C) To find the minimum spanning tree of a graph

D) To find the longest path between two nodes in a graph

Correct Answer: C) To find the minimum spanning tree of a graph

7. What is the primary application of hashing in computer science?

A) Sorting elements in an array

B) Searching for elements in a linked list

C) Mapping keys to values in a key-value store

D) Representing hierarchical relationships between elements

Correct Answer: C) Mapping keys to values in a key-value store

8. What is the purpose of a hash function in hashing?

A) To ensure all keys are unique

B) To map keys to indices in a hash table

C) To minimize collisions between keys

D) To maximize the load factor of the hash table

Correct Answer: B) To map keys to indices in a hash table

9. How does separate chaining handle collisions in hashing?

A) By storing collided keys in the same array index

B) By rehashing collided keys to different indices

C) By using linear probing to find an empty slot

D) By resizing the hash table to accommodate collided keys

Correct Answer: A) By storing collided keys in the same array index

10. Which data structure is commonly used to implement a minimum spanning tree?

A) Stack

B) Queue

C) Array

D) Priority Queue

Correct Answer: D) Priority Queue