IMPACT COLLEGE OF ENGINEERING AND APPLIED SCIENCES

(Approved By AICTE & Affiliated to VTU)

BCSL305 - DATA STRUCTURES LABORATORY

VIVA QUESTIONS

1. What is a data structure?
- A data structure is a way of organizing and storing data in a computer so that it can be accessed and modified efficiently.
2. Explain the difference between an array and a linked list.
- An array is a collection of elements stored in contiguous memory locations, while a linked list is a collection of nodes, where each node contains data and a reference to the next node in the sequence.
3. What is the difference between a linear and a non-linear data structure?
- In a linear data structure, elements are arranged sequentially (e.g., arrays, linked lists), while in a non-linear data structure, elements form a hierarchy or a network (e.g., trees, graphs).

4. What is the difference between a stack and a queue?

- A stack follows Last In, First Out (LIFO) order, while a queue follows First In, First Out (FIFO) order.

5. **What are the main types of data structures?**
- The main types include arrays, linked lists, stacks, queues, trees, graphs, and hash tables.

6. **How is memory allocated for arrays?**
Memory for arrays is allocated in a contiguous block, meaning all elements are stored consecutively in memory.
7. Difference between static and dynamic memory allocation.

. Static Memory is allocated for declared variables by the compiler. The address can be found using the address of operator and can be assigned to a pointer. The memory is allocated during compile time.

. Memory allocation done at the time of execution (run time) is known as dynamic memory allocation.
8. **Explain how a singly linked list differs from a doubly linked list.**

In a singly linked list, each node has a reference to the next node. In a doubly linked list, each node has references to both the next and previous nodes.

9. **What are the advantages and disadvantages of linked lists over arrays?**
- **Advantages:** Dynamic size, efficient insertions/deletions. **Disadvantages:** Random access is not possible, and extra memory is required for pointers.
10. **How do you insert and delete an element in a linked list?**
- To insert, adjust the pointers of the adjacent nodes. To delete, locate the node, adjust the pointers of adjacent nodes, and free the memory of the node.
11. **How do you find the length of a linked list?**
- Traverse the linked list and increment a counter until the end of the list is reached.

12. **What is a stack? Give some real-life examples.**

    A stack is a LIFO data structure. Examples include a pile of plates or the backtracking feature in web browsers.

13. **What operations can be performed on a stack?**

    The main operations are `push` (add an item), `pop` (remove the top item), and `peek` (view the top item).

14. **What is a queue? How does it differ from a stack?**

    A queue is a FIFO data structure. Unlike a stack, elements are added at the back and removed from the front.

15. **Explain the FIFO principle in queues.**

    FIFO means "First In, First Out," where the first element added to the queue is the first one to be removed.

16. **What are circular queues, and why are they used?**

    A circular queue connects the end back to the front, allowing efficient use of space by reusing empty positions at the beginning.

17. **How can you implement a stack and queue using arrays?**

    For a stack, use an array with a pointer to the top element. For a queue, use two pointers to track the front and rear positions.

18. **How can you implement a stack and queue using linked lists?**

    For a stack, link nodes where each new element is the new head. For a queue, maintain two pointers for the front and rear of the list.

19. **What is a tree data structure?**

A tree is a hierarchical structure with a root node and child nodes, with no cycles, where each node has zero or more child nodes.

20. **Explain the difference between a binary tree and a binary search tree.**

A binary tree allows up to two children per node. In a binary search tree (BST), the left child is smaller, and the right child is greater than the parent.

21. **What is a balanced tree?**

A balanced tree maintains its height to a minimum for optimal performance. AVL and Red-Black trees are examples of balanced trees.

22. **What is a binary search tree, and how do you insert elements into it?**

A BST is a tree structure that allows fast searching, insertion, and deletion. Insert by comparing the new element with nodes, placing it in the left subtree if smaller and the right if larger.

23. **What is a graph, and what are its types?**

A graph is a collection of nodes (vertices) connected by edges. Types include directed, undirected, weighted, and unweighted graphs.

24. **Explain the terms "adjacency matrix" and "adjacency list."**

An adjacency matrix is a 2D array representing graph edges, while an adjacency list uses lists to store neighbors of each vertex.

25. **What is a depth-first search (DFS), and how does it work?**

DFS explores as far down a path as possible before backtracking. It uses a stack or recursion.

26. **What is a breadth-first search (BFS), and how does it work?**

BFS explores all neighbors level by level, using a queue for traversal.

27. **How can you detect a cycle in a graph?**

For undirected graphs, DFS with a parent check can detect cycles. For directed graphs, DFS with recursion stack checks for back edges.

28. **What are the applications of trees and graphs in real life?**

Trees are used in file systems, and decision-making. Graphs are used in social networks, route planning, and computer networks.

29. **What is hashing?**

Hashing is a technique for mapping data to specific indexes in a hash table using a hash function.

30. **What is a hash function?**

A hash function maps input data to a fixed-size output, often used to index data in a hash table.

31. **What is a hash collision, and how can it be resolved?**

A collision occurs when different inputs produce the same hash. It can be resolved with chaining or open addressing.

32. **Explain open addressing and chaining.**

Open addressing finds the next empty slot when a collision occurs, while chaining stores collided elements in a linked list.

33. **What are the advantages of using a hash table?**

Hash tables provide fast access (average O(1) time) for search, insert, and delete operations.

34. **Explain the difference between linear and binary search.**

Linear search sequentially checks each element, while binary search repeatedly divides a sorted list, providing faster search.

35. **What is the time complexity of binary search?**

O(log n), where n is the number of elements.

36. **Describe the bubble sort algorithm.**

Bubble sort repeatedly swaps adjacent elements if they are in the wrong order, bubbling the largest unsorted element to the end.

37. **Describe the quicksort algorithm and its complexity.**

Quicksort partitions the array around a pivot, sorting recursively. Average complexity: O(n log n), worst-case: O(n^2).

38. **Explain the merge sort algorithm.**

Merge sort divides the array, recursively sorts, then merges sorted halves. It has a time complexity of O(n log n).

39. **What is the difference between quicksort and mergesort?**

Quicksort is in-place and faster on average but can be slower in the worst case. Mergesort has a stable O(n log n) complexity and requires extra space.

40. **What is insertion sort, and when is it useful?**

Insertion sort builds a sorted array by inserting each element into its proper position. It's efficient for small or nearly sorted arrays.

41. **What is heap sort?**

Heap sort uses a binary heap to sort an array, with O(n log n) time complexity, and it’s an in-place sorting algorithm.

Complexity and Analysis

42. **What is Big O notation, and why is it important?**

Big O notation describes an algorithm's worst-case time or space complexity, helping analyze its efficiency.

43. **Explain the time complexity of a linked list search.**

O(n), as each element may need to be checked to find a specific value.

44. **What is the space complexity of an algorithm?**

It measures the amount of memory required by an algorithm in terms of input size.

45. **How do you measure the efficiency of a data structure?**

By analysing the time and space complexity of its operations, like insertion, deletion, and access.

46. **Compare the time complexities of various sorting algorithms.**

Quick and merge sort: O(n log n), bubble and selection sort: O(n^2), and insertion sort: O(n^2) on average.