Viva Questions

1. What is a data structure?
A data structure is a specialized format for organizing, processing, and storing data in a computer so that it can be accessed and modified efficiently.

2. Can you classify the data types?
Data types can be classified into primitive types (e.g., int, char, float) and composite types (e.g., arrays, structures, classes).

3. What are the common operations in any type of data structure?
Common operations include insertion, deletion, traversal, searching, and updating elements.

4. Define structure.
A structure is a user-defined data type that allows the grouping of variables of different data types under a single name.

5. How can you access elements in an array?
Elements in an array can be accessed using their index, e.g., array[index].

6. What is a self-referential structure?
A self-referential structure is a structure that contains a pointer to another structure of the same type.

7. How does a structure differ from a union?
In a structure, all members can occupy separate memory locations, while in a union, all members share the same memory location.

8. What is a pointer?
A pointer is a variable that stores the memory address of another variable.

9. How will you access members of a structure? Write the syntax for that.
You can access members of a structure using the dot operator for normal structures and the arrow operator for pointers to structures:

struct_name.member; // for structure

pointer->member; // for pointer to structure

10. What is DMA and how does it differ from SMA?
DMA (Dynamic Memory Allocation) allocates memory at runtime, while SMA (Static Memory Allocation) allocates memory at compile time.

11. How many functions are there in DMA and what are they?
Common functions in DMA include:

malloc()

calloc()

realloc()

free()

12. What is a dynamically allocated array?
A dynamically allocated array is an array whose size can be defined at runtime using DMA functions.

13. Explain the steps for performance analysis of an algorithm.
Steps include:

Identify the basic operations.

Count the number of operations in terms of input size.

Analyze time complexity (Big O notation).

Evaluate space complexity.

14. What is space complexity?
Space complexity is the amount of memory space required by an algorithm in relation to the input size.

15. What is time complexity?
Time complexity measures the time an algorithm takes to complete as a function of the input size.

16. What is a stack?
A stack is a linear data structure that follows the Last In First Out (LIFO) principle.

17. What is a queue?
A queue is a linear data structure that follows the First In First Out (FIFO) principle.

18. What are the types of representations of a stack?
A stack can be represented using an array or a linked list.

19. What are the operations of a stack and explain them?

Push: Add an element to the top of the stack.

Pop: Remove and return the top element from the stack.

Peek/Top: Return the top element without removing it.

IsEmpty: Check if the stack is empty.

20. What are the operations of a queue and explain them?

Enqueue: Add an element to the rear of the queue.

Dequeue: Remove and return the front element from the queue.

Front: Return the front element without removing it.

IsEmpty: Check if the queue is empty.

21. How will you state that the stack is in the initial position and full in array representation?

Initial position: top = -1

Full: top = max_size - 1

22. How will you state that the stack is in the initial position and full in linked list representation?

Initial position: top = NULL

Full: Not applicable, as a linked list can grow until memory is exhausted.

23. How will you state that the queue is in the initial position and full in array representation?

Initial position: front = -1, rear = -1

Full: (rear + 1) % max_size == front

24. How will you state that the queue is in the initial position and full in linked list representation?

Initial position: front = NULL, rear = NULL

Full: Not applicable, as a linked list can grow until memory is exhausted.

25. What is a Polish notation?
Polish notation (prefix notation) is a mathematical notation where operators precede their operands.

26. What is a reverse Polish notation?
Reverse Polish notation (postfix notation) is a mathematical notation where operators follow their operands.

27. What are the types of recursions?

Direct recursion: A function calls itself.

Indirect recursion: A function calls another function, which then calls the first function.

28. What is a circular queue?
A circular queue is a linear data structure where the last position is connected back to the first position to make the queue circular.

29. How does a circular queue differ from a regular queue?
In a circular queue, the rear can wrap around to the front when there is space, while in a regular queue, elements are only added to the end and removed from the front.

30. What is a DEQUE?
A DEQUE (double-ended queue) is a data structure that allows insertion and deletion of elements from both ends.

31. What is a priority queue?
A priority queue is a data structure where each element has a priority, and elements are served according to their priority rather than their order in the queue.

32. What is a linked list?
A linked list is a linear data structure where elements are stored in nodes, and each node points to the next node in the sequence.

33. What are the types of linked lists?

Singly linked list

Doubly linked list

Circular linked list

34. What is a header linked list?
A header linked list has a special node at the beginning (header) that does not contain data, serving as a marker for the start of the list.

35. Name a few applications of linked lists.

Dynamic memory allocation

Implementation of stacks and queues

Adjacency representation of graphs

36. What is a linear search?
Linear search is a search algorithm that checks each element of a list sequentially until the desired element is found or the list ends.

37. What is a binary search?
Binary search is a search algorithm that finds the position of a target value by repeatedly dividing the search interval in half, applicable only on sorted arrays.

38. What is an interpolation search?
Interpolation search is an improved variant of binary search that uses the value of the target to estimate its position in a sorted array.

39. Write the formula for interpolation search?
The position is calculated as:

pos = low + \left(\frac{(x - arr[low]) \times (high - low)}{(arr[high] - arr[low])}\right)

40. Explain selection sort and its time complexity.
Selection sort is an in-place comparison sorting algorithm that divides the input into a sorted and an unsorted region, repeatedly selecting the smallest (or largest) element from the unsorted region to append to the sorted region. Its time complexity is $O(n^2)$.

41. Explain insertion sort and its time complexity.
Insertion sort builds a sorted array one element at a time by comparing and inserting each new element into its correct position in the sorted part. Its time complexity is $O(n^2)$ in the average and worst cases, and $O(n)$ in the best case (when the array is already sorted).

42. Explain bubble sort and its time complexity.

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. Its time complexity is $O(n^2)$.

43. Explain quick sort and its time complexity.
Quick sort is a divide-and-conquer algorithm that selects a 'pivot' element, partitions the array into elements less than and greater than the pivot, and recursively sorts the partitions. Its average time complexity is $O(n \log n)$, and the worst-case is $O(n^2)$.

44. Explain merge sort and its time complexity.
Merge sort is a divide-and-conquer algorithm that divides the array into two halves, recursively sorts each half, and then merges the sorted halves. Its time complexity is $O(n \log n)$.

45. Explain radix sort and its time complexity.
Radix sort is a non-comparative sorting algorithm that sorts numbers by processing individual digits. It has a time complexity of $O(nk)$, where n is the number of elements and k is the number of digits in the maximum number.

46. What is a binary tree?
A binary tree is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child.

47. What are the properties of a binary tree?

Each node has at most two children.

The maximum number of nodes at level is .

The maximum number of nodes in a binary tree of height is .

The minimum height of a binary tree with nodes is .

48. What are the representations of a binary tree?

Array Representation: Uses an array to represent the tree, where for a node at index :

Left child is at index

Right child is at index

Parent is at index

Pointer Representation: Each node contains pointers to its left and right children.

49. What are the types of traversals we have in binary trees?

Preorder Traversal: Visit the root, traverse the left subtree, then traverse the right subtree.

Inorder Traversal: Traverse the left subtree, visit the root, then traverse the right subtree.

Postorder Traversal: Traverse the left subtree, traverse the right subtree, then visit the root.

Level Order Traversal: Visit nodes level by level, starting from the root.

50. Explain few additional operations of a binary tree.

Insertion: Adding a new node while maintaining the binary tree properties.

Deletion: Removing a node and reorganizing the tree to maintain its properties.

Searching: Finding a specific value in the binary tree.

Height Calculation: Determining the maximum depth of the tree.

51. What is a threaded binary tree?
A threaded binary tree is a type of binary tree where null pointers are replaced with pointers to the next in-order predecessor or successor, allowing for efficient in-order traversal without recursion or a stack.

52. Define binary search tree (BST).
A binary search tree is a binary tree in which each node has a value greater than all values in its left subtree and less than all values in its right subtree.

53. Explain about insertion, deletion, traversals, and searching in a BST.

Insertion: Start at the root; compare the value with the current node. If smaller, go left; if larger, go right. Insert when a null pointer is reached.

Deletion: Can have three cases:

1. Node with no children (leaf): Simply remove it.

2. Node with one child: Replace the node with its child.

3. Node with two children: Find the in-order successor (smallest in the right subtree), replace the node's value with it, and delete the successor.

Searching: Start at the root; traverse left or right depending on whether the target value is smaller or larger than the current node until found or a null pointer is reached.
Traversals: Can be performed using preorder, inorder, postorder, or level order methods.

54. Explain about tree evaluation of expressions.
Expression trees are binary trees used to represent expressions. Each internal node is an operator, and each leaf node is an operand. The expression can be evaluated by recursively evaluating the left and right subtrees and applying the operator at the root.

55. Explain the types of rotations in AVL tree balancing.

Single Right Rotation (LL Rotation): Performed when a node is added to the left subtree of the left child.

Single Left Rotation (RR Rotation): Performed when a node is added to the right subtree of the right child.

Left-Right Rotation (LR Rotation): Performed when a node is added to the right subtree of the left child.

Right-Left Rotation (RL Rotation): Performed when a node is added to the left subtree of the right child.

56. Explain the rules to remember during red-black tree balancing.

Every node is either red or black.

The root is always black.

Red nodes cannot have red children (no two reds in a row).

Every path from a node to its descendant null nodes must have the same number of black nodes.

New insertions are always red.

57. What is hashing?

Hashing is a process that converts input (or 'keys') into a fixed-size string of bytes, typically using a hash function. The output, called a hash code, is used to index data in a hash table.

58. What is a hash table?
A hash table is a data structure that implements an associative array, using a hash function to compute an index (hash code) into an array of buckets or slots, from which the desired value can be found.

59. What are static and dynamic hashing?

Static Hashing: The size of the hash table is fixed, and collisions are handled using techniques such as chaining or open addressing.

Dynamic Hashing: The hash table can grow or shrink in size as needed, allowing for efficient memory usage and handling a varying number of entries.

60. Explain how collisions happen.
Collisions occur in a hash table when two different keys hash to the same index. This can happen due to the limited size of the hash table and the nature of the hash function.

61. Explain a few collision handling techniques.

Chaining: Each bucket in the hash table points to a linked list of entries that hash to the same index.

Open Addressing: When a collision occurs, the algorithm searches for the next available slot using a probing sequence (e.g., linear probing, quadratic probing).

Double Hashing: Uses a second hash function to calculate the next index when a collision occurs.

62. Explain various types of connections in graphs.

Directed Connections: Edges have a direction (from one vertex to another).

Undirected Connections: Edges do not have a direction (connect two vertices symmetrically).

Weighted Connections: Edges have weights (costs associated with traversing them).

Unweighted Connections: Edges do not have weights.

63. Explain different types of graphs.

Simple Graph: No loops or multiple edges between the same vertices.

Multigraph: Can have multiple edges between the same vertices.

Complete Graph: Every pair of distinct vertices is connected by a unique edge.

Bipartite Graph: Vertices can be divided into two disjoint sets, with edges only between sets.

Cyclic Graph: Contains at least one cycle.

Acyclic Graph: Contains no cycles (e.g., trees).

64. What are the two convenient representations of graphs?

Adjacency Matrix: A 2D array where each cell indicates the presence or absence of an edge between vertices.

Adjacency List: An array of lists where each index represents a vertex and stores a list of its adjacent vertices.

65. What is BFS (Breadth-First Search)?
BFS is a graph traversal algorithm that explores the neighbor vertices at the present depth prior to moving on to vertices at the next depth level. It uses a queue data structure.

66. What is DFS (Depth-First Search)?
DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It can be implemented using recursion or a stack data Structure.

All the best ...