Q1: Can an array be resized at runtime?
Ans: No, standard arrays in most languages (like C/C++) have fixed size. Dynamic arrays (like ArrayList in Java) can resize at runtime.

Q2: What is Data Structures?
Ans: A data structure is a way to organize and store data efficiently for access and modification.

Q3: What is ADT?
Ans: Abstract Data Type (ADT) defines a data type by its behavior (operations) rather than implementation. Example: Stack, Queue.

Q4: What are some applications of Data structures?
Ans: Applications include database systems, OS memory management, compiler design, network routing, and real-time systems.

Q5: Describe the types of Data Structures.
Ans: Two types:

- **Linear: Array, Linked List, Stack, Queue**
- **Non-linear: Tree, Graph**

Q6: What is a Linear Data Structure? Name a few examples.
Ans: Elements are arranged sequentially. Examples: Array, Linked List, Stack, Queue.

Q7: What is asymptotic notation in data structure?
Ans: It expresses algorithm efficiency as input size grows:

- **Big O (O): Worst-case**
- **Omega (Ω): Best-case**
- **Theta (Θ): Average-case**

Q8: What is time complexity?
Ans: Time taken by an algorithm as a function of input size (n).

Q9: What is space complexity?
Ans: Memory required by an algorithm as a function of input size (n).

Q10: What is the time complexity for accessing an element in an array?
Ans: O(1) for direct access.

Q11: Difference between an array and a linked list.
Ans:

- **Array: fixed size, O(1) access, O(n) insertion/deletion.**
- **Linked List: dynamic size, O(n) access, O(1) insertion/deletion at head.**

Q12: Advantages and disadvantages of arrays.
Ans:

- **Advantages: Fast access, simple.**
- **Disadvantages: Fixed size, costly insertion/deletion.**

Q13: What is a linked list?
Ans: A collection of nodes where each node contains data and a pointer to the next node.

Q14: Different types of linked lists.
Ans: Singly, Doubly, Circular, Circular Doubly Linked List.

Q15: Advantages of Linked List.
Ans: Dynamic size, easy insertion/deletion, efficient memory usage.

Q16: Disadvantages of Linked List.
Ans: Extra memory for pointers, sequential access, more complex than arrays.

Q17: How do you reverse a linked list?
Ans: Iteratively or recursively by reversing node pointers.

Q18: How do you detect a cycle in a linked list?
Ans: Use Floyd's cycle-finding algorithm (slow and fast pointers).

Q19: How do you find the middle element of a linked list?
Ans: Use slow and fast pointers; slow reaches middle when fast reaches end.

Q20: How do you merge two sorted linked lists?
Ans: Compare heads of both lists, attach smaller, move pointer, repeat recursively or iteratively.

Q21: How do you remove duplicates from an unsorted linked list?
Ans: Use a hash set to track seen values and remove duplicates during traversal.

Q22: Time complexity of linked list operations.
Ans: Insertion/deletion at head: O(1), at tail: O(n), Search: O(n).

Q23: Compare singly and doubly linked lists.
Ans:

- **Singly: less memory, forward traversal only.**
- **Doubly: extra memory, forward/backward traversal, easier deletion.**

Q24: Pros and cons of circular linked list.
Ans:

- **Pros: Easy rotation, no null at end.**
- **Cons: Complex implementation, risk of infinite loop.**

Q25: Operations in stack data structure.
Ans: Push, Pop, Peek/Top, isEmpty, isFull.

Q26: What is a queue and its applications?
Ans: FIFO data structure. Applications: printer queues, CPU scheduling, BFS.

Q27: Compare dynamic arrays vs linked lists.
Ans:

- **Dynamic array: O(1) access, costly resizing.**

- **Linked list: O(n) access, easy insertion/deletion.**

Q28: What is a stack?
Ans: LIFO data structure where last inserted element is removed first.

Q29: Operations on stack.
Ans: Push, Pop, Peek/Top, isEmpty, isFull.

Q30: How is a stack implemented in an array?
Ans: Use an array and a top index to manage push/pop operations.

Q31: Time complexity of stack operations.
Ans: Push/Pop/Peek: O(1).

Q32: Applications of stack.
Ans: Expression evaluation, recursion, undo operations, syntax parsing.

Q33: What is stack overflow?
Ans: Occurs when trying to push into a full stack.

Q34: What is stack underflow?
Ans: Occurs when trying to pop from an empty stack.

Q35: What is a postfix expression?
Ans: Operator appears after operands (e.g., AB+).

Q36: How can a stack evaluate a postfix expression?
Ans: Push operands, pop two for operator, push result, repeat.

Q37: What is a prefix expression?
Ans: Operator appears before operands (e.g., +AB).

Q38: How can a stack evaluate a prefix expression?
Ans: Scan right to left, push operands, pop for operator, push result.

Q39: What is a Queue?
Ans: FIFO data structure where first inserted element is removed first.

Q40: Different types of queues.
Ans: Simple queue, circular queue, priority queue, double-ended queue (deque).

Q41: How is a queue implemented in an array?
Ans: Use front and rear indices; increment rear on enqueue, front on dequeue.

Q42: How is a queue implemented in a linked list?
Ans: Use nodes with front and rear pointers.

Q43: Time complexity of enqueue and dequeue.
Ans: O(1) if implemented using linked list, O(1) amortized for circular array.

Q44: Difference between queue and stack.
Ans: Queue: FIFO, Stack: LIFO.

Q45: Applications of queues.
Ans: Job scheduling, BFS, resource management, buffering.

Q46: Handling overflow and underflow in queue.
Ans: Check if queue is full (overflow) or empty (underflow) before operations.

Q47: What is circular and priority queue?
Ans:

- **Circular: Last node points to first, efficient space.**
- **Priority: Elements dequeued by priority, not insertion order.**

Q48: What is a double-ended queue (Deque)?
Ans: Queue where insertion and deletion can occur at both ends.

Q49: How is a deque implemented?
Ans: Using arrays or doubly linked lists with front and rear pointers.

Q50: How do you delete a node from a singly linked list?
Ans: Adjust previous node's next pointer to skip the node, then free it.

Q51: "Implement a function to split a linked list into two halves."
Ans: "Use slow and fast pointers: slow moves 1 step, fast moves 2. When fast reaches the end, slow is at the middle. Split at slow."

Q52: "How would you convert a binary tree to a doubly linked list?"
Ans: "Use in-order traversal recursively, keep track of previous node, link previous.right = current and current.left = previous."

Q53: "Write a function to add two numbers represented by linked lists."
Ans: "Traverse both lists, add node values with carry, create a new list for result, handle remaining carry."

Q54: "How can you optimize a sorting algorithm for nearly sorted data?"
Ans: "Use Insertion Sort or Adaptive algorithms like TimSort which handle nearly sorted data efficiently."

Q55: "How do you implement a stack using an array?"
Ans: "Use an array with a top pointer; push increments top and stores value, pop returns array[top] and decrements top."

Q56: "How do you implement a queue using a linked list?"
Ans: "Maintain front and rear pointers; enqueue adds at rear, dequeue removes from front."

Q57: "How do you detect a cycle in a linked list?"
Ans: "Use Floyd's cycle detection (slow and fast pointers). If slow meets fast, cycle exists."

Q58: "How do you insert a node at the beginning and end of a singly linked list?"
Ans: "Beginning: create node, set its next to head, update head. End: traverse to last, set last.next = new node."

Q59: "What is a heap data structure?"
Ans: "A complete binary tree satisfying heap property: max-heap (parent ≥ children) or min-heap (parent ≤ children)."

Q60: "What are the two types of heaps?"
Ans: "Max-Heap and Min-Heap."

Q61: "What is a tree data structure?"
Ans: "A hierarchical structure with nodes, one root, children nodes, no cycles."

Q62: "What is a B-tree, and how is it different from a binary search tree?"
Ans: "B-tree is a balanced m-way search tree optimized for disk storage; BST is binary and may be unbalanced."

Q63: "Explain how a B+ tree works and its advantages over a B-tree."
Ans: "B+ tree stores keys in internal nodes and all data in leaf nodes; leaf nodes are linked for fast range queries. Advantage: efficient disk reads and range queries."

Q64: "Compare the time and space complexities of sorting using a BST, AVL Tree."
Ans: "BST avg: O(n log n), worst $O(n^2)$. AVL: always O(n log n). Space: O(n) for both."

Q65: "What is Inorder, Preorder, and Postorder traversal?"
Ans: "Inorder: left-root-right. Preorder: root-left-right. Postorder: left-right-root."

Q66: "What is the impact of tree height on the performance of sorting algorithms using trees?"
Ans: "Greater height increases search/insertion/deletion time; balanced trees improve efficiency."

Q67: "How does balancing affect the efficiency of sorting in tree-based structures?"
Ans: "Balancing keeps height minimal (O(log n)), reducing time for insert/search/delete operations."

Q68: "Implement an AVL tree and explain how it maintains balance during insertions and deletions."
Ans: "AVL uses rotations (single/double) to maintain balance factor (-1,0,1) after insert/delete."

Q69: "What is the time complexity of inserting an element into a heap?"
Ans: "O(log n) due to heapify after insertion."

Q70: "List the types of trees."
Ans: "Binary Tree, Binary Search Tree, AVL Tree, Red-Black Tree, B-tree, B+ tree, Heap, Trie."

Q71: "Write an algo sum of right node of a Binary tree."
Ans: "Traverse tree, if node.right exists, add node.right.data + sum(right subtree)."

Q72: "Implementation of binary tree."
Ans: "Use Node class with data, left, right. Root points to first node. Traverse with recursion or queue for BFS."

Q73: "Calculate sum of all right leaves of a BST."
Ans: "Traverse BST; if node.right exists and is leaf, add value. Recurse on left and right children."

Q74: "What is binary tree data structure?"
Ans: "A tree where each node has at most two children: left and right."

Q75: "What are the applications for binary search trees?"
Ans: "Searching, sorting, priority queues, associative arrays, in-memory databases."

Q76: "What are tree traversals?"
Ans: "Methods to visit all nodes: Inorder, Preorder, Postorder, Level-order."

Q77: "What is the difference between BFS and DFS?"
Ans: "BFS: level by level using queue. DFS: depth first using recursion/stack."

Q78: "What is a Tree?"
Ans: "Hierarchical structure with root node, children nodes, no cycles."

Q79: "Explain different types of trees."
Ans: "Binary, BST, AVL, Red-Black, B-tree, B+ tree, Heap, Trie, N-ary."

Q80: "What are the basic operations performed on a tree?"
Ans: "Insertion, Deletion, Traversal, Searching."

Q81: "What are the different ways to represent a tree in memory?"
Ans: "Linked representation (nodes with pointers), Array representation (for complete binary trees)."

Q82: "What are the advantages and disadvantages of using trees?"
Ans: "Advantage: hierarchical, fast search. Disadvantage: memory overhead, complex operations."

Q83: "What are Binary trees?"
Ans: "Tree with at most two children per node: left and right."

Q84: "How to add element in Binary search tree?"
Ans: "Compare with root, go left if smaller, right if larger; insert at null position."

Q85: "How to delete an element in Binary search tree?"
Ans: "Three cases: leaf node (delete), one child (replace with child), two children (replace with inorder successor)."

Q86: "Difference between Binary tree and Binary search tree?"
Ans: "Binary Tree: no order. BST: left < root < right."

Q87: "Construct the given BST and get sum of all leaf nodes."
Ans: "Insert nodes per BST rules; sum nodes with no children."

Q88: "Write code to add node to left of a node in binary tree."
Ans: "Create new node; set new.left = parent.left; parent.left = new."

Q89: "What is B and B+ tree?"
Ans: "B-tree: balanced m-way tree, keys in all nodes. B+ tree: keys in internal nodes, data in leaves, leaves linked."

Q90: "Differences between B-tree and B+ tree?"
Ans: "B-tree: data + keys in all nodes. B+ tree: data only in leaves; internal nodes store keys for indexing."

Q91: "Advantages of binary search over linear search?"
Ans: "O(log n) vs O(n); efficient for large sorted datasets."

Q92: "Explain the concept of a binary search tree."
Ans: "BST maintains left < root < right property; enables efficient search, insertion, deletion."

Q93: "How can you convert a BST into a sorted array?"
Ans: "Inorder traversal produces elements in sorted order; store in array."

Q94: "What is AVL tree data structure, its operations, and its rotations?"
Ans: "AVL: self-balancing BST. Operations: insert/delete/search. Rotations: LL, RR, LR, RL to maintain balance factor."

Q95: "Applications for AVL trees?"
Ans: "Database indexing, memory management, maintaining sorted datasets efficiently."

Q96: "Which sorting algorithm is considered the fastest? Why?"
Ans: "QuickSort on average; O(n log n), cache friendly; but worst-case $O(n^2)$."

Q97: "How does Selection sort work?"
Ans: "Repeatedly find the minimum element from unsorted part, swap with first unsorted element."

Q98: "Examples of divide and conquer algorithms?"
Ans: "Merge Sort, QuickSort, Binary Search, Strassen's matrix multiplication."

Q99: "What is sorting?"
Ans: "Arranging data in ascending or descending order."

Q100: "Difference between comparison-based and non-comparison-based sorting?"
Ans: "Comparison-based: sort by comparing elements (QuickSort, MergeSort). Non-comparison: use key properties (Counting Sort, Radix Sort)."

101. Explain how Bubble Sort works. [ICRA ANALYTICS]
Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.

Algorithm:

1. **Start from the first element of the array.**
2. **Compare the current element with the next element.**
3. **If the current element is greater than the next, swap them.**
4. **Move to the next element and repeat.**
5. **Repeat the entire process for n-1 passes (where n is array size).**

Time Complexity:

- **Best case (already sorted): O(n)**
- **Worst case (reverse sorted): $O(n^2)$**
- **Average case: $O(n^2)$**

Example: [5, 2, 9, 1] → Pass 1: [2,5,1,9] → Pass 2: [2,1,5,9] → Pass 3: [1,2,5,9]

102. Write algo merge sort. [TCS, Accenture]

Merge Sort is a divide-and-conquer algorithm. It divides the array into halves, recursively sorts them, and merges the sorted halves.

Algorithm:

1. **If array has 1 element, return.**
2. **Divide array into left and right halves.**
3. **Recursively apply merge sort on left and right halves.**
4. **Merge the two sorted halves.**

Pseudocode:

```
MergeSort(arr, l, r):

  if l >= r:

    return

  mid = (l + r) / 2

  MergeSort(arr, l, mid)

  MergeSort(arr, mid+1, r)

  Merge(arr, l, mid, r)
```

Time Complexity: O(n log n)
Space Complexity: O(n)

103. Time complexity of Quick Sort in best, average, and worst case

- **Best case: O(n log n) → When the pivot divides the array into two equal halves.**
- **Average case: O(n log n) → Random distribution of pivot values.**
- **Worst case: O(n²) → When pivot is always the smallest or largest element (sorted or reverse sorted array).**

104. How does Merge Sort work, and its time complexity?

Merge Sort splits the array into halves recursively until each subarray has one element, then merges the subarrays in sorted order.

Time Complexity: O(n log n)
Space Complexity: O(n)
Example: [4,3,1,2] → Split: [4,3] & [1,2] → Merge: [3,4] & [1,2] → [1,2,3,4]

105. Main advantage of Heap Sort over other sorting algorithms

- **Heap Sort has O(n log n) worst-case time complexity, unlike Quick Sort which can degrade to O(n²).**
- **It sorts in-place using a binary heap, so no extra array is needed like Merge Sort.**
- **Good for memory-constrained systems.**

106. Compare the time and space complexities of different sorting algorithms

Algorithm	Best Case	Average Case	Worst Case	Space Complexity	Stable?
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	No
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	No
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	No

107. How does the choice of pivot affect Quick Sort?

- **Choosing a good pivot (middle element or median) ensures balanced partitions → O(n log n).**
- **Choosing first or last element in a sorted/reverse array → unbalanced partitions → $O(n^2)$.**
- **Randomized pivoting reduces the chance of worst-case performance.**

108. How to sort a linked list

- **Use Merge Sort (recommended) because it works efficiently with linked lists.**
- **Steps:**
  1. **Find the middle node of the linked list using slow/fast pointers.**
  2. **Split the list into two halves.**
  3. **Recursively sort both halves.**
  4. **Merge the two sorted halves.**

Time Complexity: O(n log n)
Space Complexity: O(log n) for recursion

109. Impact of sorting on cache performance and memory usage

- **Cache Performance: Sequential memory access (like in Merge Sort on arrays) improves cache hits. Random access (like Quick Sort on linked lists) reduces cache efficiency.**
- **Memory Usage: Algorithms like Merge Sort require extra memory for merging arrays. In-place algorithms like Quick Sort and Heap Sort are memory-efficient.**

110. Explain how a binary tree is constructed. [TCS, Blue Flame Labs]

- **A binary tree is constructed by defining a Node structure containing data, left, and right pointers.**
- **Nodes are inserted based on the type of tree (Binary Search Tree: smaller values to left, larger to right).**
- **Example in C++:**

```
struct Node {

  int data;
```

```
    Node* left;
    Node* right;
};
Node* insert(Node* root, int value) {
    if (!root) return new Node{value, nullptr, nullptr};
    if (value < root->data) root->left = insert(root->left, value);
    else root->right = insert(root->right, value);
    return root;
}
```

111. Function to find shortest path distance between two nodes in a complete binary tree

- **Use level order traversal (BFS).**
- **For nodes u and v, find levels of both nodes, then compute distance using:**

distance = level(u) + level(v) - 2 * level(LCA(u,v))

- **LCA(u,v) = Lowest Common Ancestor.**

112. Write algo of linear search

Algorithm:

1. **Start from index 0.**
2. **Compare each element with target value.**
3. **If found, return index.**
4. **If end of array reached, return -1.**

Time Complexity: O(n)
Space Complexity: O(1)

113. What is a binary search algorithm? [TCS, CTS]

Binary Search works on a sorted array by repeatedly dividing the search interval in half.

Algorithm:

1. **Set low = 0 and high = n-1.**
2. **Find mid = (low + high)/2.**
3. **If arr[mid] == target, return mid.**
4. **If arr[mid] < target, search right half.**
5. **If arr[mid] > target, search left half.**
6. **Repeat until found or low > high.**

Time Complexity: O(log n)
Space Complexity: O(1)

114. How to perform depth-first search (DFS) on a graph

- **DFS explores as far as possible along each branch before backtracking.**
- **Can be implemented using stack (iterative) or recursion.**

Algorithm (Recursive):

```
DFS(node):

  mark node as visited

  for each neighbor of node:

    if neighbor not visited:

      DFS(neighbor)
```

115. How to find length of a string without using strlen()

Algorithm:

1. **Initialize count = 0.**
2. **Traverse string until null character '\0'.**
3. **Increment count for each character.**
4. **Return count.**

```
int length(char str[]) {

  int count = 0;

  while(str[count] != '\0') count++;

  return count;

}
```

Time Complexity: O(n)

116. Write algo of quicksort? [TCS, IBM]

Quick Sort Algorithm:

1. **Choose a pivot element.**
2. **Partition the array such that elements < pivot are on left, > pivot are on right.**
3. **Recursively apply Quick Sort on left and right partitions.**

Pseudocode:

```
QuickSort(arr, low, high):

  if low < high:

    pi = partition(arr, low, high)

    QuickSort(arr, low, pi-1)
```

QuickSort(arr, pi+1, high)

Time Complexity:

- **Best/Average: O(n log n)**
- **Worst: O(n²)**