1. What is a Data Structure?

A data structure is a way of organizing and storing data so that it can be accessed and modified efficiently.
Example: Arrays, Linked Lists, Stacks, Queues.

---

2. Classify Data Types.

- **Primitive:** int, float, char, double
- **Non-Primitive:** Arrays, Structures, Unions, Linked Lists, Trees, Graphs

---

3. Common Operations on Data Structures:

Insertion, Deletion, Traversal, Searching, Sorting, and Updating.

---

4. Define Structure.

A structure is a user-defined data type that groups different data types under one name.

struct Student { int id; char name[20]; float marks; };

---

5. How can you access elements in an array?

Using indices starting from 0.
Example: a[0], a[1], a[2]...

---

6. What is a Self-Referential Structure?

A structure that contains a pointer to another structure of the same type.

struct Node { int data; struct Node *next; };

---

7. Difference Between Structure and Union:

In a **structure**, all members have separate memory.
In a **union**, all members share the same memory location.

8. What is a Pointer?

A pointer is a variable that stores the address of another variable.
Example: int *p, a=5; p = &a;

9. Syntax to Access Structure Members:

Using **dot (.)** or **arrow (->)** operator.
Example:

```
s1.name;   // Direct
```

```
ptr->name; // Using pointer
```

10. What is DMA and How It Differs from SMA?

DMA (Dynamic Memory Allocation) allocates memory at runtime, while **SMA (Static Memory Allocation)** allocates memory at compile time.

11. Functions in DMA:

malloc(), calloc(), realloc(), free()

12. What is a Dynamically Allocated Array?

An array whose memory size is decided during runtime using malloc() or calloc().

13. Steps in Performance Analysis:

1. Identify input size
2. Count operations
3. Find time & space requirements
4. Express complexity (Big O)

14. What is Space Complexity?

It is the amount of memory required by a program during execution.

---

15. What is Time Complexity?

It is the total time taken by a program as a function of input size.

Example Codes (Unit 1):

Dynamic Array Example

```
int *arr = (int*)malloc(n * sizeof(int));

for(int i=0;i<n;i++) scanf("%d",&arr[i]);

free(arr);
```

🧱 UNIT 2 – STACKS, RECURSION & QUEUES

---

16. What is a Stack?

A stack is a linear data structure that follows **LIFO (Last In First Out)** principle.

---

17. What is a Queue?

A queue is a linear data structure that follows **FIFO (First In First Out)** principle.

---

18. Representations of Stack:

1. Array representation
2. Linked list representation

---

19. Stack Operations:

- **Push:** Add element to top

- **Pop:** Remove element from top
- **Peek:** Get top element

---

20. Queue Operations:

- **Enqueue:** Add at rear
- **Dequeue:** Remove from front
- **Display:** Show all elements

---

21. Stack Status (Array):

- **Empty:** top == -1
- **Full:** top == MAX - 1

---

22. Stack Status (Linked List):

- **Empty:** top == NULL
- **Full:** When no memory (malloc fails)

---

23. Queue Status (Array):

- **Empty:** front == -1
- **Full:** rear == MAX - 1

---

24. Queue Status (Linked List):

- **Empty:** front == NULL
- **Full:** No memory available

---

25. What is Polish Notation?

Prefix form – operator comes before operands.
Example: +AB

---

26. What is Reverse Polish Notation?

Postfix form – operator comes after operands.
Example: AB+

---

27. Types of Recursion:

Direct, Indirect, Tail, and Nested Recursion.

---

28. What is a Circular Queue?

A queue where the last position connects back to the first to form a circle.

---

29. Difference Between Regular and Circular Queue:

In a circular queue, memory is reused using modulo operation.

---

30. What is a Deque?

(Double Ended Queue) – Insertion and deletion can happen from both ends.

---

31. What is a Priority Queue?

Each element has a priority; highest priority element is served first.

Example Codes (Unit 2):

Stack using Array

```
void push(int val){ if(top==MAX-1) printf("Overflow"); else stack[++top]=val; }
```

```
int pop(){ return (top==-1)?-1:stack[top--]; }
```

Circular Queue Condition:
rear = (rear + 1) % MAX

🔗 UNIT 3 – LINKED LISTS

---

32. What is a Linked List?

A linear collection of nodes connected by pointers.

---

33. Types of Linked Lists:

- Singly
- Doubly
- Circular
- Header Linked List

---

34. What is a Header Linked List?

It contains a special header node at the beginning storing list information.

---

35. Applications of Linked Lists:

Used in stacks, queues, polynomial manipulation, memory management.

Example Codes:

Node Definition

```
struct Node { int data; struct Node *next; };
```

Insertion at Beginning

```
newNode->next = head; head = newNode;
```

🔍 UNIT 4 – SEARCHING AND SORTING

36. What is Linear Search?

Searches each element one by one — O(n) time.

37. What is Binary Search?

Searches in a sorted list using divide and conquer — O(log n).

38. What is Interpolation Search?

Improved binary search based on value position — O(log log n) average.

39. Formula for Interpolation Search:

pos = low + ((key - arr[low]) * (high - low)) / (arr[high] - arr[low]);

40–45. Sorting Algorithms & Time Complexities:

Sort	Logic	Time Complexity
Selection	Find min & swap	$O(n^2)$
Insertion	Insert at correct place	$O(n^2)$
Bubble	Compare & swap	$O(n^2)$
Quick	Partition & recurse	O(n log n)
Merge	Divide & merge	O(n log n)
Radix	Based on digits	O(nk)

🌳 UNIT 5 – TREES

46. What is a Binary Tree?

A tree where each node has at most two children.

---

47. Properties of Binary Tree:

Max nodes = 2^h - 1, height = $\log_2(n+1)$

---

48. Representations of Binary Tree:

1. Array representation
2. Linked list representation

---

49. Tree Traversals:

- Inorder (LNR)
- Preorder (NLR)
- Postorder (LRN)

---

50. Additional Operations:

Copy tree, count nodes, find height, mirror tree.

---

51. Threaded Binary Tree:

Pointers to in-order successor/predecessor replace NULL links.

---

52. Binary Search Tree (BST):

Binary tree where left < root < right.

---

53. BST Operations:

Insertion, Deletion, Searching, Traversal.

---

54. Tree Evaluation of Expression:

Binary expression tree used to evaluate postfix/infix expressions.

🌲 UNIT 6 – AVL & RED-BLACK TREES

---

55. AVL Rotations:

- Left Rotation (LL)
- Right Rotation (RR)
- Left-Right (LR)
- Right-Left (RL)

---

56. Red-Black Tree Rules:

1. Root is black
2. No two consecutive reds
3. Every path has equal black nodes
4. New node is red

💠 UNIT 7 – HASHING

---

57. What is Hashing?

A technique to map keys to positions in a table using a hash function.

---

58. What is a Hash Table?

An array that stores data using hash indices.

---

59. (Duplicate) – same as Q58.

---

60. Static vs Dynamic Hashing:

Static uses fixed table size; Dynamic grows/shrinks with data.

---

61. How Does Collision Occur?

When two keys hash to the same index.

---

62. Collision Handling Techniques:

1. Chaining
2. Linear Probing
3. Quadratic Probing
4. Double Hashing

🌐 UNIT 8 – GRAPHS

---

63. Types of Graph Connections:

Simple, Multigraph, Complete, Connected, Disconnected.

---

64. Types of Graphs:

Directed, Undirected, Weighted, Unweighted.

---

65. Graph Representations:

1. Adjacency Matrix
2. Adjacency List

---

66. BFS (Breadth First Search):

Uses **Queue**, visits neighbors level by level.

---

67. DFS (Depth First Search):

Uses **Stack/Recursion**, visits depth-wise.

1. What is a Data Structure?

A data structure is a way of organizing and storing data so that it can be accessed and modified efficiently.
Example: Arrays, Linked Lists, Stacks, Queues.

---

2. Classify Data Types.

- **Primitive:** int, float, char, double
- **Non-Primitive:** Arrays, Structures, Unions, Linked Lists, Trees, Graphs

---

3. Common Operations on Data Structures:

Insertion, Deletion, Traversal, Searching, Sorting, and Updating.

---

4. Define Structure.

A structure is a user-defined data type that groups different data types under one name.

struct Student { int id; char name[20]; float marks; };

---

5. How can you access elements in an array?

Using indices starting from 0.
Example: a[0], a[1], a[2]...

---

6. What is a Self-Referential Structure?

A structure that contains a pointer to another structure of the same type.

struct Node { int data; struct Node *next; };

---

7. Difference Between Structure and Union:

In a **structure**, all members have separate memory.
In a **union**, all members share the same memory location.

---

8. What is a Pointer?

A pointer is a variable that stores the address of another variable.
Example: int *p, a=5; p = &a;

---

9. Syntax to Access Structure Members:

Using **dot (.)** or **arrow (->)** operator.
Example:

s1.name;   // Direct

ptr->name; // Using pointer

---

10. What is DMA and How It Differs from SMA?

DMA (Dynamic Memory Allocation) allocates memory at runtime, while **SMA (Static Memory Allocation)** allocates memory at compile time.

---

11. Functions in DMA:

malloc(), calloc(), realloc(), free()

---

12. What is a Dynamically Allocated Array?

An array whose memory size is decided during runtime using malloc() or calloc().

---

13. Steps in Performance Analysis:

1. Identify input size
2. Count operations
3. Find time & space requirements
4. Express complexity (Big O)

---

14. What is Space Complexity?

It is the amount of memory required by a program during execution.

---

15. What is Time Complexity?

It is the total time taken by a program as a function of input size.

Example Codes (Unit 1):

Dynamic Array Example

```
int *arr = (int*)malloc(n * sizeof(int));

for(int i=0;i<n;i++) scanf("%d",&arr[i]);

free(arr);
```

🧱 UNIT 2 – STACKS, RECURSION & QUEUES

---

16. What is a Stack?

A stack is a linear data structure that follows **LIFO (Last In First Out)** principle.

---

17. What is a Queue?

A queue is a linear data structure that follows **FIFO (First In First Out)** principle.

---

18. Representations of Stack:

1. Array representation
2. Linked list representation

---

19. Stack Operations:

- **Push:** Add element to top
- **Pop:** Remove element from top
- **Peek:** Get top element

---

20. Queue Operations:

- **Enqueue:** Add at rear
- **Dequeue:** Remove from front
- **Display:** Show all elements

---

21. Stack Status (Array):

- **Empty:** top == -1
- **Full:** top == MAX - 1

---

22. Stack Status (Linked List):

- **Empty:** top == NULL
- **Full:** When no memory (malloc fails)

---

23. Queue Status (Array):

- **Empty:** front == -1
- **Full:** rear == MAX - 1

---

24. Queue Status (Linked List):

- **Empty:** front == NULL
- **Full:** No memory available

---

25. What is Polish Notation?

Prefix form – operator comes before operands.
Example: +AB

---

26. What is Reverse Polish Notation?

Postfix form – operator comes after operands.
Example: AB+

---

27. Types of Recursion:

Direct, Indirect, Tail, and Nested Recursion.

---

28. What is a Circular Queue?

A queue where the last position connects back to the first to form a circle.

---

29. Difference Between Regular and Circular Queue:

In a circular queue, memory is reused using modulo operation.

---

30. What is a Deque?

(Double Ended Queue) – Insertion and deletion can happen from both ends.

---

31. What is a Priority Queue?

Each element has a priority; highest priority element is served first.

Example Codes (Unit 2):

Stack using Array

```
void push(int val){ if(top==MAX-1) printf("Overflow"); else stack[++top]=val; }
```

```
int pop(){ return (top==-1)?-1:stack[top--]; }
```

Circular Queue Condition:
```
rear = (rear + 1) % MAX
```

🔗 UNIT 3 – LINKED LISTS

---

32. What is a Linked List?

A linear collection of nodes connected by pointers.

---

33. Types of Linked Lists:

- Singly
- Doubly
- Circular
- Header Linked List

---

34. What is a Header Linked List?

It contains a special header node at the beginning storing list information.

---

35. Applications of Linked Lists:

Used in stacks, queues, polynomial manipulation, memory management.

Example Codes:

Node Definition

```
struct Node { int data; struct Node *next; };
```

Insertion at Beginning

newNode->next = head; head = newNode;

🔍 UNIT 4 – SEARCHING AND SORTING

---

36. What is Linear Search?

Searches each element one by one — O(n) time.

---

37. What is Binary Search?

Searches in a sorted list using divide and conquer — O(log n).

---

38. What is Interpolation Search?

Improved binary search based on value position — O(log log n) average.

---

39. Formula for Interpolation Search:

pos = low + ((key - arr[low]) * (high - low)) / (arr[high] - arr[low]);

40–45. Sorting Algorithms & Time Complexities:

Sort	Logic	Time Complexity
Selection	Find min & swap	O(n^2)
Insertion	Insert at correct place	O(n^2)
Bubble	Compare & swap	O(n^2)
Quick	Partition & recurse	O(n log n)
Merge	Divide & merge	O(n log n)

Sort	Logic	Time Complexity
Radix	Based on digits	O(nk)

🌳 UNIT 5 – TREES

---

46. What is a Binary Tree?

A tree where each node has at most two children.

---

47. Properties of Binary Tree:

Max nodes = 2^h - 1, height = $\log_2(n+1)$

---

48. Representations of Binary Tree:

1. Array representation
2. Linked list representation

---

49. Tree Traversals:

- Inorder (LNR)
- Preorder (NLR)
- Postorder (LRN)

---

50. Additional Operations:

Copy tree, count nodes, find height, mirror tree.

---

51. Threaded Binary Tree:

Pointers to in-order successor/predecessor replace NULL links.

---

52. Binary Search Tree (BST):

Binary tree where left < root < right.

---

53. BST Operations:

Insertion, Deletion, Searching, Traversal.

---

54. Tree Evaluation of Expression:

Binary expression tree used to evaluate postfix/infix expressions.

🌲 UNIT 6 – AVL & RED-BLACK TREES

---

55. AVL Rotations:

- Left Rotation (LL)
- Right Rotation (RR)
- Left-Right (LR)
- Right-Left (RL)

---

56. Red-Black Tree Rules:

1. Root is black
2. No two consecutive reds
3. Every path has equal black nodes
4. New node is red

🔹 UNIT 7 – HASHING

---

57. What is Hashing?

A technique to map keys to positions in a table using a hash function.

---

58. What is a Hash Table?

An array that stores data using hash indices.

---

59. (Duplicate) – same as Q58.

---

60. Static vs Dynamic Hashing:

Static uses fixed table size; Dynamic grows/shrinks with data.

---

61. How Does Collision Occur?

When two keys hash to the same index.

---

62. Collision Handling Techniques:

1. Chaining
2. Linear Probing
3. Quadratic Probing
4. Double Hashing

🌐 UNIT 8 – GRAPHS

---

63. Types of Graph Connections:

Simple, Multigraph, Complete, Connected, Disconnected.

---

64. Types of Graphs:

Directed, Undirected, Weighted, Unweighted.

---

65. Graph Representations:

1. Adjacency Matrix
2. Adjacency List

---

66. BFS (Breadth First Search):

Uses **Queue**, visits neighbors level by level.

---

67. DFS (Depth First Search):

Uses **Stack/Recursion**, visits depth-wise.