Data Structures

Data structures organize data so operations can be efficient.

Complexity basics

Notation	Meaning	Example
O(1)	constant time	array index access
O(log n)	logarithmic	binary search
O(n)	linear	linear search
O(n log n)	efficient sorting	merge sort
$O(n^2)$	nested loops	bubble sort

Array

Array stores elements in contiguous memory.

Advantages:

- O(1) random access using index.
- Easy traversal.

Disadvantages:

- Fixed size in many languages.
- Insertion/deletion in middle costs O(n).

Common MCQ:

> Time complexity of accessing `arr[i]` ?

Answer: O(1).

Linked List

Linked list stores nodes. Each node contains data and pointer/reference to next node.

Types:

- Singly linked list
- Doubly linked list

- Circular linked list

Advantages:

- Dynamic size.
- Insertion/deletion is easy if node pointer is known.

Disadvantages:

- No random access.
- Search is O(n).
- Extra memory for pointers.

Stack

Stack follows LIFO: Last In, First Out.

Operations:

- push
- pop
- peek/top

Applications:

- recursion call stack,
- undo operation,
- balanced parentheses,
- expression evaluation,
- syntax parsing.

MCQ trap:

> Which data structure is used for function calls?

Answer: Stack.

Queue

Queue follows FIFO: First In, First Out.

Operations:

- enqueue
- dequeue
- front
- rear

Applications:

- CPU scheduling,
- printer queue,
- BFS traversal,
- ticket distribution.

Circular Queue

Circular queue connects last position back to first. It avoids wastage of array space after dequeues.

Condition examples:

- Full: `(rear + 1) % size == front`
- Empty: `front == -1` or implementation-specific

Deque

Double-ended queue allows insertion and deletion from both front and rear.

Priority Queue

Elements are removed by priority, not arrival order. Usually implemented using heap.

Tree

A tree is hierarchical.

Important terms:

- root: top node
- parent/child
- leaf: no child
- height: longest path to leaf
- degree: number of children

Binary Tree

Each node has at most two children.

Maximum nodes at level `l` if root is level 0:

```
2^l
```

Maximum nodes in binary tree of height `h`:

```
2^(h+1) - 1
```

Binary Search Tree

For each node:

- left subtree values are smaller,
- right subtree values are larger.

Average search: O(log n) if balanced.
Worst search: O(n) if skewed.

Heap

Complete binary tree.

- Max heap: parent >= children
- Min heap: parent <= children

Used in priority queues and heap sort.

Graph

Graph has vertices and edges.

Types:

- directed/undirected,
- weighted/unweighted,
- cyclic/acyclic,
- connected/disconnected.

Representations:

Representation	Space	Best for
Adjacency matrix	$O(V^2)$	dense graph
Adjacency list	O(V+E)	sparse graph

BFS and DFS

Algorithm	Uses	Applications
BFS	Queue	shortest path in unweighted graph
DFS	Stack/recursion	cycle detection, topological sort

Hashing

Hashing maps key to index using a hash function.

Average operations:

- insert: O(1)
- search: O(1)
- delete: O(1)

Collision handling:

- chaining,
- open addressing,
- linear probing,
- quadratic probing.

Must-remember table

Operation	Array	Linked List	Stack	Queue
Access by index	O(1)	O(n)	not direct	not direct
Search	O(n)	O(n)	O(n)	O(n)
Insert at end	O(1) amortized	O(1) with tail	push O(1)	enqueue O(1)
Delete from front	O(n)	O(1)	pop top only	dequeue O(1)