Data Structures

Short notes for B.Tech CSE / IT · BCA · MCA · GATE CS

Uncodemy Handwritten Notes · free for students

1. Introduction

- **Data structure** = a way to store and organise data so it can be used efficiently.
- **Linear**: elements in a sequence (array, linked list, stack, queue).
- **Non-linear**: elements in a hierarchy or network (tree, graph).
- **ADT (Abstract Data Type)** = defines *what* operations do, not *how*. Example: Stack ADT has push, pop, peek.

➤ Asymptotic notation (short)

Notation	Meaning
O(f(n))	Upper bound (worst case growth)
Ω(f(n))	Lower bound
Θ(f(n))	Tight bound (both)

- Growth order: 1 < log n < n < n log n < n^2 < n^3 < 2^n < n!

2. Arrays and Linked Lists

➤ Array

- Contiguous memory, fixed size, O(1) random access by index.
- Insert / delete in the middle needs shifting: O(n).

➤ Linked list

- Each node has data and a pointer to the next node. Size can grow at run time.

```
struct node { int data; struct node *next; };
// head -> [10|*] -> [20|*] -> [30|NULL]
```

- Types: **singly**, **doubly** (prev and next pointers), **circular** (last node points to first).

Operation	Array	Singly linked list
Access k-th element	O(1)	O(n)
Insert at beginning	O(n)	O(1)
Insert at end	O(1) if space	O(n), O(1) with tail pointer
Delete given node	O(n)	O(n) to find previous node
Memory	No extra	Extra pointer per node

➤ Insert at beginning (singly)

```
newnode->data = x;
newnode->next = head;
head = newnode;
```

TIP To reverse a singly linked list use three pointers prev, curr, next. Time O(n), space O(1).

3. Stack

- **Stack** = LIFO (Last In First Out). Operations: `push`, `pop`, `peek`, all O(1).
- Array stack: `top = -1` means empty. Overflow when `top == MAX - 1`.
- Uses: function calls, recursion, undo, expression conversion and evaluation, balanced brackets.

➤ Infix to postfix (rules)

1. Operand: write it to output.
2. `(`: push.
3. `)`: pop and output until `(` is found.
4. Operator: pop operators with higher or equal precedence (for left-assoc), then push.
5. At the end, pop everything.

EXAMPLE A + B * C - D becomes A B C * + D -. Prefix of the same: - + A * B C D.

➤ Postfix evaluation

- Scan left to right. Operand: push. Operator: pop two (right first), apply, push result.
- `5 3 2 * + 4 -` gives 5 + 6 = 11, then 11 − 4 = 7.

REMEMBER Number of different orderings when n distinct items are pushed in order and popped at any time = Catalan number C(2n, n) / (n + 1).

4. Queue

- **Queue** = FIFO. Insert at **rear** (enqueue), delete from **front** (dequeue).
- Simple array queue wastes space at the front. **Circular queue** fixes this.

FORMULA Circular queue: rear = (rear + 1) % N. Full when (rear + 1) % N front. Empty when front -1 (or front == rear, depending on the method).

Type	Idea
Circular queue	Last position connects to first
Deque	Insert / delete at both ends
Priority queue	Element with highest priority leaves first (usually a heap)
Input-restricted deque	Insert at one end only
Output-restricted deque	Delete at one end only

EXAM POINT A queue can be made with two stacks. Enqueue O(1); dequeue O(1) amortised.

5. Trees

- **Tree** = hierarchical structure with a root and child nodes, no cycles.
- Terms: root, parent, child, leaf (no child), height (longest path root to leaf), depth, degree.
- **Binary tree**: each node has at most 2 children.

➤ Important facts (height h, root at level 0)

- Max nodes at level i = 2^i. Max nodes in a binary tree of height h = 2^(h+1) − 1.
- In any binary tree: leaves = (nodes with 2 children) + 1, i.e. **n0 = n2 + 1**.
- Minimum height with n nodes = ⌈log2(n + 1)⌉ − 1.

FORMULA Number of distinct unlabelled binary trees with n nodes = C(2n, n) / (n + 1). For n = 3 it is 20 / 4 = 5.

Traversals

Traversal	Order
Preorder	Root, Left, Right
Inorder	Left, Root, Right
Postorder	Left, Right, Root
Level order	Level by level using a queue



- Preorder: A B D E C F. Inorder: D B E A C F. Postorder: D E B F C A. Level: A B C D E F.

EXAM POINT Inorder + Preorder (or Inorder + Postorder) gives a unique binary tree. Preorder + Postorder alone does not (unless the tree is full).

Binary Search Tree (BST)

- Left subtree < node < right subtree. Inorder of a BST is sorted.
- Search / insert / delete: O(h). Best O(log n), worst O(n) (skewed tree).
- Delete a node with two children: replace it with its **inorder successor** (smallest in right subtree) or predecessor.

AVL tree

- Self-balancing BST. **Balance factor** = height(left) − height(right) must be −1, 0 or +1.
- Rotations: LL (right rotate), RR (left rotate), LR (left then right), RL (right then left).
- All operations O(log n).

FORMULA Minimum nodes in an AVL tree of height h: $N(h) = N(h-1) + N(h-2) + 1$, with $N(0) = 1$, $N(1) = 2$. So $N(2) = 4$, $N(3) = 7$.

B-tree and B+ tree (order m)

- Multi-way balanced search trees used in databases and file systems.
- Every node has at most m children; non-root nodes at least $\lceil m/2 \rceil$ children.

- **B+ tree**: all data in leaves, leaves are linked. Good for range queries.

6. Heap

- **Heap** = complete binary tree with heap property. **Max-heap**: parent >= children. **Min-heap**: parent <= children.
- Stored in an array. With index starting at 1: parent = $\lfloor i/2 \rfloor$, left = $2i$, right = $2i + 1$.

Operation	Time
Find max (max-heap)	O(1)
Insert	O(log n)
Delete max	O(log n)
Build heap (bottom-up)	O(n)
Heap sort	O(n log n)

EXAMPLE Insert 10, 20, 15, 30 into an empty max-heap. After 30 bubbles up: array = [30, 20, 15, 10].

7. Graphs

- **Graph** $G = (V, E)$. Directed or undirected, weighted or unweighted.
- Max edges: undirected = $n(n - 1)/2$, directed = $n(n - 1)$.
- Sum of degrees in an undirected graph = 2 x number of edges.

➤ Representation

Method	Space	Check edge (u, v)
Adjacency matrix	O(V^2)	O(1)
Adjacency list	O(V + E)	O(degree of u)

➤ Traversal

BFS	DFS
Uses a queue	Uses a stack / recursion
Level by level	Goes deep first
Shortest path in unweighted graph	Cycle detection, topological sort
O(V + E) with list	O(V + E) with list

TIP Spanning tree, shortest path (Dijkstra) and MST (Prim, Kruskal) are covered in DAA notes too. Learn both.

8. Hashing

- **Hashing** maps a key to a table index using a hash function, e.g. $h(k) = k \bmod m$.
- **Collision**: two keys map to the same index.
- **Load factor** $\alpha = n / m$ (keys / table size).

Method	Idea
Chaining	Each slot holds a linked list
Linear probing	Try h, h+1, h+2, ... (primary clustering)
Quadratic probing	Try h + 1^2, h + 2^2, ... (secondary clustering)
Double hashing	Step = second hash h2(k)

EXAMPLE $m = 7$, $h(k) = k \bmod 7$, linear probing, insert 50, 700, 76, 85. 50 → 1, 700 → 0, 76 → 6, 85 → 1 taken, so 2. Final: 0:700, 1:50, 2:85, 6:76.

9. Searching and Sorting (summary)

Algorithm	Best	Average	Worst	Stable?	In-place?
Linear search	O(1)	O(n)	O(n)	-	-
Binary search	O(1)	O(log n)	O(log n)	-	-
Bubble sort	O(n)	O(n^2)	O(n^2)	Yes	Yes
Selection sort	O(n^2)	O(n^2)	O(n^2)	No	Yes
Insertion sort	O(n)	O(n^2)	O(n^2)	Yes	Yes
Merge sort	O(n log n)	O(n log n)	O(n log n)	Yes	No
Quick sort	O(n log n)	O(n log n)	O(n^2)	No	Yes
Heap sort	O(n log n)	O(n log n)	O(n log n)	No	Yes

- Binary search works only on a sorted array.
- Bubble sort best case O(n) needs the "swapped" flag check.

10. Most Asked Questions (practice)

1. Array vs linked list. Write code to reverse a singly linked list.
2. Convert an infix expression to postfix and prefix using a stack.
3. Evaluate a postfix expression step by step.
4. Explain circular queue with full and empty conditions.
5. Construct a binary tree from given inorder and preorder traversals.
6. Insert keys into a BST and an AVL tree, showing rotations.
7. Build a max-heap and show heap sort on a given array.
8. BFS vs DFS with an example graph.
9. Explain collision resolution techniques with linear probing example.
10. Compare sorting algorithms by time, stability and space.

TIP In GATE, expect questions on tree traversals, BST/AVL insertion, heap array positions, hashing with probing, and n0 = n2 + 1. Draw every step.