DATA STRUCTURES AND ALGORITHMS

QUESTION BANK

UNIT I | LISTS

PART A – 2 Mark Q&A; | PART B – 16 Mark Questions

UNIT I

LISTS

PART A – 2 MARK QUESTIONS WITH ANSWERS

Q. No	Question	Answer
1	What is an Abstract Data Type (ADT)?	An ADT is a mathematical model that defines a data structure by its behaviour (operations) and not by its implementation. Example: List, Stack, Queue.
2	What is the List ADT?	The List ADT is an ordered sequence of elements supporting operations: insert, delete, find, get, size, and isEmpty.
3	What is a singly linked list?	A singly linked list is a linked list where each node contains data and a pointer to the next node. Traversal is possible only in one direction.
4	What is a doubly linked list?	A doubly linked list has nodes with two pointers: one to the next node and one to the previous node, allowing traversal in both directions.
5	What is a circular linked list?	A circular linked list is a linked list where the last node's next pointer points back to the head, forming a circle with no NULL termination.
6	What are the advantages of linked list over arrays?	Dynamic size, efficient insertions/deletions at any position (O(1) with pointer), no pre-allocation needed.

7	What is a Polynomial ADT?	A Polynomial ADT represents a polynomial as a linked list where each node stores a coefficient and exponent. Example: 4x^3+3x^2 stored as [4,3]→[3,2].
8	What is Radix Sort?	Radix Sort is a non-comparative sort that sorts integers digit by digit from least significant to most significant using buckets (0-9). Time: O(d*(n+k)).
9	What is a multi-list?	A multi-list is a data structure where elements can belong to multiple linked lists simultaneously, implemented using multiple next pointers per node.
10	Compare array-based and linked list implementations.	Array: O(1) access, O(n) insert/delete, fixed size. Linked List: O(n) access, O(1) insert/delete with pointer, dynamic size.
11	What is the time complexity of searching in a linked list?	O(n) in both average and worst case since elements must be traversed sequentially from head to the target node.
12	What is a header node in a linked list?	A header (sentinel) node is a dummy node at the beginning of a linked list that simplifies boundary conditions for insert/delete operations.

PART B – 16 MARK QUESTIONS

Q. No	Question (16 Marks)
1	Explain the Array-based implementation of List ADT. Discuss insert, delete, and search operations with algorithms and time complexity analysis.
2	Explain Singly Linked List in detail. Write algorithms for insertion at beginning, end, and a given position, deletion, and traversal with examples.
3	Explain Doubly Linked List and Circularly Linked List. Compare them with Singly Linked List and discuss their advantages and applications.
4	Explain Polynomial ADT using linked list representation. Write algorithms for polynomial addition and multiplication with examples.
5	Explain Radix Sort algorithm with a detailed example. Also explain Multi-lists with suitable applications and diagrams.

DATA STRUCTURES AND ALGORITHMS

QUESTION BANK

UNIT II | STACKS AND QUEUES

PART A – 2 Mark Q&A; | PART B – 16 Mark Questions

UNIT II

STACKS AND QUEUES

PART A – 2 MARK QUESTIONS WITH ANSWERS

Q. No	Question	Answer
1	What is a Stack ADT?	A Stack is a linear LIFO (Last In First Out) data structure supporting push (insert at top), pop (remove from top), and peek operations.
2	What are the applications of Stack?	Balancing symbols, infix-to-postfix conversion, postfix expression evaluation, function call management, undo operations, backtracking.
3	What is infix, postfix, and prefix notation?	Infix: A+B (operator between operands). Postfix: AB+ (operator after). Prefix: +AB (operator before). Postfix is easiest to evaluate using a stack.
4	Convert A+B*C to postfix.	Postfix: A B C * + (since * has higher precedence than +, B*C is evaluated first).
5	What is a Queue ADT?	A Queue is a linear FIFO (First In First Out) data structure. Elements are inserted at the rear (enqueue) and removed from the front (dequeue).
6	What is a Circular Queue?	A Circular Queue is a queue where the rear pointer wraps around to the front after reaching the end of the array, efficiently reusing space.

7	What is a Deque?	A Deque (Double-Ended Queue) allows insertion and deletion at both ends. Types: Input-Restricted (insert only at rear) and Output-Restricted (delete only from front).
8	What is the difference between Stack and Queue?	Stack: LIFO, single end for insert/delete. Queue: FIFO, separate ends (rear for insert, front for delete).
9	How is a stack used for balancing symbols?	Push opening symbols; on closing symbol, pop and check match. If stack is empty at end and all matched, expression is balanced.
10	What is the condition for circular queue full and empty?	Full: (rear+1) % MAX == front. Empty: front == rear.
11	What is the use of stack in function calls?	Each function call pushes a stack frame (return address, local vars, parameters). On return, frame is popped. Enables recursion.
12	Evaluate the postfix expression: 5 4 3 + *	Push 5, push 4, push 3. '+': pop 3,4 → 7, push 7. '*': pop 7,5 → 35, push 35. Result = 35.

PART B – 16 MARK QUESTIONS

Q. No	Question (16 Marks)
1	Explain Stack ADT with array and linked list implementations. Write algorithms for push, pop, and peek with time complexity.
2	Explain infix to postfix conversion algorithm using a stack with detailed steps and examples. Also explain postfix expression evaluation.
3	Explain the application of stack in balancing symbols and function call management. Write algorithms with examples.
4	Explain Queue ADT with array-based implementation. Discuss the problem of linear queue and how Circular Queue solves it with algorithms.
5	Explain Deque (Double-Ended Queue): types, operations, and applications. Compare Stack, Queue, and Deque.

DATA STRUCTURES AND ALGORITHMS

QUESTION BANK

UNIT III | TREES

PART A – 2 Mark Q&A; | PART B – 16 Mark Questions

UNIT III TREES

PART A – 2 MARK QUESTIONS WITH ANSWERS

Q. No	Question	Answer
1	What is a Tree ADT?	A Tree is a non-linear hierarchical data structure with a root node, where each node has zero or more child nodes. No cycles allowed.
2	Define: Root, Leaf, Height, Depth of a tree.	Root: top-most node. Leaf: node with no children. Height: max edges from root to leaf. Depth: edges from root to a specific node.
3	What are the tree traversal methods?	Inorder (Left-Root-Right), Preorder (Root-Left-Right), Postorder (Left-Right-Root), and Level-order (BFS using a queue).
4	What is a Binary Search Tree (BST)?	A BST is a binary tree where left subtree nodes are less than the root and right subtree nodes are greater than the root.
5	What is an AVL Tree?	An AVL Tree is a self-balancing BST where the balance factor (height(left) - height(right)) of every node is at most ±1.
6	What are AVL rotations?	LL Rotation (single right), RR Rotation (single left), LR Rotation (double: left then right), RL Rotation (double: right then left).

7	What is an Expression Tree?	An expression tree stores operands as leaves and operators as internal nodes. Inorder gives infix, preorder gives prefix, postorder gives postfix.
8	What is a Priority Queue?	A Priority Queue is an ADT where elements have priorities. The element with the highest (or lowest) priority is dequeued first.
9	What is a Binary Heap?	A Binary Heap is a complete binary tree satisfying the heap property. Min-Heap: parent ≤ children. Max-Heap: parent ≥ children.
10	What is the time complexity of heap operations?	Insert: O(log n) – sift up. Delete min/max: O(log n) – sift down. Get min/max: O(1). Build heap: O(n).
11	What is a complete binary tree?	A complete binary tree is a binary tree where all levels are fully filled except possibly the last, which is filled from left to right.
12	How is a binary heap stored in an array?	For node at index i: Left child = 2i, Right child = 2i+1, Parent = i/2 (1-indexed). This works because heaps are complete binary trees.

PART B – 16 MARK QUESTIONS

Q. No	Question (16 Marks)
1	Explain Tree Traversals (Inorder, Preorder, Postorder, Level-order) with algorithms, examples, and their applications.
2	Explain Binary Search Tree (BST): insertion, deletion (3 cases), and search operations with algorithms and examples.
3	Explain AVL Trees: balance factor, all four rotation types (LL, RR, LR, RL) with diagrams and examples. Discuss advantages over BST.
4	Explain Expression Trees: construction from postfix expression and traversal to get infix, prefix, and postfix notations with examples.
5	Explain Binary Heap (Min-Heap and Max-Heap): array representation, insertion (sift-up), delete-min (sift-down), and build-heap algorithm.

DATA STRUCTURES AND ALGORITHMS

QUESTION BANK

UNIT IV | MULTIWAY SEARCH TREES AND GRAPHS

PART A – 2 Mark Q&A; | PART B – 16 Mark Questions

UNIT IV

MULTIWAY SEARCH TREES AND GRAPHS

PART A – 2 MARK QUESTIONS WITH ANSWERS

Q. No	Question	Answer
1	What is a B-Tree?	A B-Tree of order m is a self-balancing multiway search tree where each node has at most m children, all leaves are at the same level, and is used for disk-based storage.
2	What are the properties of a B-Tree of order m?	Max m children per node. Min ⌈m/2⌉ children (non-root). Root has ≥2 children. All leaves at same level. Node with k children has k-1 keys.
3	What is a B+ Tree?	A B+ Tree stores all data in leaf nodes only. Internal nodes store only keys. Leaf nodes are linked as a sorted list for efficient range queries.
4	Differentiate B-Tree and B+ Tree.	B-Tree: data in all nodes, leaves not linked. B+ Tree: data only in leaves, leaves linked, better for range queries. B+ used in databases (MySQL InnoDB).
5	What is a Graph?	A Graph G=(V,E) consists of vertices V and edges E. Edges can be directed/undirected and weighted/unweighted.
6	What are the types of graph representation?	Adjacency Matrix: $O(V^2)$ space, O(1) edge check. Adjacency List: O(V+E) space, better for sparse graphs.

7	What is BFS?	BFS (Breadth-First Search) explores all neighbours at current depth first using a Queue. Time: O(V+E). Finds shortest path in unweighted graphs.
8	What is DFS?	DFS (Depth-First Search) explores as far as possible before backtracking using a Stack/recursion. Time: O(V+E). Used for topological sort, cycle detection.
9	What is Topological Sort?	Topological sort orders vertices of a DAG such that for every edge u→v, u appears before v. Possible only for DAGs (no cycles).
10	What is Dijkstra's algorithm?	Dijkstra's algorithm finds the shortest path from a source to all vertices in a weighted graph with non-negative weights. Uses a priority queue. Time: O((V+E) log V).
11	What is a Minimum Spanning Tree (MST)?	An MST is a spanning tree of a connected weighted graph with the minimum total edge weight. Has exactly V-1 edges and no cycles.
12	Differentiate Prim's and Kruskal's algorithms.	Prim's: vertex-based greedy, grows tree one vertex at a time, O(E log V). Kruskal's: edge-based greedy, sorts edges by weight, uses DSU, O(E log E).

PART B – 16 MARK QUESTIONS

Q. No	Question (16 Marks)
1	Explain B-Tree of order m: properties, search, insertion, and deletion operations with examples and diagrams.
2	Explain B+ Tree: structure, differences from B-Tree, insertion and deletion with examples. Discuss its use in database indexing.
3	Explain Graph representation (Adjacency Matrix and Adjacency List). Write BFS and DFS algorithms with examples and comparison.
4	Explain Dijkstra's shortest path algorithm with a detailed example. Discuss its time complexity and limitations.
5	Explain Prim's and Kruskal's algorithms for Minimum Spanning Tree with step-by-step examples and comparison.

DATA STRUCTURES AND ALGORITHMS

QUESTION BANK

UNIT V | SEARCHING, SORTING AND HASHING

PART A – 2 Mark Q&A; | PART B – 16 Mark Questions

UNIT V SEARCHING, SORTING AND HASHING

PART A – 2 MARK QUESTIONS WITH ANSWERS

Q. No	Question	Answer
1	What is Linear Search?	Linear Search sequentially checks each element until the target is found. Time: O(n) worst case. Works on unsorted arrays.
2	What is Binary Search?	Binary Search finds an element in a sorted array by repeatedly halving the search space. Time: O(log n). Requires sorted input.
3	What is Bubble Sort?	Bubble Sort repeatedly compares adjacent elements and swaps if out of order. Largest element bubbles to end each pass. Time: $O(n^2)$.
4	What is Selection Sort?	Selection Sort finds the minimum element from the unsorted part and places it at the correct position. Time: $O(n^2)$ always. Not stable.
5	What is Insertion Sort?	Insertion Sort builds a sorted array by inserting each element into its correct position. Time: O(n) best, $O(n^2)$ worst. Stable sort.
6	What is Shell Sort?	Shell Sort is an improved Insertion Sort that sorts elements far apart first using a gap sequence, then reduces gap until 1. Better than $O(n^2)$ on average.

7	What is Merge Sort?	Merge Sort is a divide-and-conquer algorithm: divide array in half, recursively sort, merge. Time: O(n log n) always. Space: O(n). Stable.
8	What is Hashing?	Hashing maps keys to table positions using a hash function. Provides O(1) average-case for search, insert, and delete.
9	What is a collision in hashing?	A collision occurs when two different keys produce the same hash value (index). Resolved by separate chaining or open addressing.
10	What is Separate Chaining?	Separate Chaining stores colliding elements in a linked list at each hash table index. Load factor $\lambda = n/m$ can exceed 1.
11	What is Open Addressing?	Open Addressing stores all elements in the hash table itself. On collision, probes for the next empty slot using linear, quadratic, or double hashing.
12	What is Rehashing?	Rehashing creates a new larger table (typically 2× size, next prime) and re-inserts all elements when the load factor exceeds a threshold (0.5–0.75).

PART B – 16 MARK QUESTIONS

Q. No	Question (16 Marks)
1	Explain Linear Search and Binary Search algorithms with examples, time complexity analysis, and comparison.
2	Explain Bubble Sort, Selection Sort, and Insertion Sort with step-by-step examples and time/space complexity comparison.
3	Explain Shell Sort and Merge Sort algorithms with detailed examples, recurrence relations, and complexity analysis.
4	Explain Hashing: hash functions (division, multiplication, folding), separate chaining, and all open addressing methods (linear, quadratic, double hashing) with examples.
5	Explain Rehashing and Extendible Hashing with examples. Discuss when rehashing is triggered and how extendible hashing grows dynamically.