Data Structures – Notes

Unit 1: Overview and Arrays

1. Data Structures

Detailed Definition: Data structures are specialized formats for organizing, storing, and managing data in a computer so that it can be used efficiently. They provide a way to handle information so that it can be accessed and modified effectively.

Types of Data Structures:

- **Linear:** Arrays, Linked Lists, Stacks, Queues
- **Non-linear:** Trees, Graphs
- **Homogeneous:** Same data type (Arrays)
- **Non-homogeneous:** Different data types (Structures)

2. Data Abstraction

Detailed Definition: The concept of hiding the complex implementation details and showing only the essential features to the user. It focuses on what the data structure does rather than how it does it.

Example: When using a car, you only need to know how to drive, not how the engine works internally.

3. Arrays

Detailed Definition: A collection of elements of the same data type stored in contiguous memory locations. Each element can be accessed directly using its index.

Characteristics:

- Fixed size
- Random access
- Memory efficient
- Same data type elements

Declaration:

```
int numbers[5] = {1, 2, 3, 4, 5};
```

4. Searching Algorithms

Linear Search

Detailed Definition: A simple search algorithm that checks each element in the list sequentially until the target element is found or the list ends.

Time Complexity: O(n)
Best for: Small or unsorted arrays

Process:

1. Start from first element
2. Compare with target
3. If match, return position
4. If not, move to next element
5. Repeat until found or end

Binary Search

Detailed Definition: An efficient search algorithm that finds the position of a target value within a sorted array by repeatedly dividing the search interval in half.

Time Complexity: O(log n)
Requirements: Sorted array

Process:

1. Find middle element
2. If matches target, return position
3. If target smaller, search left half
4. If target larger, search right half
5. Repeat until found or interval empty

5. Sorting Algorithms

Bubble Sort

Detailed Definition: A simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in wrong order.

Time Complexity: $O(n^2)$
Process:

- Compare element 1 with 2, swap if needed
- Compare element 2 with 3, swap if needed
- Continue until end
- Repeat process until no swaps needed

Insertion Sort

Detailed Definition: Builds the final sorted array one item at a time by inserting each new element into its proper position in the already sorted part.

Time Complexity: $O(n^2)$
Best for: Small arrays or nearly sorted arrays

Process:

1. Consider first element as sorted
2. Pick next element

3. Insert it into correct position in sorted part
4. Repeat for all elements

Selection Sort

Detailed Definition: Finds the minimum element from unsorted part and puts it at the beginning, then repeats for remaining elements.

Time Complexity: $O(n^2)$
Process:

1. Find minimum element in array
2. Swap with first position
3. Find minimum in remaining array
4. Swap with second position
5. Repeat until sorted

Video Links:

- **English:** Arrays and Sorting Algorithms
- **Hindi:** Data Structures Basics in Hindi

Unit 2: Linked Lists

1. Linked List

Detailed Definition: A linear data structure where elements are stored in nodes, and each node contains a data field and a reference (link) to the next node in the sequence.

Advantages over Arrays:

- Dynamic size
- Easy insertion/deletion
- No memory waste

Disadvantages:

- No random access
- Extra memory for pointers
- Cache unfriendly

2. Types of Linked Lists

Singly Linked List

Detailed Definition: Each node points only to the next node in the sequence. The last node points to NULL.

Operations:

- **Insertion:** At beginning, end, or specific position
- **Deletion:** From beginning, end, or specific position

- **Traversal:** Visit each node sequentially

Doubly Linked List

Detailed Definition: Each node contains two pointers - one to the next node and one to the previous node.

Advantages:

- Can traverse in both directions
- Easier deletion

Disadvantages:

- Extra memory for previous pointer
- More complex operations

Circular Linked List

Detailed Definition: The last node points back to the first node, forming a circle. Can be singly or doubly circular.

Applications:

- Round-robin scheduling
- Music playlists
- Browser history

3. Linked List Operations

Insertion at Beginning:

1. Create new node
2. New node's next = current head
3. Head = new node

Deletion from Beginning:

1. Store head in temp
2. Head = head→next
3. Free temp

Traversal:

1. Start from head
2. While current != NULL
3. Process current node
4. Current = current→next

Video Links:

- **English:** Linked Lists Tutorial

- **Hindi:** Linked Lists in Hindi

Unit 3: Stacks and Queues

1. Stack

Detailed Definition: A linear data structure that follows LIFO (Last-In-First-Out) principle. The last element added is the first one to be removed.

Operations:

- **Push:** Add element to top
- **Pop:** Remove element from top
- **Peek/Top:** View top element without removing
- **isEmpty:** Check if stack is empty

Implementation:

- Using arrays
- Using linked lists

Applications:

- Function call management
- Expression evaluation
- Undo/Redo operations
- Back button in browsers

2. Recursion

Detailed Definition: A programming technique where a function calls itself directly or indirectly to solve a problem.

Components:

- **Base Case:** Condition to stop recursion
- **Recursive Case:** Function calls itself

Example - Factorial:

```
factorial(n):
    if n == 0: return 1
    else: return n * factorial(n-1)
```

3. Expression Evaluation

Detailed Definition: Using stacks to evaluate arithmetic expressions by converting them between different notations.

Notations:

- **Infix:** A + B (operator between operands)

- **Prefix:** + A B (operator before operands)
- **Postfix:** A B + (operator after operands)

4. Queue

Detailed Definition: A linear data structure that follows FIFO (First-In-First-Out) principle. The first element added is the first one to be removed.

Operations:

- **Enqueue:** Add element to rear
- **Dequeue:** Remove element from front
- **Front:** Get front element
- **Rear:** Get last element

5. Types of Queues

Simple Queue

- Basic FIFO structure
- Fixed size

Circular Queue

- Rear connects back to front
- Efficient memory usage
- Prevents wastage

Double-ended Queue (Deque)

- Insert/delete from both ends
- More flexible

Priority Queue

- Elements with priority
- Higher priority served first
- Implemented using heaps

Applications:

- CPU scheduling
- Print spooling
- Breadth-first search

Video Links:

- **English:** Stacks and Queues
- **Hindi:** Stacks and Queues in Hindi

Unit 4: Trees

1. Tree

Detailed Definition: A non-linear hierarchical data structure consisting of nodes connected by edges. Each tree has a root node, and nodes may have zero or more child nodes.

Terminology:

- **Root:** Topmost node
- **Parent/Child:** Relationship between nodes
- **Leaf:** Node with no children
- **Depth:** Number of edges from root to node
- **Height:** Maximum depth of any node

2. Binary Tree

Detailed Definition: A tree where each node has at most two children, referred to as left child and right child.

Types:

- **Full Binary Tree:** Every node has 0 or 2 children
- **Complete Binary Tree:** All levels filled except possibly last
- **Perfect Binary Tree:** All internal nodes have 2 children, all leaves same level

3. Tree Traversal Techniques

Inorder (Left-Root-Right)

1. Traverse left subtree
2. Visit root
3. Traverse right subtree

Preorder (Root-Left-Right)

1. Visit root
2. Traverse left subtree
3. Traverse right subtree

Postorder (Left-Right-Root)

1. Traverse left subtree
2. Traverse right subtree
3. Visit root

Level Order

- Visit nodes level by level

- Uses queue

4. Binary Search Tree (BST)

Detailed Definition: A binary tree where for each node:

- Left subtree contains values less than node
- Right subtree contains values greater than node

Operations:

- Search: O(h) time
- Insert: O(h) time
- Delete: O(h) time
  (where h is height of tree)

5. AVL Tree

Detailed Definition: A self-balancing binary search tree where the difference between heights of left and right subtrees cannot be more than 1 for all nodes.

Balancing Operations:

- **Left Rotation**
- **Right Rotation**
- **Left-Right Rotation**
- **Right-Left Rotation**

6. B-Tree

Detailed Definition: A self-balancing search tree that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.

Characteristics:

- All leaves same depth
- Minimum and maximum keys per node
- Used in databases and file systems

7. Heap Tree

Detailed Definition: A complete binary tree that satisfies the heap property.

Types:

- **Max Heap:** Parent >= Children
- **Min Heap:** Parent <= Children

Applications:

- Priority queues
- Heap sort

- Graph algorithms

Video Links:

- **English:** Tree Data Structures
- **Hindi:** Trees in Hindi

Unit 5: Graphs

1. Graph

Detailed Definition: A non-linear data structure consisting of vertices (nodes) and edges that connect these vertices.

Terminology:

- **Vertex:** Fundamental unit (node)
- **Edge:** Connection between vertices
- **Directed Graph:** Edges have direction
- **Undirected Graph:** Edges have no direction
- **Weighted Graph:** Edges have weights
- **Path:** Sequence of vertices connected by edges

2. Graph Representations

Adjacency Matrix

- 2D array of size V×V
- Matrix[i][j] = 1 if edge exists, else 0
- Space: $O(V^2)$
- Good for dense graphs

Adjacency List

- Array of lists
- Each list stores neighbors of a vertex
- Space: O(V+E)
- Good for sparse graphs

3. Graph Traversal Algorithms

Breadth-First Search (BFS)

Detailed Definition: Explores all neighbors at current depth before moving to next level.

Uses Queue
Applications: Shortest path in unweighted graphs, social networks

Process:

1. Start from source vertex
2. Visit all neighbors
3. Then neighbors of neighbors
4. Continue level by level

Depth-First Search (DFS)

Detailed Definition: Explores as far as possible along each branch before backtracking.

Uses Stack (recursion)
Applications: Cycle detection, topological sort, maze solving

Process:

1. Start from source
2. Go to first unvisited neighbor
3. Continue deep until dead end
4. Backtrack and try other paths

4. Shortest Path Algorithms

Dijkstra's Algorithm

- Single source shortest path
- Non-negative weights
- Greedy approach

Bellman-Ford

- Single source shortest path
- Handles negative weights
- Dynamic programming

Floyd-Warshall

- All pairs shortest path
- Dynamic programming

5. Applications of Graphs

- Social networks
- Maps and navigation
- Computer networks
- Recommendation systems

Video Links:

- **English:** Graph Data Structures
- **Hindi:** Graphs in Hindi

Unit 6: Hashing and Memory

1. Hashing

Detailed Definition: A technique that maps data of arbitrary size to fixed-size values using a hash function. The values are used to index a hash table.

Components:

- **Hash Function:** Converts key to index
- **Hash Table:** Array storing data
- **Collision:** When two keys map to same index

2. Hash Functions

Characteristics of Good Hash Function:

- Uniform distribution
- Fast computation
- Minimal collisions

Types:

- Division method
- Multiplication method
- Universal hashing

3. Collision Resolution Techniques

Open Addressing

- Store in next available slot
- **Linear Probing:** Check next slot
- **Quadratic Probing:** Check i^2 slots away
- **Double Hashing:** Use second hash function

Separate Chaining

- Each slot contains linked list
- Colliding elements stored in same list
- Simple implementation

4. Memory Management

Static Memory Allocation

- Memory allocated at compile time
- Size fixed
- Fast access

- Examples: Global variables, static variables

Dynamic Memory Allocation

- Memory allocated at runtime
- Size can change
- Flexible but slower
- Examples: malloc(), calloc(), new

Video Links:

- **English:** Hashing and Hash Tables
- **Hindi:** Hashing in Hindi

Unit 7: Advanced Applications and Complexity

1. Performance Analysis

Big-O Notation

Detailed Definition: Mathematical notation describing the limiting behavior of a function when the argument tends towards infinity. Used to classify algorithms by how they respond to changes in input size.

Common Complexities:

- **O(1):** Constant time
- **O(log n):** Logarithmic time
- **O(n):** Linear time
- **O(n log n):** Linearithmic time
- **O(n^2):** Quadratic time
- **O(2^n):** Exponential time

2. Divide and Conquer

Detailed Definition: A problem-solving paradigm that breaks a problem into smaller subproblems, solves them recursively, and combines their solutions.

Examples:

- Merge Sort
- Quick Sort
- Binary Search

3. Recurrence Relations

Detailed Definition: Equations that describe functions in terms of their values on smaller inputs, used to analyze recursive algorithms.

Solving Methods:

- Substitution method
- Recursion tree method
- Master theorem

4. Abstract Data Types (ADT)

Detailed Definition: A mathematical model for data types that defines the type solely by the operations that can be performed on it, not by implementation.

Examples:

- **List:** Insert, delete, search
- **Stack:** Push, pop, peek
- **Queue:** Enqueue, dequeue
- **Tree:** Insert, delete, traverse

5. Standard Libraries

- **C++:** STL (Standard Template Library)
- **Java:** Collections Framework
- **Python:** Built-in data structures

Video Links:

- **English:** Algorithm Complexity
- **Hindi:** Big O Notation in Hindi

Important Concepts Summary

Time Complexities of Common Operations:

Data Structure	Access	Search	Insert	Delete
Array	O(1)	O(n)	O(n)	O(n)
Linked List	O(n)	O(n)	O(1)	O(1)
Stack	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)
BST	O(n)	O(n)	O(n)	O(n)
Hash Table	O(1)	O(1)	O(1)	O(1)

When to Use Which Data Structure:

- **Arrays:** When size is fixed and random access needed
- **Linked Lists:** When frequent insertions/deletions
- **Stacks:** When LIFO behavior needed
- **Queues:** When FIFO behavior needed
- **Trees:** When hierarchical data
- **Graphs:** When relationships between data
- **Hash Tables:** When fast lookup needed

Real-World Applications:

- **Arrays:** Student marks, employee records
- **Linked Lists:** Music playlists, browser history
- **Stacks:** Undo operations, function calls
- **Queues:** Ticket counter, CPU scheduling
- **Trees:** File systems, organization charts
- **Graphs:** Social networks, maps
- **Hash Tables:** Dictionaries, database indexing

Final English Video: Data Structures Full Course
Final Hindi Video: Data Structures Complete Course