① Introduction to Data structure

② Need for Data structure

③ Application of Data structure
- Databoses – B-Tree, Hash Table
- Operating System - Queue, Stack, Linked List
- Compiler Design - Trees
- Networking – Graphs
- AI/ML - Trees & Graphs (Neural Networks)
- Web Development – Arrays & Hash Table
- Graphics & Computer Vision – Trees
- Game Development - Trees, Graphs
- Undo Redo feature - Stack
- Contact Details on Cell phone - Arrays
- Image Processing – Storing image pixel details
- Social Media – Storing friendship information on a social Media networking (Graphs)

③ Classification of DS
- Primitive
  - Integer
  - Float
  - Character (string is collection of characters)
  - Boolean
  - Pointer
- Non Primitive
  - Linear
    - Static – Arrays
    - Dynamic
      - Linked List
      - Stack
      - Queue
  - Non Linear
    - Graphs
    - Tree

④ Abstract Data Types (ADT's) ✓

⑤ Structure, Typedef, Union

⑥ Homogeneous v/s Non Homogeneous ✓

Homogeneous: Element of same Type

Non Homogeneous: Element of diff Type

⑦ Static v/s Dynamic

⑧ Linear v/s Non Linear ✓

Linear: Sequential Order, Array, LL

Non Linear: Hierarchical Order, Trees, Graphs.

① Arrays

- Single Dimension (1D Arrays) - Syntax - int array[10] = {1,2,3,4,5 . . . 10};
- How to calculate size of array (1D)
- Types of ~~1D~~ array
  - 1D
  - Multidimentional ~~1D~~ - 2D
  - Fixed Size / ~~Dynamic~~ Static Array
  - Dynamic Array
  - Sparse Array
  - Jogged array
  - String Array
  - Boolean Array
  - Character Array
  - Object Array
- 1D Array ~~Insertion~~ Traversal
- 1D Array Insertion
  - Insertion at beginning
  - Insertion at Middle or at position
  - Insertion at End
- 1D Array Deletion
  - Deletion at Beginning
  - Deletion at Middle or at position
  - Deletion at End
- Memory Representation
- Address Calculation - B + index × size

  B = Base Address
  Index = Element Index
  Size = Size of the data type
  (In Array Random / Direct Access is possible)
- 1D Array Searching
  - Linear Search
  - Binary Search 5★

D) Array
- 2D Declaration - int array[2][2];
- Declaration & Initialization at compile time - int array[2][2] = {{10,20},{30,40}};
- Declaration & Initialization at run time - Jab user se input lete h & array m store krte h usse runtime kehte h.
- 2D Array Traversal
- Memory Representation
  - Row Major
  - Column Major

	0	1	2
0	1	2	3
1	4	5	6

m×n
2×3

array[1][1]
array[i][j]

```
int array[2][3];
for(int i=0; i<2; i++){
    for(int j=0; j<3; j++){
        scanf("%d", &array[i][j]);
    }
}
```

Random Access Row Major - B+((i×n)+j)*size

Random Access Column Major - B+((j*m)+i)*size

) Sparse Matrix – 2D Array or Matrix with maximum no. of zeros
- Memory Representation
  - Coordinate List / 3 column Array or 3 Tuple Representation
  - Compressed Sparse row / Linked Representation
- Addition of Two Sparse Matrix (Theory + Diagram + Code)
- Multiplication of Two Sparse Matrix (Theory + Diagram + Code)
- Types of Sparse Matrix
  - Lower Triangular → $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 \\ 6 & 4 & 3 & 0 \end{bmatrix}$
  - Upper Triangular → $\begin{bmatrix} 1 & 5 & 6 & 10 \\ 0 & 2 & 4 & 9 \\ 0 & 0 & 3 & 8 \\ 0 & 0 & 0 & 7 \end{bmatrix}$

Why Binary Search cannot be used with linked list

Sorting Algorithms
- Bubble Sort
- Insertion Sort → Shifting (Insert an element from unsorted array to its correct position in sorted array)
- Selection Sort → Swapping (Find the minimum element in unsorted array & swap it with element at begining)
- Quick Sort → In Place Sort
- Merge Sort
- Radix Sort
- Bucket Sort
- Shell Sort

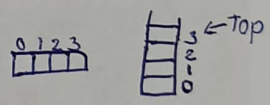
In Place Sorting algorithms

(Theory + Code + Diagram)

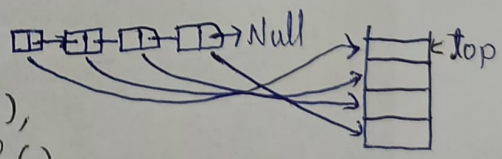
Revise

4) Linked List
- Static v/s Dynamic Memory Allocation
- What is self referential structure
- Functions of Memory Allocation - malloc(), calloc(), realloc(), free()
- Singly Linked List
  - Insertion - Beginning, Middle /At Position, End
  - Deletion - Beginning, Middle, End
  - Traversal
  - Searching
- Doubly Linked List
  - Insertion - Biggening, Middle, End
  - Deletion - Beginning, Middle, End
  - Traversal
  - Searching
- Circular Linked List
  - Insertion - Beginning Middle, End
  - Deletion - Beginning, Middle, End
  - Traversal
  - Searching
- Application of linked list
- Application's polynomial Arithematic

Stacks
- Introduction & Implementation
- Static Stack Using Arrays (code + Theory)



- Dynamic stack using ~~Linked List~~ Arrays (Not Used just for learning)
- Dynamic stack using Linked List



- Stack Operations - IsEmpty(), IsFull(), Push(), Pop(), Peek(), stackTop(), StackBottom(), Traversal()

Push(Insertion) is done at the beginning of the linked list

- Application of stack
  - Parenthesis Matching
  - Conversion b/w Polish & Reverse polish Notations (Infix to Postfix) (Infix to Prefix) (Postfix to Prefix) (Prefix to Postfix)
  - Quick Sort
  - Recursion

Numerical
Code
Theory

6) Queue
- Introduction & Implementation
- Types of Queues -
- Applications of Queue
  - Task Scheduling, Printer Queues
  - Resource Allocation, Web servers
  - Batch Processing, BFS, CPU scheduling
  - OS
- Queue using Arrays
- Queue using Linked List
- Queue Operations - EnQueue(), DeQueue(), Display()
- Double Ended Queue - EnQueueFront(), EnQueueRear(), DeQueueFront(), DeQueueRear(), Display()
- Circular Queue - EnQueue(), DeQueue(), Display()

~~Enqueue()~~

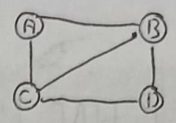
) Trees
- Notations & Terminologies (Theory)
- Binary Tree
  - Creation
  - Traversal
    - PreOrder Traversal
    - InOrder Traversal
    - Post Order Traversal
- Binary Search Tree
  - Creation
  - Traversal
    - PreOrder Traversal
    - InOrder Traversal
    - PostOrder Traversal
  - Insertion
  - Deletion
  - Searching
    - Recursive
    - Iterative
- Threaded Binary Tree
- Tree Sort
- Tries
- AVL Trees
  - Theory + Advantage + Disadvantage + Application + Properties
  - AVL Tree Rotation - LL, LR, RL, RR (Numerical) 5☆
  - Insertion
  - Deletion
  - Traversal
    - PreOrder Traversal
    - InOrder Traversal
    - Post Order Traversal

- M Way Trees
  - General Concept
  - B Trees
  - B+ Trees
  - B* Trees

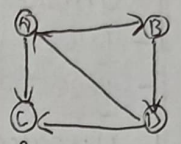
- ~~Heaps~~
  - ~~Structural Properties~~
  - ~~Heapify~~
  - ~~Heap Sort~~
  - ~~Priority Queue Implementation~~

- Heap
  - Definition
  - Binary Heap
  - Types of Heap
    - Min Heap
    - Max Heap
  - Min Heap
    - Insertion
    - Deletion
  - Max Heap
    - Insertion
    - Deletion
  - Heapify
  - Heap Operations
    - Insertion
    - Deletion
    - Heapify
  - Heap Sort
  - Priority Queue Implementation

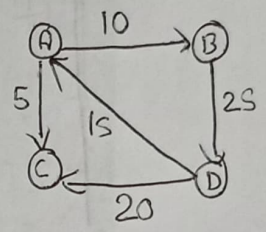
⑨ Graphs
- Theory (Directed/Undirected Graphs) (Weighted) (Unweighted)
- Representation of Graphs
  - Adjacency Matrix
  - Adjacency List/Linked Representation
- Types of Graphs
  - Undirected Graphs



  - Directed Graphs



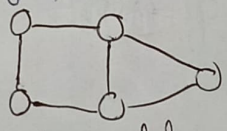
  - Weighted Directed Graph



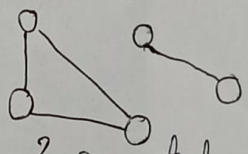
- Graphs Traversal
  - BFS (Level By Level)
    - Uses queue
    - Visited array
    - explored queue
  - DFS – Uses stack (Depth By Depth)

  } code + Theory + Diagram + Algo ☆

- Connected Components



1 connected component



2 connected component

⑩ ~~Spanning Trees Topology Sort~~

Spanning Tree
- Minimum Spanning tree
- Greedy Approach
  - Prims algorithm
  - Kruskal's algorithm

Using Min Heap Time complexity of Kruskal can be reduced to $O(n \log n)$

$|E| = |V| - 1$

$O(|V||E|) = (o(n.e)) = O(n^2)$

Shortest Path Algorithms
- Single Source & all Pairs
- Dijkstras Algorithm – Greedy
- Floyd Warshall – Dynamic
- Topology Sort

Formula

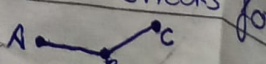
$A^k[i,j] = \min\{A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j]\}$

intermediate vertex value

D – Single Sou
F – All pair

D – Greedy A
F – Dynamic

D – Checks for
F – Checks for

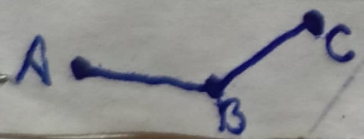


Case Studies
- Dynamic Graphs
- Social Networks
- Sparse Graphs

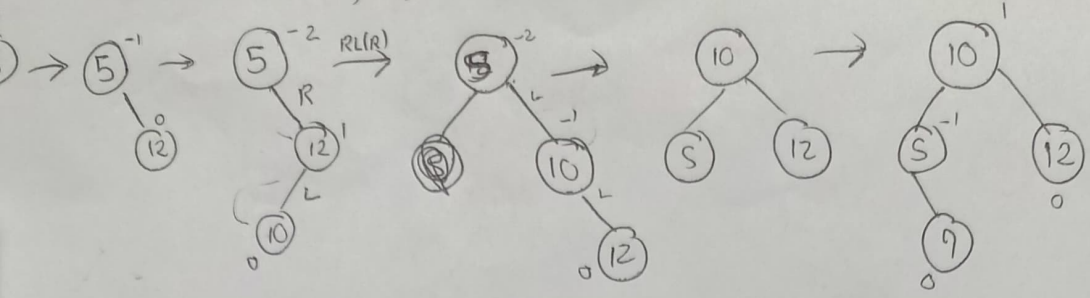
D - Single Source shortest path
F - All pair shortest path

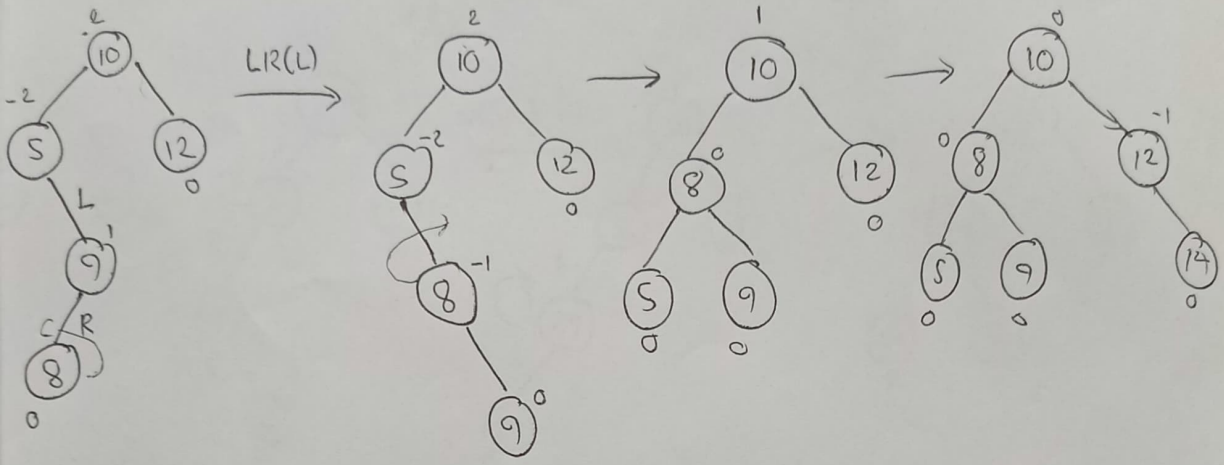
D - Greedy Algo
F - Dynamic Programming

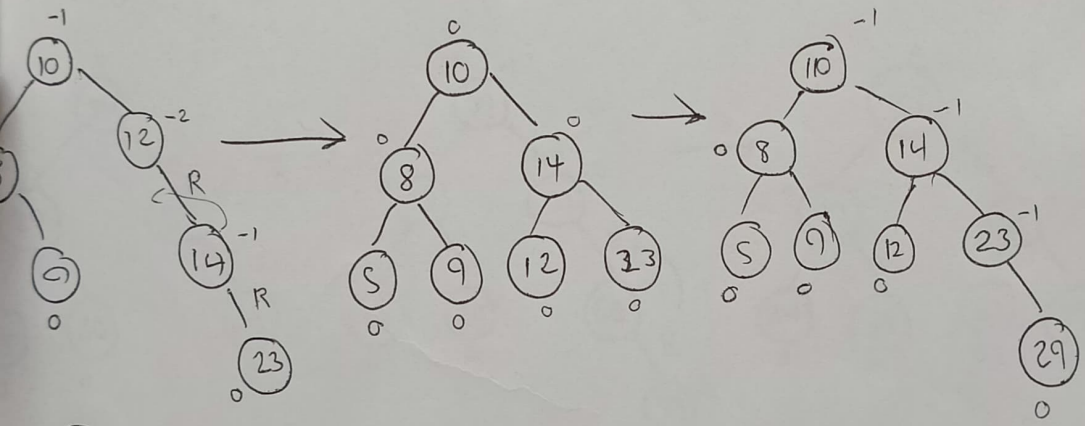
D - Checks for Direct Path A→C
F - Checks for indirect path A→B→C

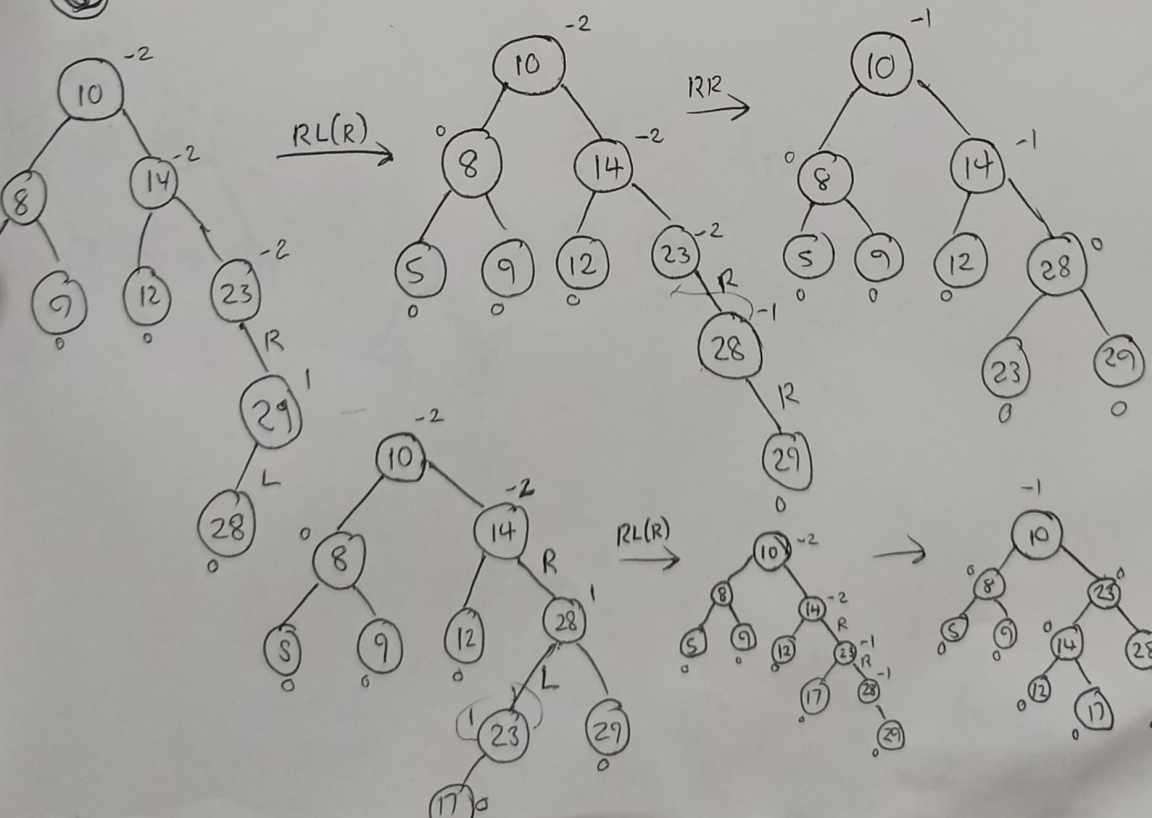


5,12,10,9,8,14,23,29,28,17

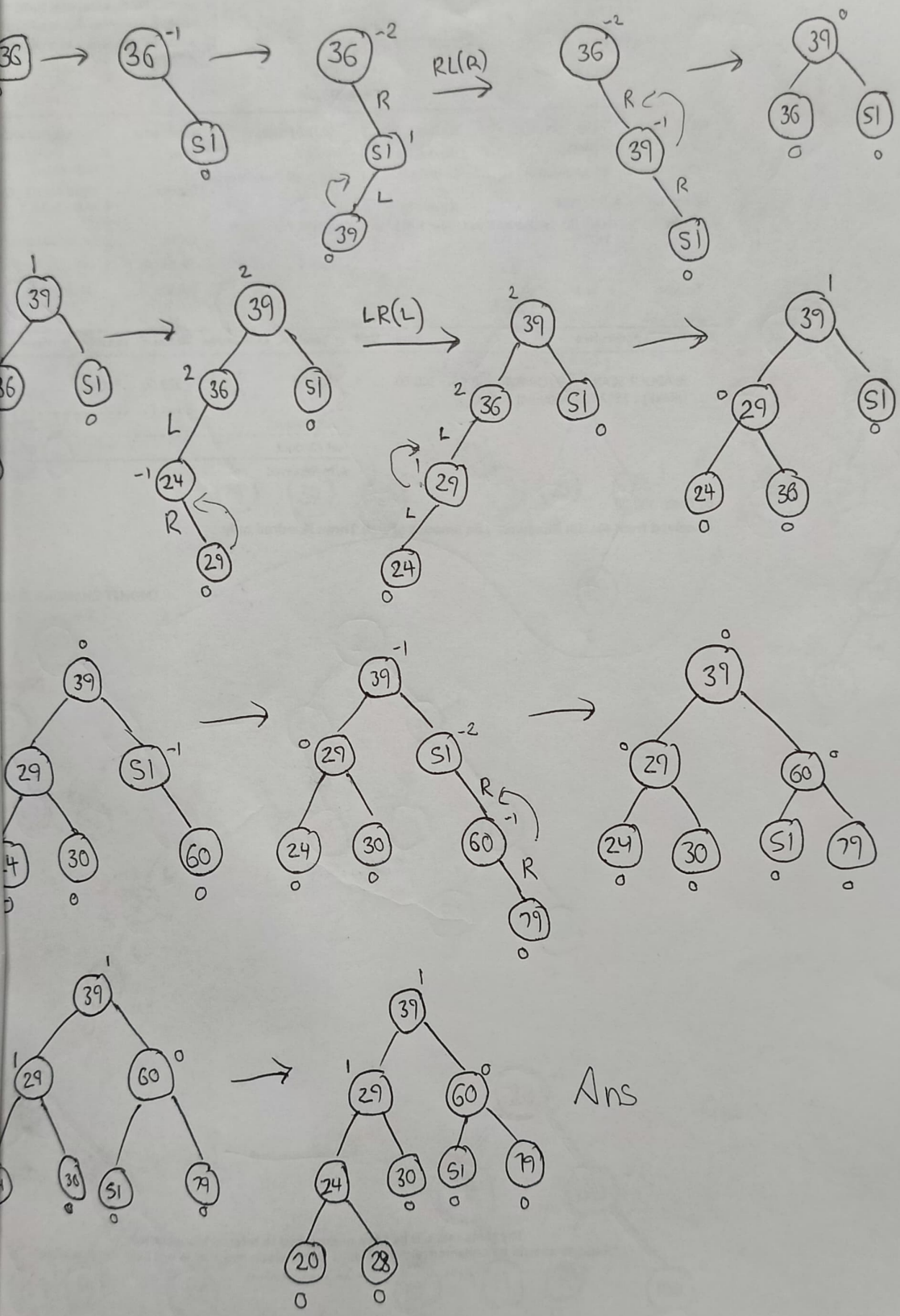








6, 51, 39, 24, 29, 60, 79, 20, 28



60, 10, 20, 30, 19, 120, 100, 80, 19