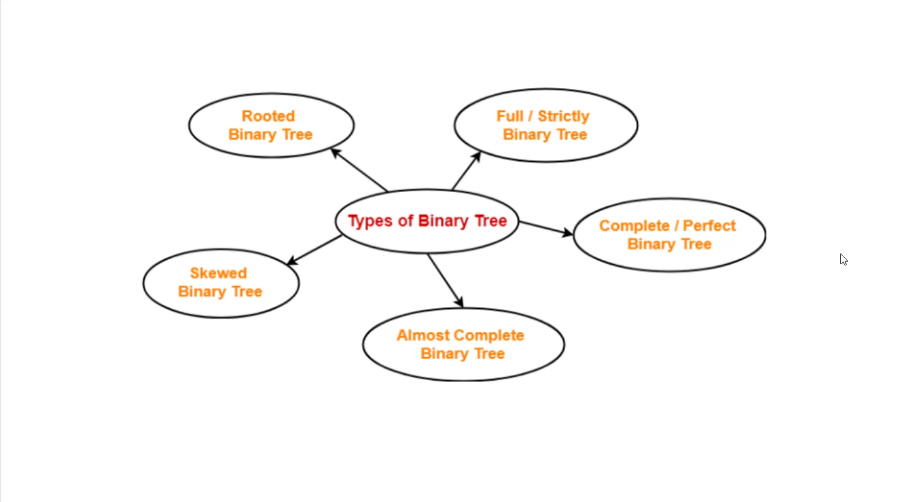
Day 16 – 3rd July 2025

****

## ****Rooted Binary Tree****

The most basic form where there's a designated root node, and every node has at most two children (left and right). It's the foundation for all other binary tree types.

## ****Full/Strictly Binary Tree****

Every node has either 0 children (leaf node) or exactly 2 children. No node can have just one child. This creates a tree where all internal nodes are "full" with two branches.

## ****Complete/Perfect Binary Tree****

* **Complete**: All levels are completely filled except possibly the last level, which fills from left to right
* **Perfect**: All internal nodes have exactly two children AND all leaves are at the same depth. This is the most "balanced" possible binary tree structure.

## ****Skewed Binary Tree****

A degenerate binary tree where each node has only one child, making it essentially a linked list. Can be:

* **Left-skewed**: All nodes have only left children
* **Right-skewed**: All nodes have only right children This results in poor performance (O(n) instead of O(log n)) for operations.

## ****Almost Complete Binary Tree****

Similar to a complete binary tree, but allows the last level to have some missing nodes on the right side. All levels are filled except the last, and the last level is filled from left to right up to a certain point.

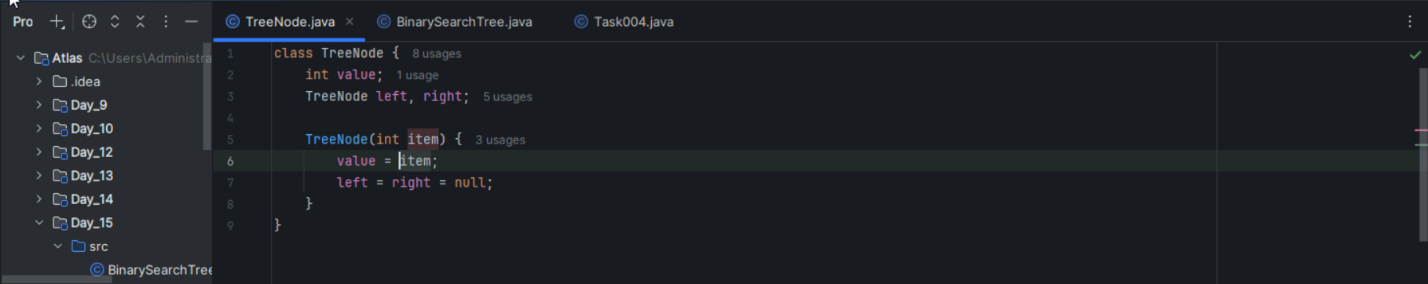
These classifications help determine the efficiency of tree operations and are important for choosing the right tree structure for specific applications. Perfect and complete trees offer the best balance and performance, while skewed trees should generally be avoided or rebalanced.

Task 1:

Create a node for a tree and include a constructor (empty)

Hint

A node which consists of 1 data part and 2 refs (1 Left ref and another right ref)



Task 2:

Create a class named Binarty Search tree in which you have 2 insert operations

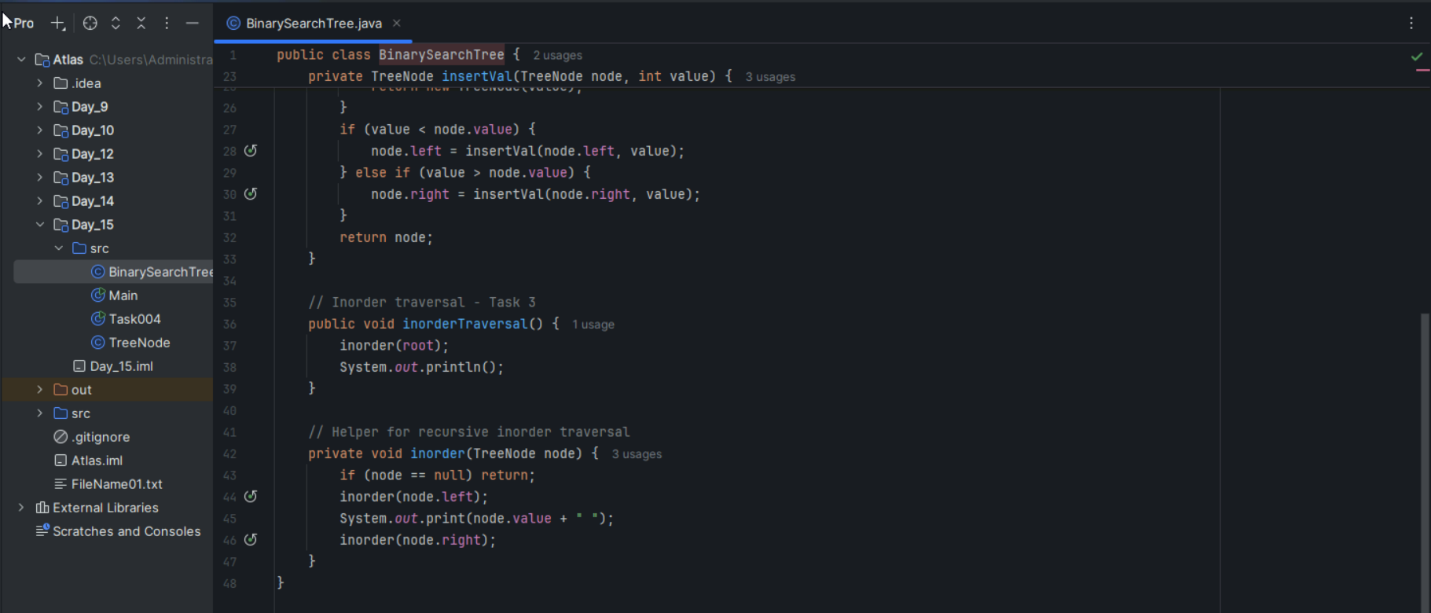
1 insert —----> for inserting if the tree is empty

1 insert —----> for inserting if the tree is 1 or more nodes

Task 3:

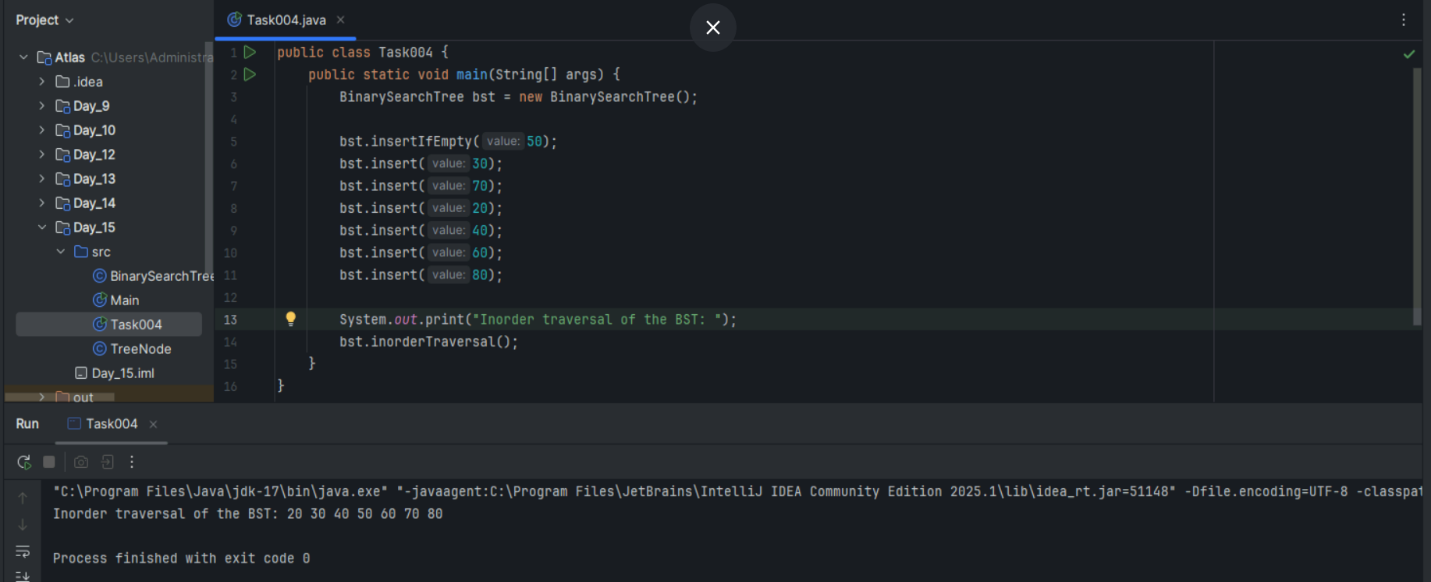
In order travel of the above code snippets from task 1 and Task 2





Task 4:

Create a main method Task 1, 2 and 3



Solution for Task 1 to task 4

In order Binary Search tree..

class TreeNode {

    int value;

    TreeNode left, right;

    TreeNode(int item) {

        value = item;

        left = right = null;

    }

}

class BinarySearchTreeOp {

    TreeNode root;

    void insert(int value) {

        root = insertVal(root, value);

    }

    TreeNode insertVal(TreeNode node, int value) {

        if (node == null) {

            node = new TreeNode(value);

            return node;

        }

        if (value < node.value) {

            node.left = insertVal(node.left, value);

        } else if (value > node.value) {

            node.right = insertVal(node.right, value);

        }

        return node;

    }

    void inorder() {

        inorderVal(root);

    }

    void inorderVal(TreeNode node) {

        if (node != null) {

            inorderVal(node.left);

            System.out.print(node.value + " ");

            inorderVal(node.right);

        }

    }

}

public class BinarySearchTree {

    public static void main(String[] args) {

        BinarySearchTreeOp bstobj = new BinarySearchTreeOp();

bstobj.insert(10);

bstobj.insert(50);

bstobj.insert(40);

bstobj.insert(70);

bstobj.insert(5);

Sytem.out.println("here is the code for in order traversal of Binary search tree ");

bstobj.inorder();

    }

}

Task 5:

Applications of Trees

**File Systems**: Operating systems use tree structures to organize directories and files hierarchically, with root directory at top and subdirectories as branches.

**Database Indexing**: B-trees and B+ trees enable fast data retrieval in databases by maintaining sorted data in a balanced tree structure.

**Expression Parsing**: Abstract Syntax Trees (ASTs) represent mathematical expressions and code, making it easy to evaluate expressions and compile programs.

**Decision Making**: Decision trees in AI/ML help classify data by creating branching paths based on feature values, widely used in machine learning algorithms.

Task 6:

 Create a binary search operation on tree

Hint:

Create a node

Class for binary search

Solution 👍

class TreeNode {

    int item;

    TreeNode left, right;

    TreeNode(int item) {

        item = item;

        left = right = null;

    }

}

class BinarySearchTreeOp02 {

    TreeNode root;

    public BinarySearchTreeOp02() {

        this.root = null;

    }

    public TreeNode search(int key) {

        TreeNode current = root;

        while (current != null) { // key 30    current 50 == root

            if (key == current.item) {

                return current;

            } else if (key < current.item) { // key 80    current 50 == root

                current = current.left;

            } else {

                current = current.right;

            }

        }

        return null;

    }

}

Task 8:

Types of binary trees:

Main types of binary trees:

**Full Binary Tree**: Every node has either 0 or 2 children (no node has exactly 1 child). Also called "proper" or "strict" binary tree.

**Complete Binary Tree**: All levels are filled except possibly the last, and the last level is filled from left to right. Used in heap implementations.

**Perfect Binary Tree**: All internal nodes have exactly 2 children and all leaves are at the same level. Has 2^h - 1 nodes for height h.

**Balanced Binary Tree**: Height difference between left and right subtrees is at most 1 for every node. Examples include AVL trees and Red-Black trees.

Task 9:

Applications of Graphs

Key applications of graphs in computer science:

**Social Networks**: Represent users as nodes and friendships/connections as edges. Used for friend recommendations, influence analysis, and community detection.

**Maps & Navigation**: Cities/locations as nodes, roads as edges. GPS systems use shortest path algorithms like Dijkstra's to find optimal routes.

**Computer Networks**: Devices as nodes, connections as edges. Used for routing protocols, network topology design, and analyzing connectivity.

**Web Pages & Search**: Web pages as nodes, hyperlinks as edges. Search engines use PageRank algorithm to rank pages based on graph structure.

Task 10:

Types of Graphs:

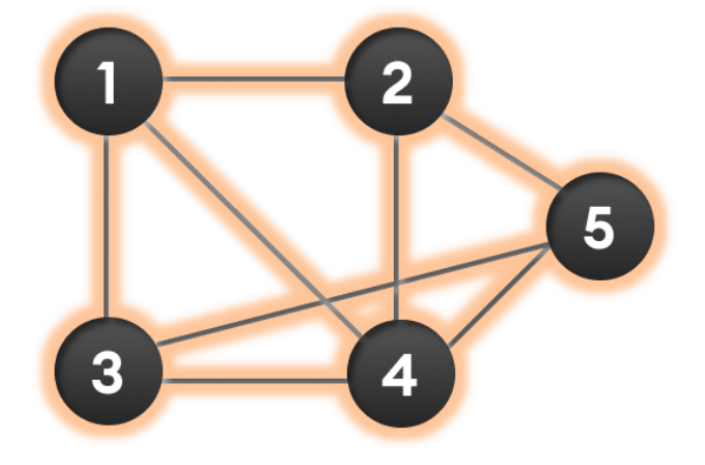
Main types of graphs:

**Directed vs Undirected**: Directed graphs have edges with direction (one-way), while undirected graphs have bidirectional edges. Examples: Twitter follows (directed) vs Facebook friends (undirected).

**Weighted vs Unweighted**: Weighted graphs assign values to edges (like distance, cost), unweighted graphs treat all edges equally. Used in shortest path problems.

**Connected vs Disconnected**: Connected graphs have a path between every pair of vertices, disconnected graphs have isolated components or vertices.

**Cyclic vs Acyclic**: Cyclic graphs contain cycles (closed paths), acyclic graphs don't. DAGs (Directed Acyclic Graphs) are used in scheduling and dependency resolution.



Wap to display a graph edges .in the below order no od edges 8 and no of vertex 5

1 - 2

1 - 3

1 - 4

2 - 4

2 - 5

3 - 4

3 - 5

4 - 5

Hint:

Class Graph{

Class Edge{

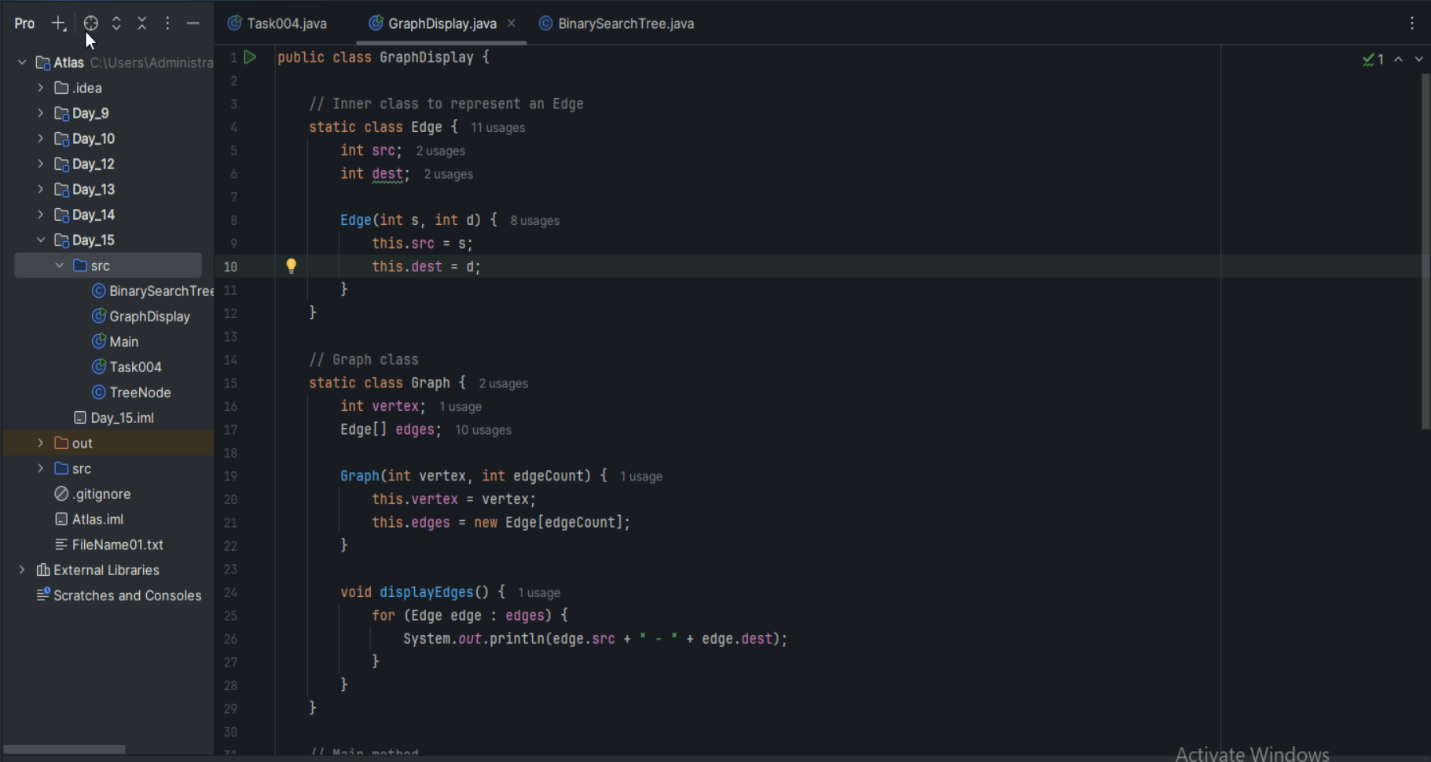
Int start/src;

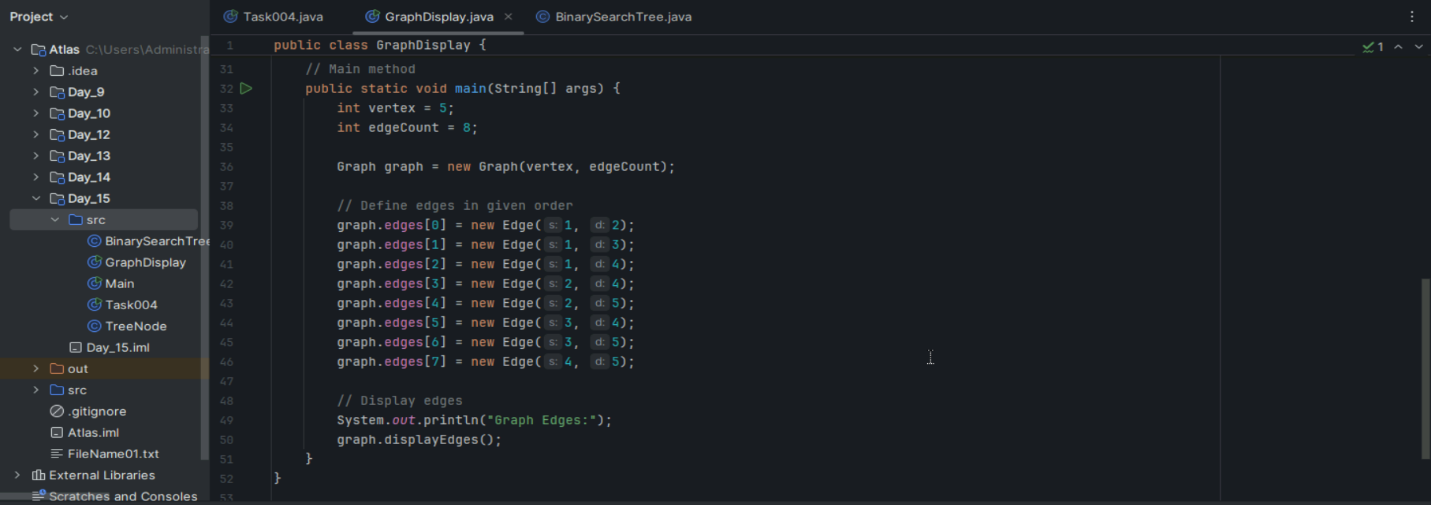
Int end/dest;

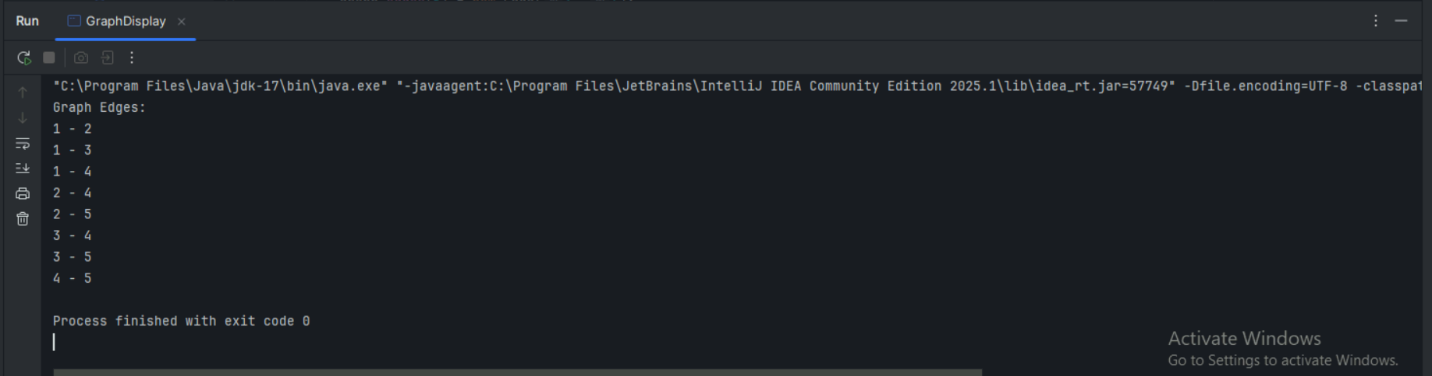
}

Int vertex;

}







class Graph01 {  
  class Edge {  
      int src, dest;  
  }  
  int vertices, edges;

  Edge[] edge;

  Graph(int vertices, int eges) {  
    this.vertices = vertices;  
        this.edges = edges;

    edge = new Edge[edges];  
    for(int i = 0; i < edges; i++) {  
      edge[i] = new Edge();  
    }  
  }  
  
  public static void main(String[] args) {  
    int noVertices = 5;  
       int noEdges = 8;  
        Graph gObj = new Graph(noVertices, noEdges);  
  
    gObj.edge[0].src = 1;     
        gObj.edge[0].dest = 2;  
       gObj.edge[1].src = 1;    
        gObj.edge[1].dest = 3;  
        gObj.edge[2].src = 1;    
        gObj.edge[2].dest = 4;  
    gObj.edge[3].src = 2;   
    gObj.edge[3].dest = 4;  
    gObj.edge[4].src = 2;   
    gObj.edge[4].dest = 5;  
    gObj.edge[5].src = 3;   
    gObj.edge[5].dest = 4;  
    gObj.edge[6].src = 3;  
    gObj.edge[6].dest = 5;  
    gObj.edge[7].src = 4;    
    gObj.edge[7].dest = 5;  
  
  for(int i =0; i < noEdges; i++) {  
    System.out.println(gObj.edge[i].src+ " - " + gObj.edge[i].dest);  
  }  
    }  
}

===============================================================================================================================================

Graphs

===============================================================================================================================================

================================================================================================================================================

Home Task 👍

Pre order and post order traversal..

Home Tasks:

 Recursion: assigned on Day 14th 02nd July 2025..

Wap to find the factorial of a number

Wap to find the Fibonacci series of a number

What is the difference between recursion and iteration

Wap to reverse a string using recursion..

Assigned on Day 14 - 03rd july 2025

Carry forward examples till 10 th july

1. Write a recursive function to search for an element in an array

2. Write a recursive function to count the digits of a positive integer (do also for sum of digits)

3. Write a recursive function to reverse a null-terminated string

4. Write a recursive function to convert a decimal number to binary

5. Write a recursive function to check if a string is a palindrome or not

6. Write a recursive function to copy one array to another

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**info box**

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

<https://drive.google.com/drive/folders/1LwhNov1s1-vHzF9GPAObLSnP9kAvipmw?usp=drive_link>

Binary tree delete op ..along with insertion and traversal inorder …

<https://www.codecademy.com/learn/graph-data-structures-java/modules/graphs-java/cheatsheet>

Graphs  in data structures.. Along with code – juz for reference..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*