**1.0-1 knapsack problem**

**CODE:**

public class KnapSack {

    static int knapsackmethod(int s\_w, int weight[], int value[], int n)

    {

        int i, w;

        int K[][] = new int[n + 1][s\_w + 1];

        for (i = 0; i <= n; i++) {

            for (w = 0; w <= s\_w; w++) {

                if (i == 0 || w == 0)

                    K[i][w] = 0;

                else if (weight[i - 1] <= w)

                    K[i][w]

                        = Math.max(value

                [i - 1]

                                  + K[i - 1][w - weight[i - 1]],

                              K[i - 1][w]);

                else

                    K[i][w] = K[i - 1][w];

            }

        }

        return K[n][s\_w];

    }

    public static void main(String args[])

    {

        int profit[] = new int[] { 40, 120, 230 };

        int weight[] = new int[] { 20, 60, 70 };

        int s\_w = 25;

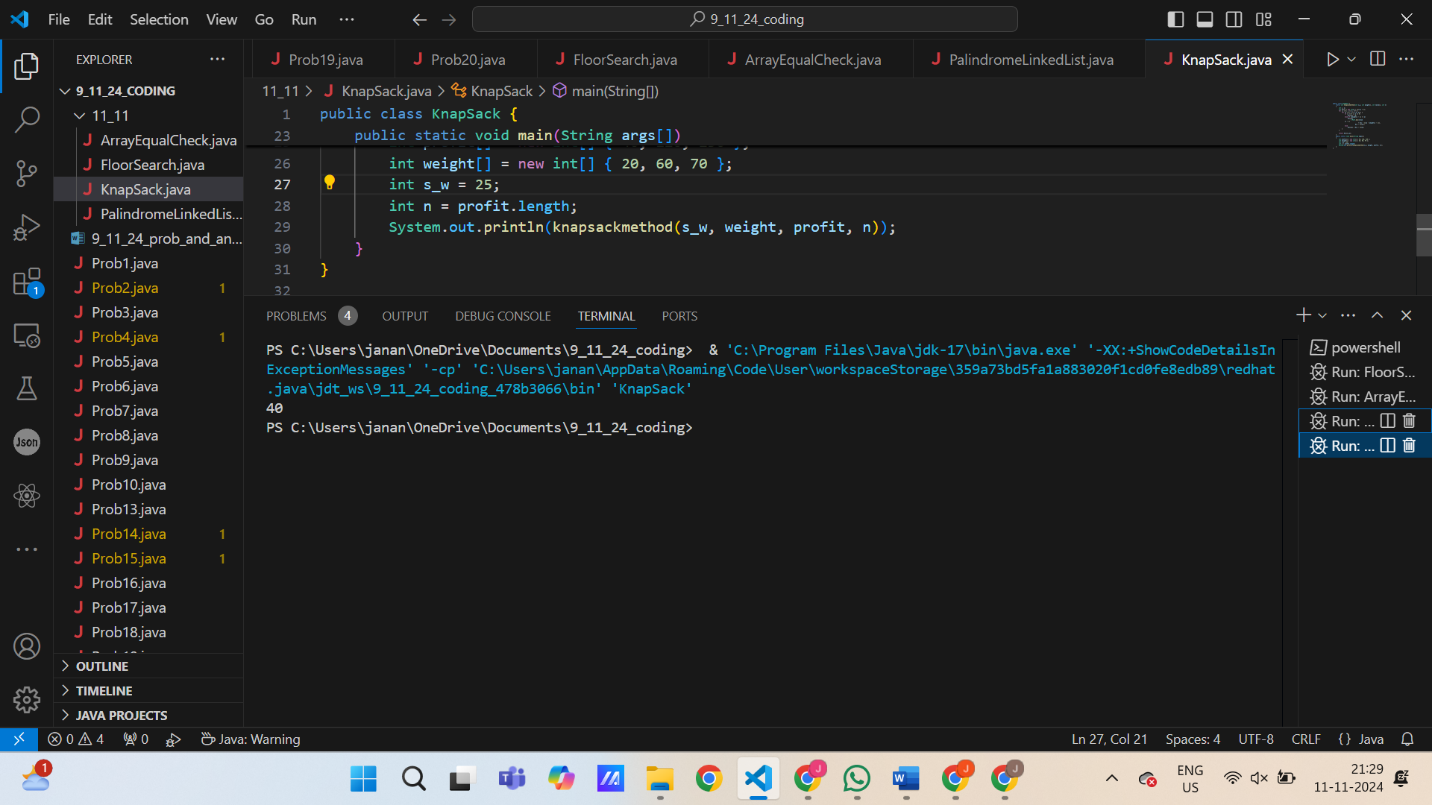
        int n = profit.length;

        System.out.println(knapsackmethod(s\_w, weight, profit, n));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:**  O(n\*w) **SPACE COMPLEXITY:** O(n\*w)

**2. Floor in sorted array**

**CODE:**

public class FloorSearch {

    int search(int[] arr,int x){

        int low=0;

        int high=arr.length-1;

        int mid;

        while(low<=high){

            mid=(low+high)/2;

            if(arr[mid]==x){

                return arr[mid];

            }

            else if(x>=arr[high]){

                return arr[high];

            }

            else if(mid>0 && arr[mid]>x && arr[mid-1]<x){

                return arr[mid-1];

            }

            else if(x>arr[mid]){

                low=mid+1;

            }

            else if(x<arr[mid]){

                high=mid-1;

            }

        }

        return -1;

    }

    public static void main(String[] args) {

        FloorSearch f=new FloorSearch();

        int[] arr1={3,5,6,8,10};

        int x1=7;

        int[] arr2={6,8,9,12,14,15};

        int x2=3;

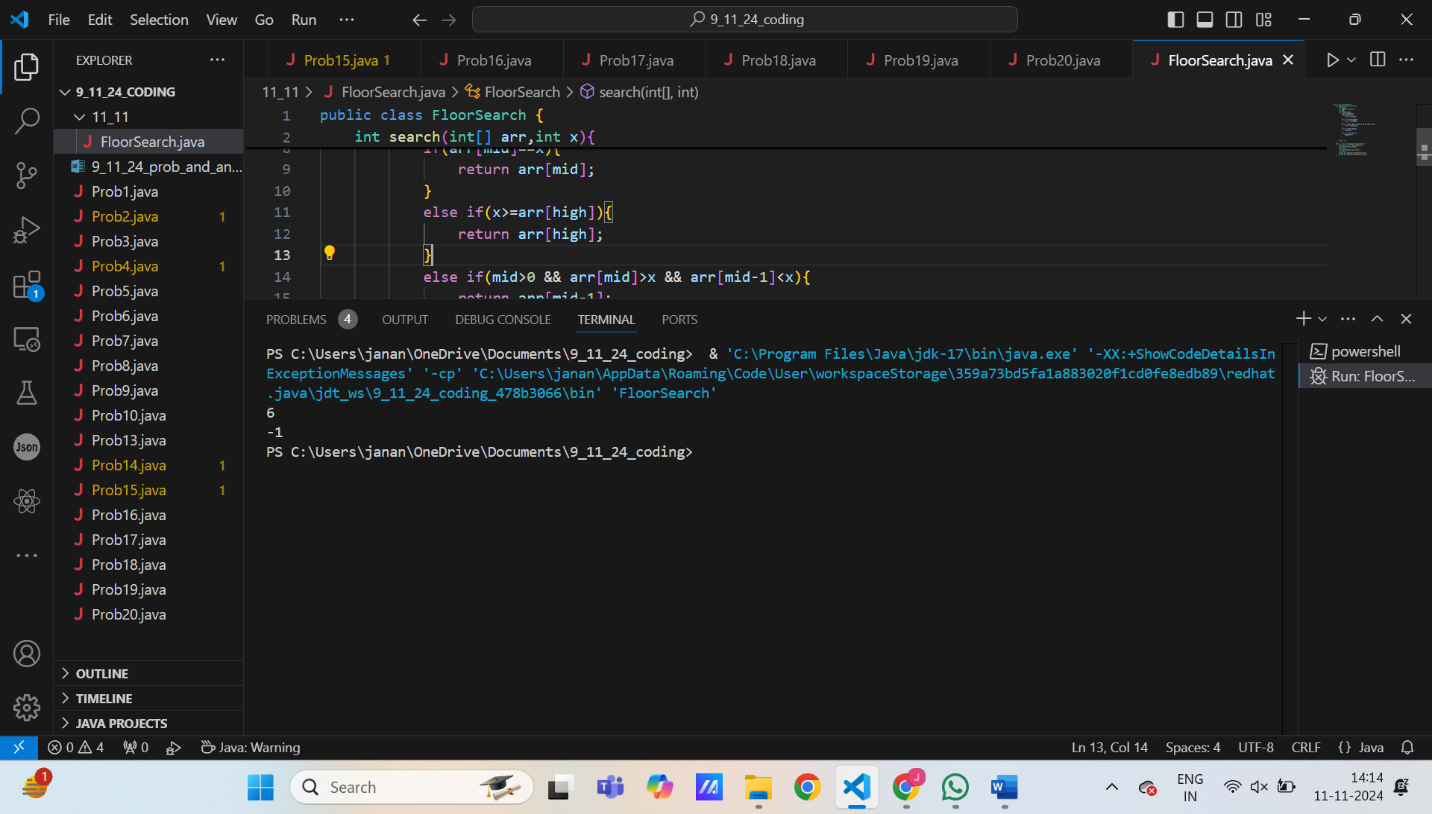
        System.out.println(f.search(arr1,x1));

        System.out.println(f.search(arr2,x2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(log n)

**SPACE COMPLEXITY:** O(1)

**3. Check equal arrays**

**CODE:**

import java.util.HashMap;

public class ArrayEqualCheck {

    boolean check(int[] arr1,int[] arr2){

        HashMap<Integer,Integer> h1= new HashMap<>();

        HashMap<Integer,Integer> h2= new HashMap<>();

        int n1=arr1.length;

        int n2=arr2.length;

        if(n1!=n2){

            return false;

        }

        for (int i=0;i<n1;i++){

            h1.put(arr1[i],h1.getOrDefault(h1, 0)+1);

        }

        for (int i=0;i<n2;i++){

            h2.put(arr2[i],h2.getOrDefault(h2, 0)+1);

        }

        return h1.equals(h2);

    }

    public static void main(String[] args) {

        ArrayEqualCheck a=new ArrayEqualCheck();

        int[] arr1={1,2,3,4,5,6,7};

        int[] arr2={7,6,5,4,3,2,1};

        int[] arr3={1,2,3};

        int[] arr4={12,3};

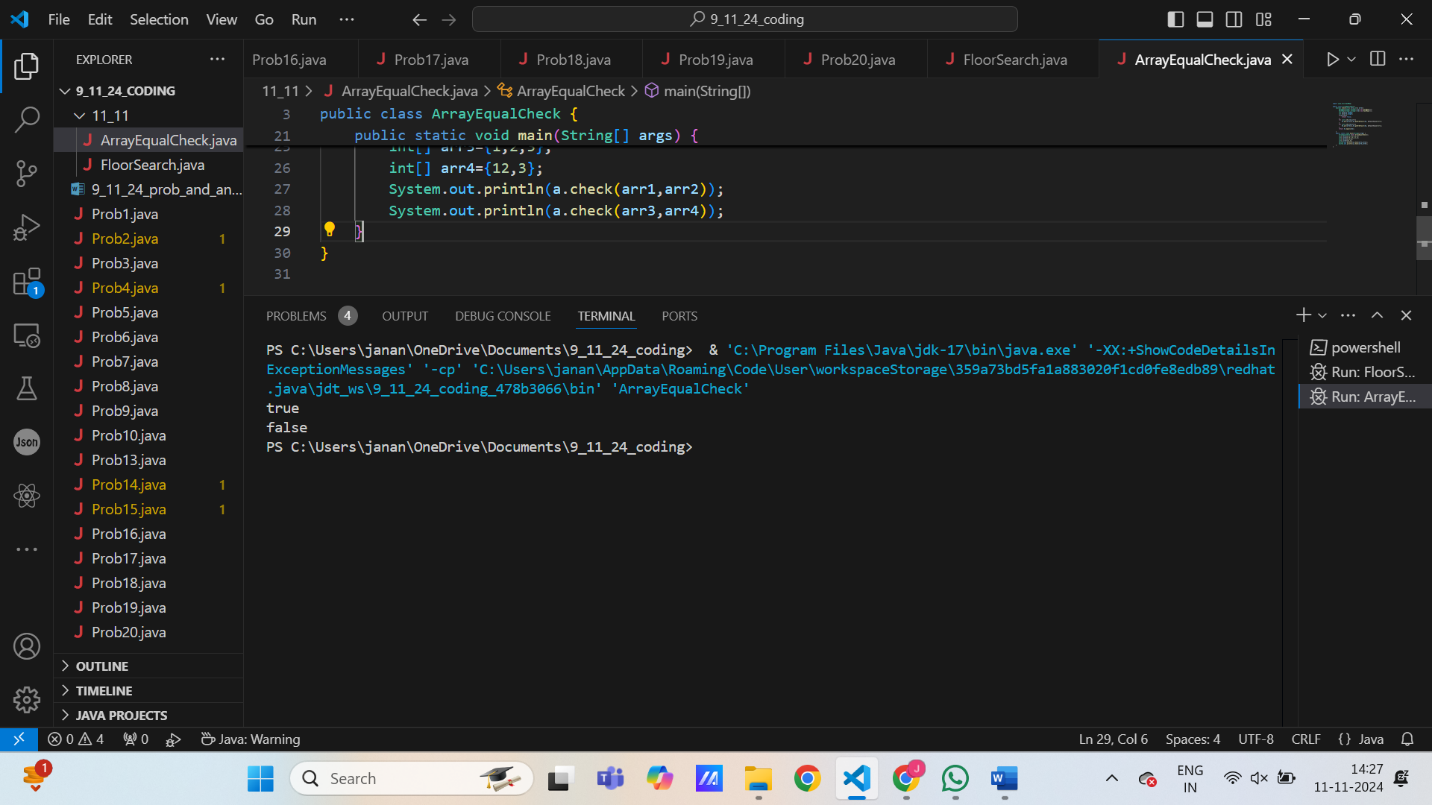
        System.out.println(a.check(arr1,arr2));

        System.out.println(a.check(arr3,arr4));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(n)

**3. Palindrome linked list**

**CODE:**

public class PalindromeLinkedList {

    public class ListNode {

        int val;

        ListNode next;

        ListNode() {}

        ListNode(int val) {

            this.val = val;

        }

        ListNode(int val, ListNode next) {

            this.val = val;

            this.next = next;

        }

    }

    public ListNode reverse(ListNode head) {

        ListNode prev = null;

        ListNode curr = head;

        while (curr != null) {

            ListNode next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

    public boolean isPalindrome(ListNode head) {

        ListNode first = head;

        ListNode second = head.next;

        while (second != null && second.next != null) {

            first = first.next;

            second = second.next.next;

        }

        ListNode rev = reverse(first.next);

        first.next = null;  // Split the list into two halves

        while (rev != null) {

            if (head.val != rev.val) {

                return false;

            }

            head = head.next;

            rev = rev.next;

        }

        return true;

    }

    public static void main(String[] args) {

        PalindromeLinkedList p = new PalindromeLinkedList();

        PalindromeLinkedList.ListNode head = p.new ListNode(1);

        head.next = p.new ListNode(2);

        head.next.next = p.new ListNode(3);

        head.next.next.next = p.new ListNode(2);

        head.next.next.next.next = p.new ListNode(1);

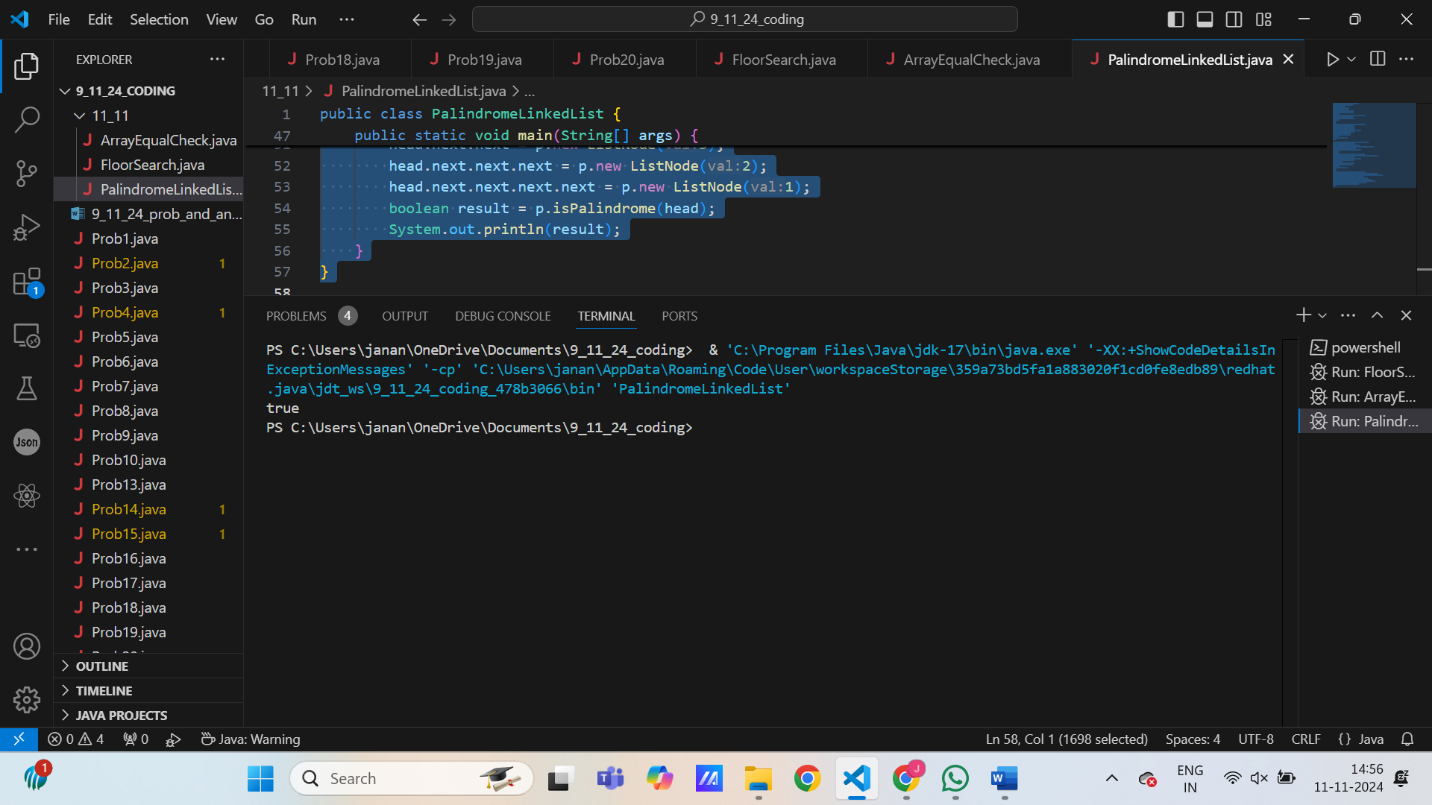
        boolean result = p.isPalindrome(head);

        System.out.println(result);

    }

}

**OUTPUT:**



**5. Balanced tree check**

**CODE:**

class Node {

    int key;

    Node left;

    Node right;

    Node(int k)

    {

        key = k;

        left = right = null;

    }

}

class Main {

    public static int BalanceCheck(Node root)

    {

        if (root == null)

            return 0;

        int lh = BalanceCheck(root.left);

        if (lh == -1)

            return -1;

        int rh = BalanceCheck(root.right);

        if (rh == -1)

            return -1;

        if (Math.abs(lh - rh) > 1)

            return -1;

        else

            return Math.max(lh, rh) + 1;

    }

    public static void main(String args[])

    {

        Node root = new Node(1);

        root.left = new Node(2);

        root.right = new Node(3);

        root.right.left = new Node(4);

        root.right.right = new Node(5);

        if (BalanceCheck(root) > 0)

            System.out.print("Balanced");

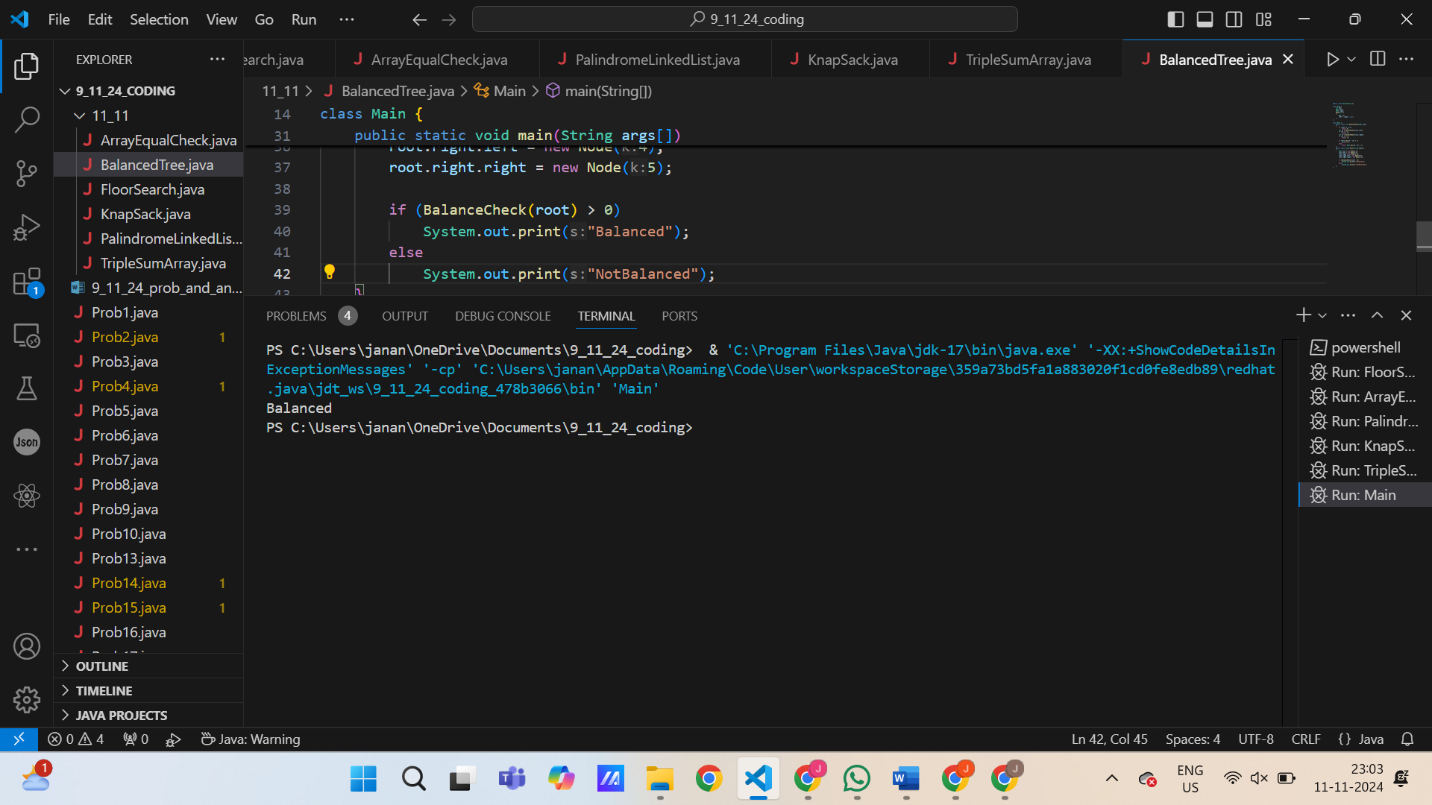
        else

            System.out.print("NotBalanced");

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n) **SPACE COMPLEXITY:** O(h)

**6. Triplet sum in array**

**CODE:**

import java.util.\*;

public class TripleSumArray{

    boolean findSum(int arr[], int n, int x) {

        Arrays.sort(arr);

        for(int i=0;i<n;i++){

            int left=i+1;

            int right=n-1;

       while(left<right){

           int sum=arr[i]+arr[left]+arr[right];

           if(x==sum)

           {

               return true;

           }

           else if(x>sum){

               left++;

           }

           else{

               right--;

           }

       }

       }

       return false;

        }

        public static void main(String args[]){

            TripleSumArray t=new TripleSumArray();

            int arr1[]={3,67,8,56,34,16};

            int n1=6;

            int x1=27;

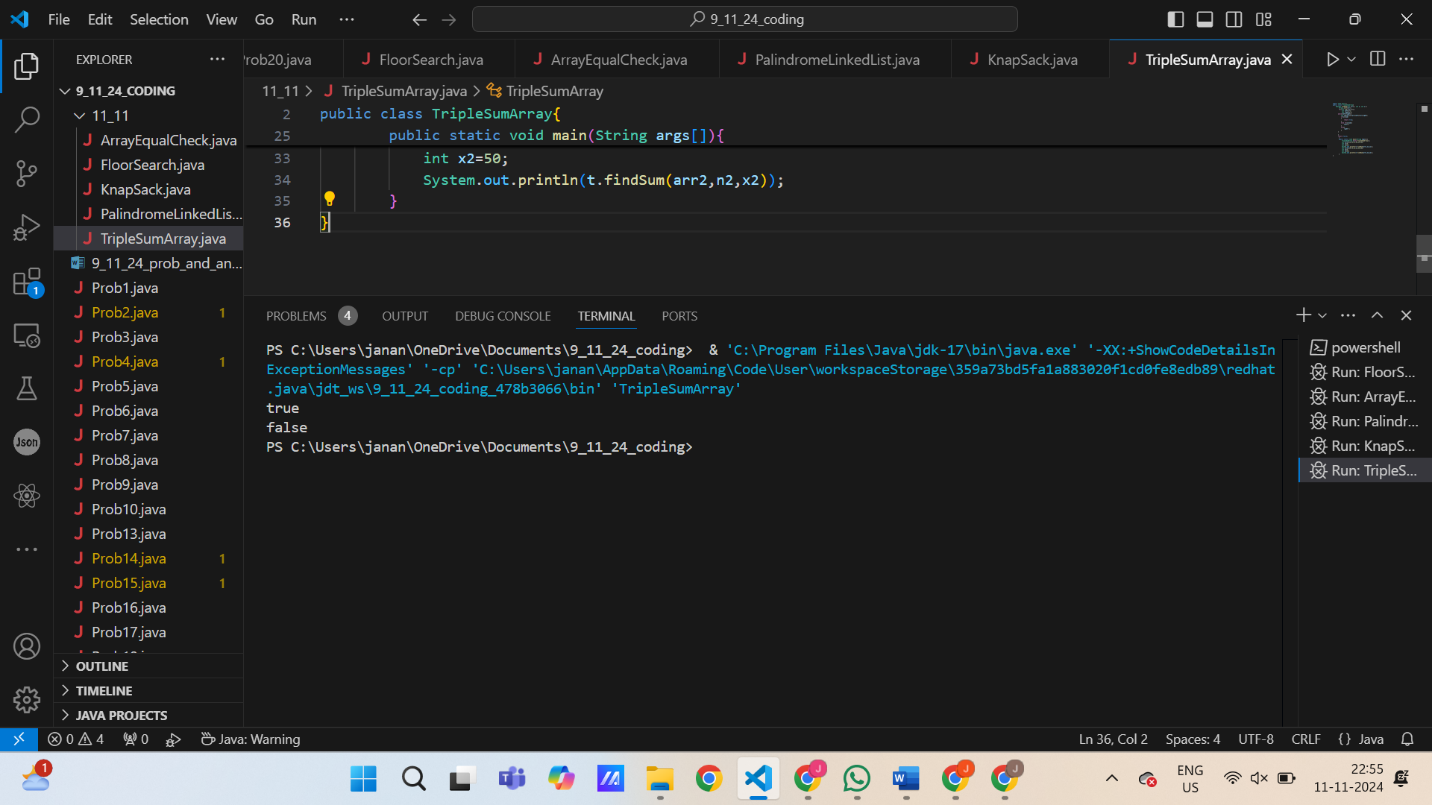
            System.out.println(t.findSum(arr1,n1,x1));

            int arr2[]={7,9,12,12,34,16};

            int n2=4;

            int x2=50;

            System.out.println(t.findSum(arr2,n2,x2)); }}

**OUTPUT:** 

**TIME COMPLEXITY:** O(n^2) **SPACE COMPLEXITY:** O(1)