**Name : Murali srinivasan S  
Department : CSE  
Test : Practice\_set\_ 02.**

**1.** (0-1) Knapsack problem.  
  
**Code :  
  
import java.io.\*;**

**class Example {  
 static int knapSackRec(int W, int wt[], int val[], int n, int[][] dp) {**

**if (n == 0 || W == 0)**

**return 0;**

**if (dp[n][W] != -1)**

**return dp[n][W];**

**if (wt[n - 1] > W)**

**return dp[n][W] = knapSackRec(W, wt, val, n - 1, dp);**

**else**

**return dp[n][W] = Math.max((val[n - 1] + knapSackRec(W - wt[n - 1], wt, val, n - 1, dp)), knapSackRec(W, wt, val, n - 1, dp));**

**}**

**static int knapSack(int W, int wt[], int val[], int N) {**

**int dp[][] = new int[N + 1][W + 1];**

**for (int i = 0; i < N + 1; i++)**

**for (int j = 0; j < W + 1; j++)**

**dp[i][j] = -1;**

**return knapSackRec(W, wt, val, N, dp);**

**}**

**public static void main(String[] args) {**

**int profit[] = { 60, 100, 120 };**

**int weight[] = { 10, 20, 30 };**

**int W = 50;**

**int N = profit.length;**

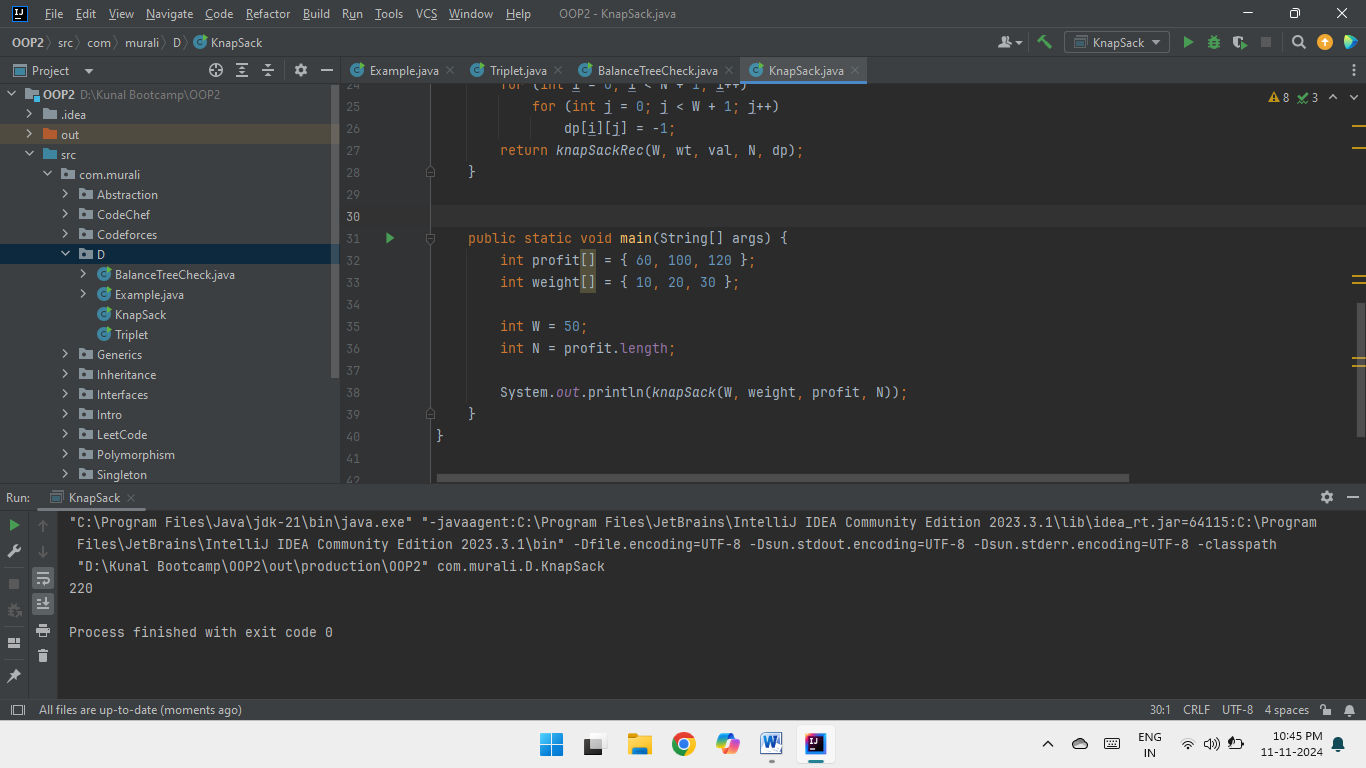
**System.out.println(knapSack(W, weight, profit, N));**

**}**

**}**

**Complexities :**Time Complexity : O(n\*w)

Space Complexity : O(n\*w) + O(n)

**Output :**  
  
**2.** Find the Floor of a given number in a sorted array.  
  
**Code :  
  
import java.util.Arrays;**

**class Example {**

**public static void main(String[] args) {**

**int[] arr = {2,3,5,7,9};**

**int val = 6;**

**System.out.println(floorIndex(arr, val));**

**}**

**public static int floorIndex(int[] arr, int val){**

**/\* arr = {2,3,5,7,9} , val = 6 \*/**

**int s = 0;**

**int e = arr.length - 1;**

**while(s<=e){**

**int mid = s + (e - s)/2;**

**if(arr[mid] < val){**

**s = mid + 1;**

**} else if(arr[mid] > val){**

**e = mid - 1;**

**} else{**

**return mid;**

**}**

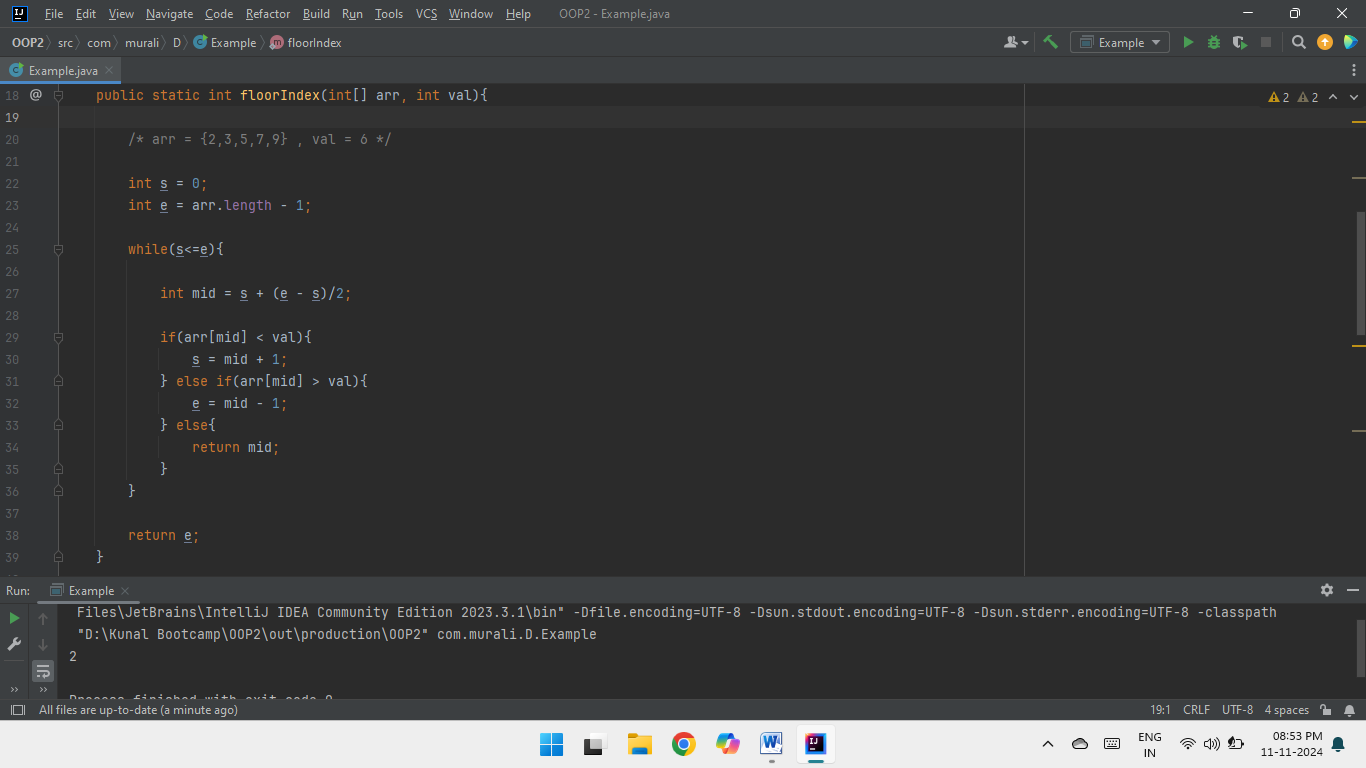
**}**

**return e;**

**}  
}**

**Complexities :**Time Complexity : O(log n)

Space Complexity : O(1)

**Output :**  
  
**3.** Check equals array.  
  
**Code :  
import java.util.Arrays;**

**class Example {**

**public static void main(String[] args) {**

**int[] f = {1,7,1};**

**int[] s = {7,7,1};**

**System.out.println(checkEqualsArray(f,s));**

**}**

**public static boolean checkEqualsArray(int[] f, int[] s){**

**/\* f = {1,7,1} , s = {7,7,1} \*/**

**if(f.length != s.length) { return false; }**

**Arrays.sort(f);**

**Arrays.sort(s);**

**for(int i=0; i<f.length; i++){**

**if(f[i] != s[i]){**

**return false;**

**}**

**}**

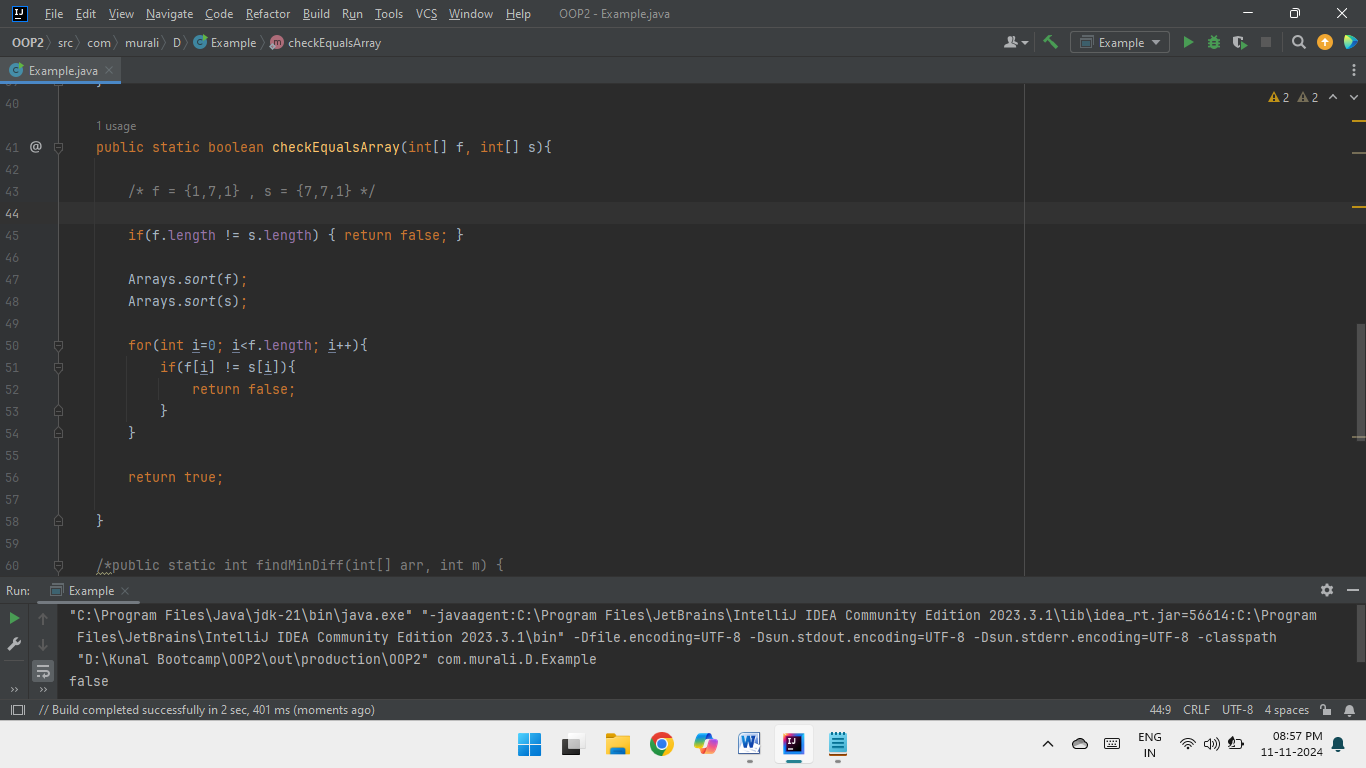
**return true;**

**}**

**}**

**Complexities :**Time Complexity : O(n + n log n)

Space Complexity : O(1)

**Output :**   
  
**4.** Palindrome linked list  
  
**Code :**

**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; }**

**}**

**class Example {  
  
 public static void main(String[] args) {**

**/\* Hard-coded singly linked list : 1 -> 2 -> 2 -> 1 \*/**

**ListNode head = new ListNode(1);**

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

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

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

**System.out.println(isPalindrome(head)==true ? "Palindrome" : "Non-Palindrome");**

**}**

**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 slow = head;**

**ListNode fast = head.next;**

**while(fast != null && fast.next != null) {**

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**ListNode rev = reverse(slow.next); // reverse second list**

**slow.next = null;**

**while(rev != null) {**

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

**return false;**

**}**

**head = head.next;**

**rev = rev.next;**

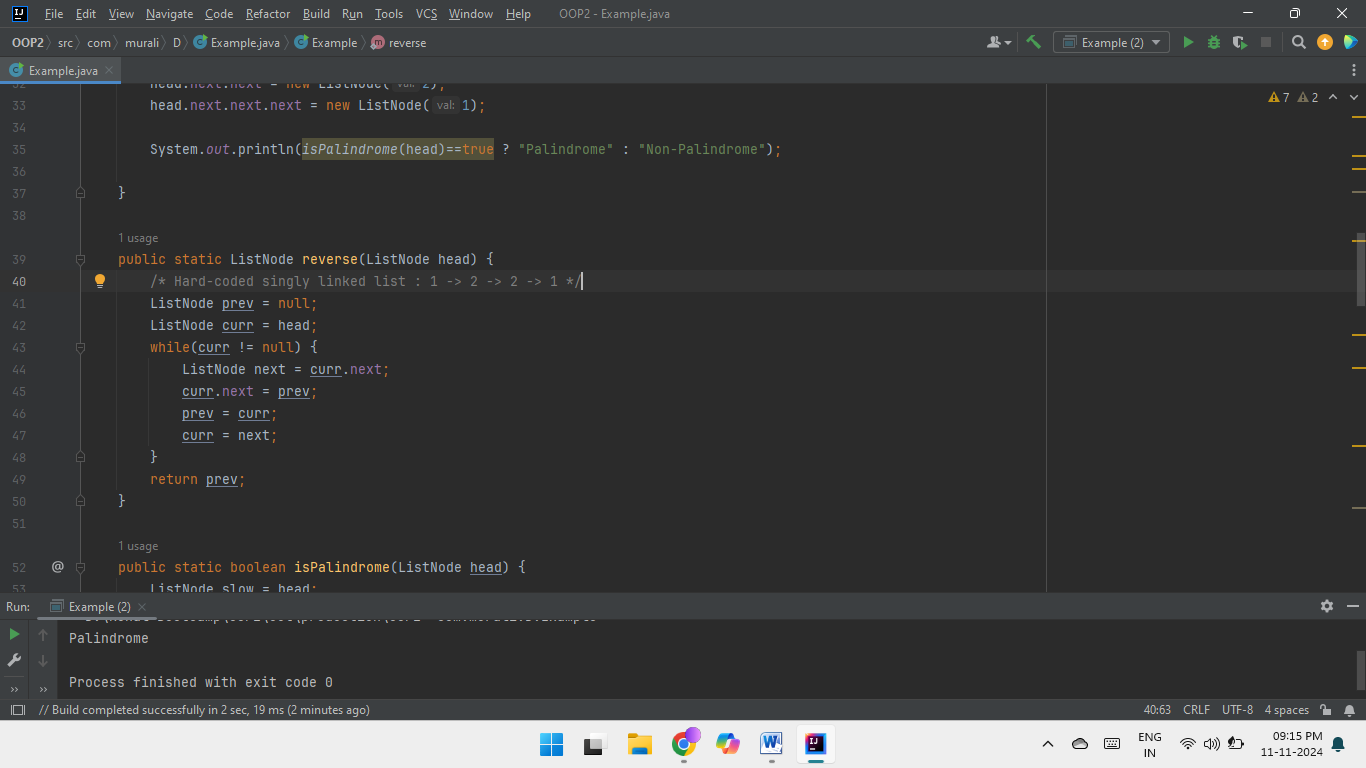
**}**

**return true;**

**}**

**}**

**Complexities :**Time Complexity : O(n)

Space Complexity : O(1)  
  
  
  
  
**Output :**  
  
**5.** Balanced tree Check  
  
**Code :**

**public class TreeNode {**

**int val;**

**TreeNode left;**

**TreeNode right;**

**TreeNode() {}**

**TreeNode(int val) { this.val = val; }**

**TreeNode(int val, TreeNode left, TreeNode right) {**

**this.val = val;**

**this.left = left;**

**this.right = right;**

**}**

**}**

**class Example {**

**int height = 0;**

**public boolean isBalanced(TreeNode root) {**

**if(root == null) return true;**

**if(Math.abs(height(root.left) - height(root.right)) >1) return false;**

**return isBalanced(root.left) && isBalanced(root.right);**

**}**

**public int height(TreeNode node) {**

**if(node == null) return 0;**

**return 1 + Math.max(height(node.left),height(node.right));**

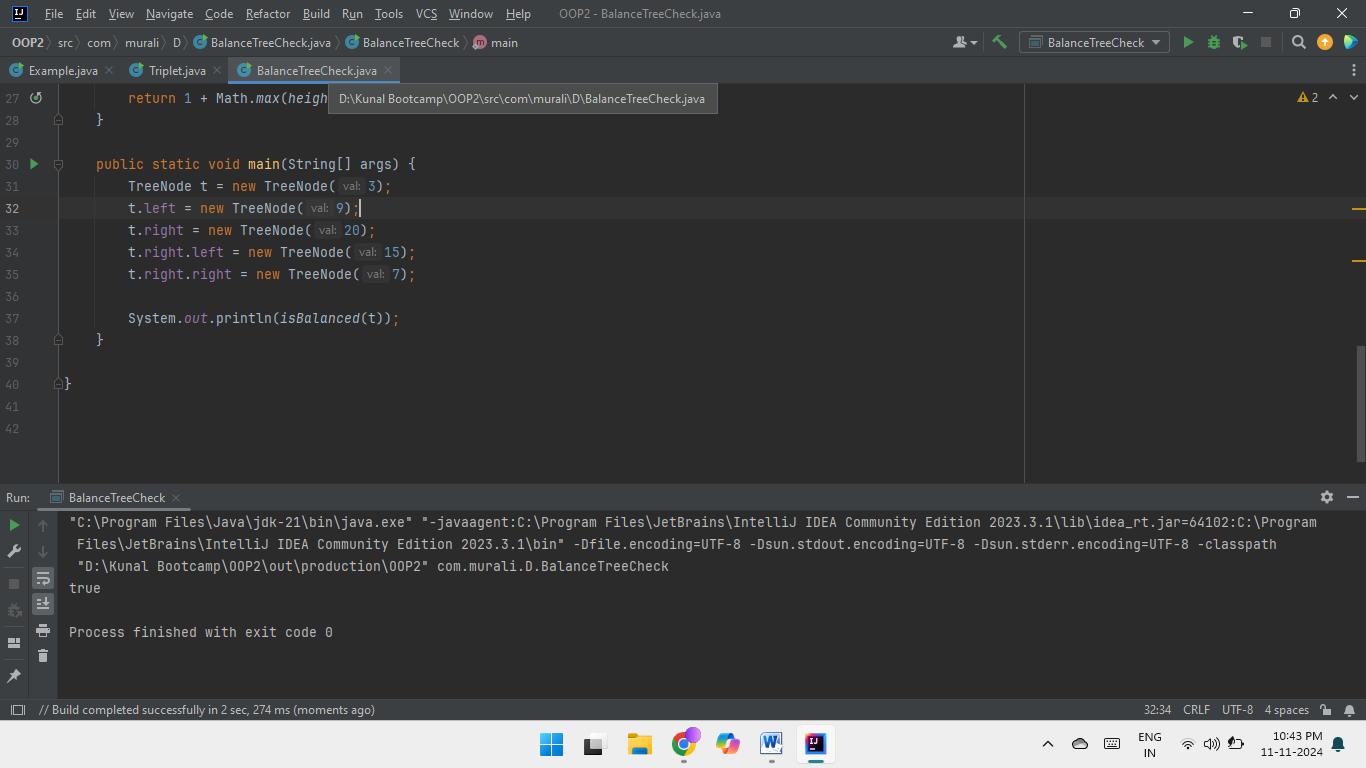
**}**

**}**

**Complexities :**Time Complexity : O(n)

Space Complexity : O(1)

**Output :**

  
  
**6.** Triplet sum in array  
  
**Code :  
  
import java.util.Arrays;**

**public class Example {**

**static boolean find3Numbers(int[] arr, int sum)**

**{**

**int n = arr.length;**

**Arrays.sort(arr);**

**for (int i = 0; i < n - 2; i++) {**

**int l = i + 1;  
 int r = n - 1;**

**while (l < r) {**

**int curr\_sum = arr[i] + arr[l] + arr[r];**

**if (curr\_sum == sum) {**

**System.out.println("Triplet is " + arr[i] + ", "+ arr[l] + ", " + arr[r]);**

**return true;**

**}**

**else if (curr\_sum < sum) {**

**l++;**

**}**

**else {**

**r--;**

**}**

**}**

**}**

**return false;**

**}**

**public static void main(String[] args)**

**{**

**int[] arr = { 1, 4, 45, 6, 10, 8 };**

**int sum = 22;**

**find3Numbers(arr, sum);**

**}**

**}**

**Complexities :**Time Complexity : O(n^2)

Space Complexity : O(1)   
  
**Output :**