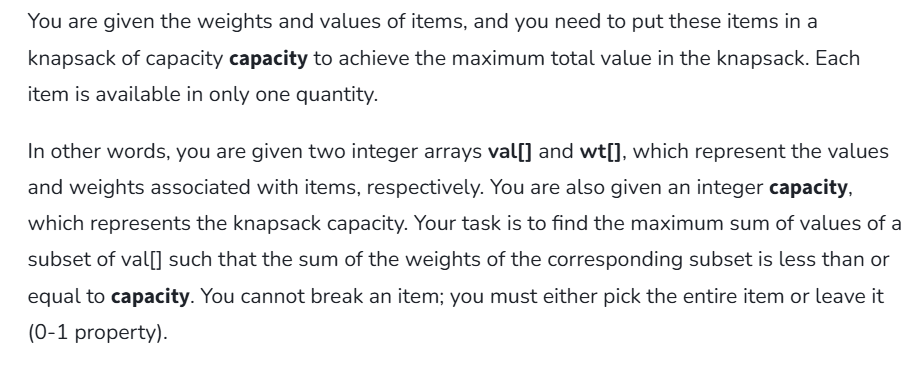
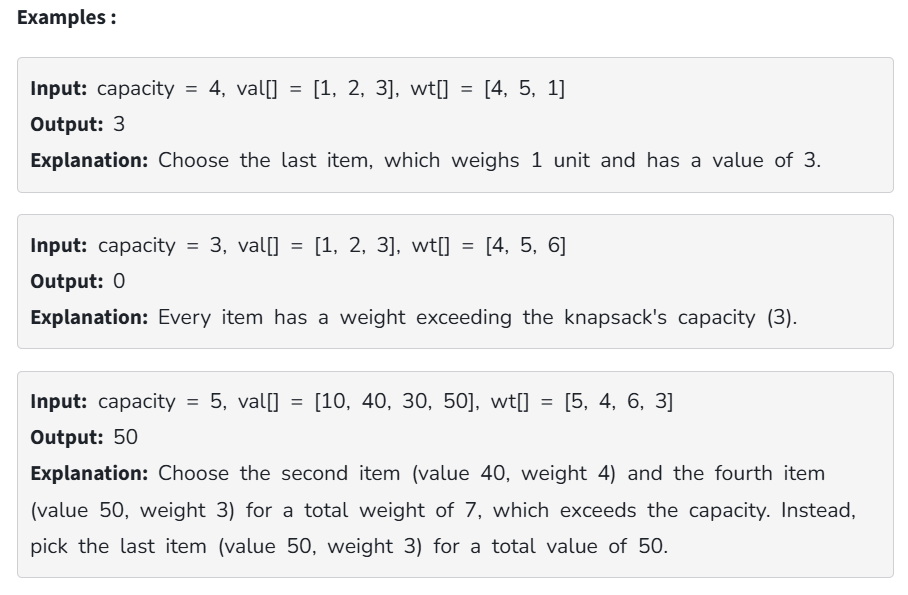
**DSA Practice Test – 2** 11th Nov 2024

**1. 0 – 1 Knapsack Problem**

****

****

**Code:**

class Solution {

static int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int K[][] = new int[n + 1][W + 1];

for (i = 0; i <= n; i++) {

for (w = 0; w <= W; w++) {

if (i == 0 || w == 0)

K[i][w] = 0;

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

K[i][w]

= Math.max(val[i - 1]

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

K[i - 1][w]);

else

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

}

}

return K[n][W];

}

public static void main(String args[])

{

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

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

int W = 50;

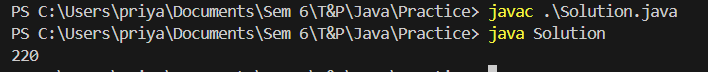
int n = profit.length;

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

}

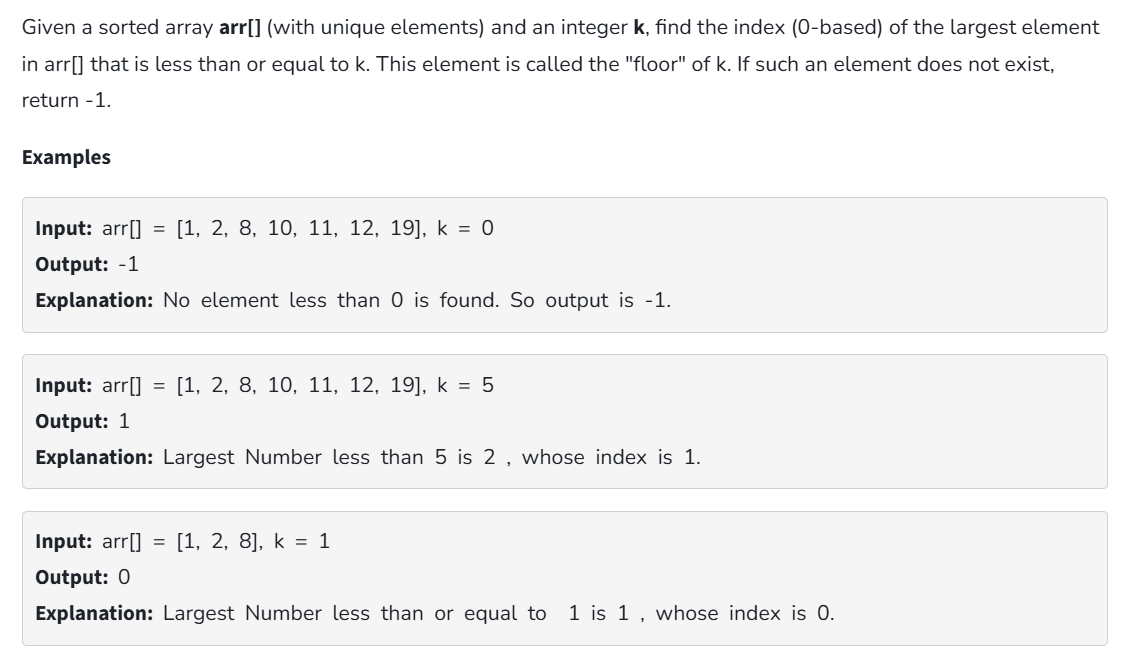
}

**Output:**



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

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

**2. Floor in Sorted Array**

**Code:**

import java.util.Scanner;

class Solution {

static int findFloor(int[] arr, int k) {

if(k<arr[0])

return -1;

int ind=0;

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

if(arr[i]<=k)

ind=i;

}

return ind;

}

public static void main (String ar[]){

Scanner sc=new Scanner(System.in);

System.out.println("Enter the size of the array");

int size=sc.nextInt();

int[] arr=new int[size];

System.out.println("Enter the array elements");

for(int i=0;i<size;i++){

arr[i]=sc.nextInt();

}

System.out.println("Enter the value of k");

int k=sc.nextInt();

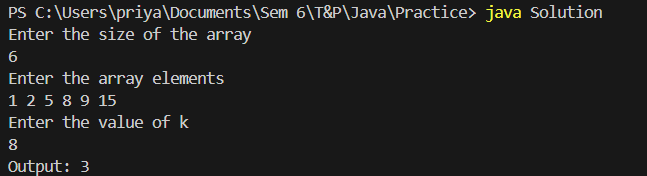
System.out.println(findFloor(arr, k));

sc.close();

}

}

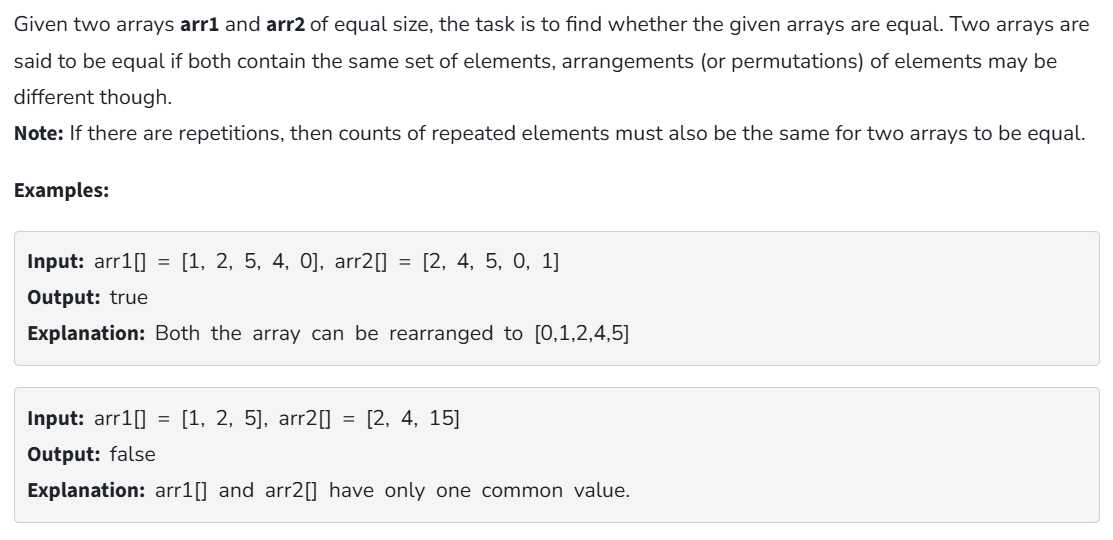
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**3. Check Equal Arrays:**



**Code:**

import java.util.Scanner;

import java.util.Arrays;

class Solution {

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

Arrays.sort(arr1);

Arrays.sort(arr2);

for(int i=0;i<arr1.length;i++){

if(arr1[i]!=arr2[i]){

return false;

}

}

return true;

}

public static void main (String ar[]){

Scanner sc=new Scanner(System.in);

System.out.println("Enter the size of the first array:");

int size=sc.nextInt();

int[] arr1=new int[size];

System.out.println("Enter the elements of the first array:");

for(int i=0;i<size;i++){

arr1[i]=sc.nextInt();

}

int[] arr2=new int[size];

System.out.println("Enter the elements of the second array:");

for(int i=0;i<size;i++){

arr2[i]=sc.nextInt();

}

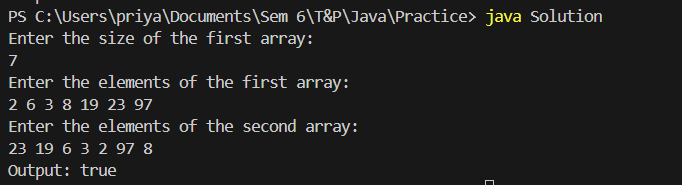
System.out.println("Output: "+check(arr1, arr2));

sc.close();

}

}

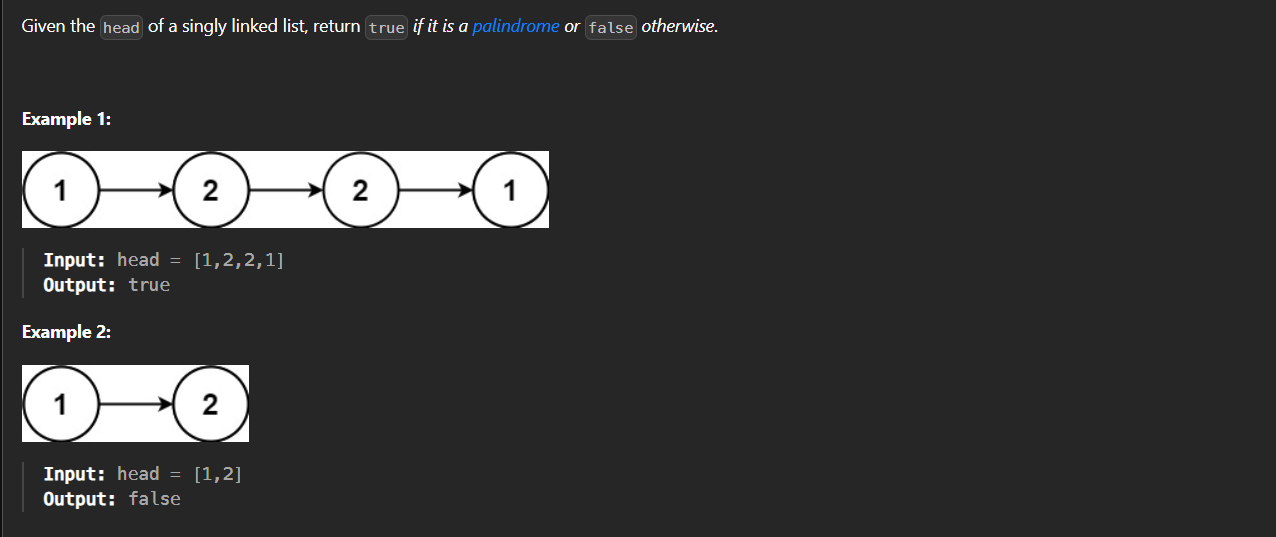
**Output:**



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

**Space Complexity:** O (1)

**4. Palindrome Linked List**



**Code:**

import java.util.Stack;

class ListNode {

int data;

ListNode next;

ListNode(int d) {

data = d;

next = null;

}

}

class Solution {

static boolean isPalindrome(ListNode head) {

ListNode currNode = head;

Stack<Integer> s = new Stack<>();

while (currNode != null) {

s.push(currNode.data);

currNode = currNode.next;

}

while (head != null) {

int c = s.pop();

if (head.data != c) {

return false;

}

head = head.next;

}

return true;

}

public static void main(String[] args) {

ListNode head = new ListNode(1);

head.next = new ListNode(2);

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

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

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

boolean result = isPalindrome(head);

if (result)

System.out.println("true");

else

System.out.println("false");

}

}

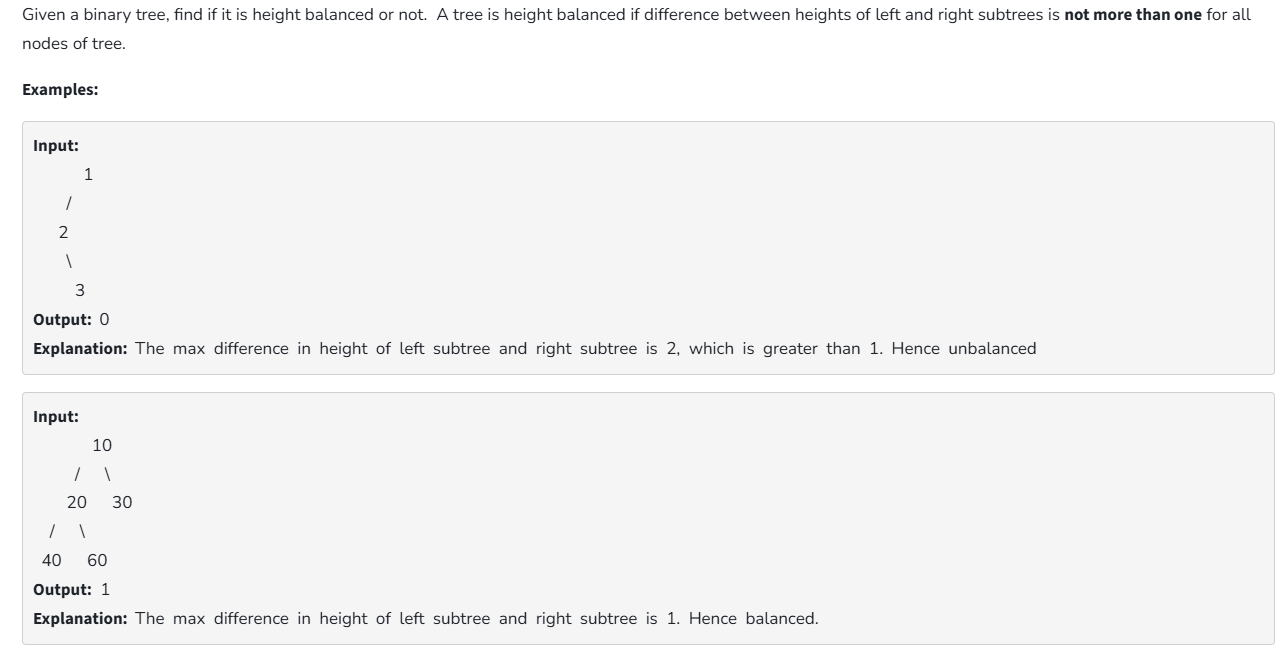
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**5. Balanced Tree Check**



**Code:**

class Node {

int data;

Node left, right;

Node(int d)

{

data = d;

left = right = null;

}

}

class Solution {

Node root;

boolean isBalanced(Node node)

{

int lh;

int rh;

if (node == null)

return true;

lh = height(node.left);

rh = height(node.right);

if (Math.abs(lh - rh) <= 1 && isBalanced(node.left)

&& isBalanced(node.right))

return true;

return false;

}

int height(Node node)

{

if (node == null)

return 0;

return 1

+ Math.max(height(node.left),

height(node.right));

}

public static void main(String args[])

{

Solution tree = new Solution();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

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

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

tree.root.left.left.left = new Node(8);

if (tree.isBalanced(tree.root))

System.out.println("Tree is balanced");

else

System.out.println("Tree is not balanced");

}

}

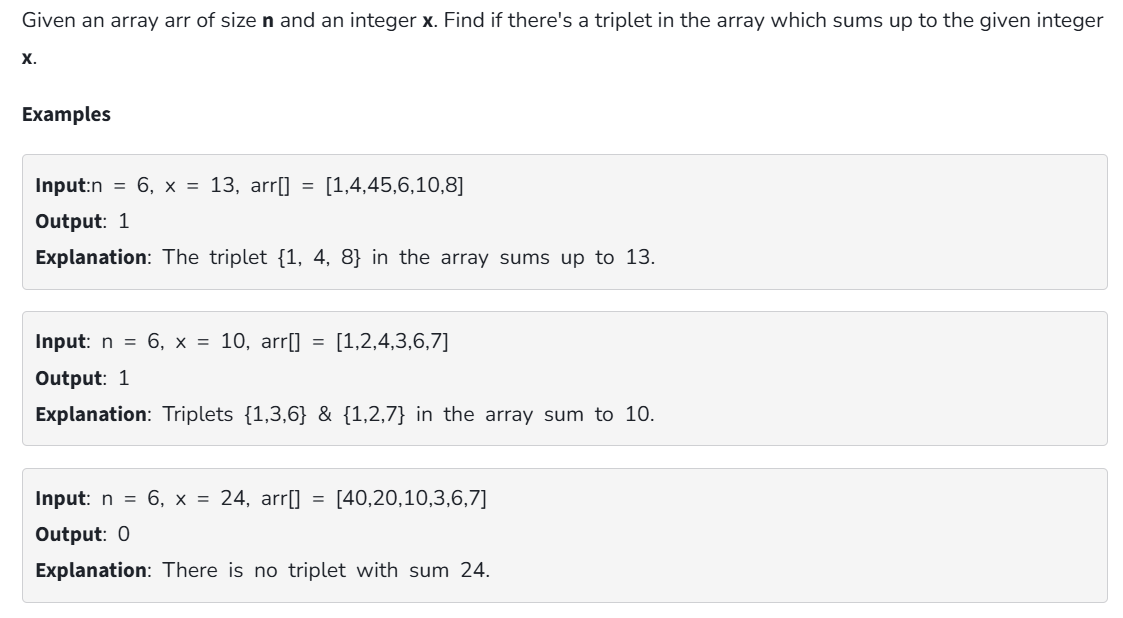
**Output:**



**Time Complexity:** O (n2)

**Space complexity:** O (n)

**6. Triplet Sum in Array**



**Code:**

import java.util.Arrays;

public class Solution {

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 };

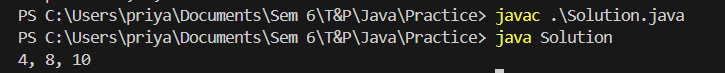
int sum = 22;

find3Numbers(arr, sum);

}

}

**Output:**



**Time Complexity:** O (n2)

**Space Complexity:** O (1)