**DSA PRACTICE DAY 2-DHANUSHSHRUTHI S T-AI&DS**

1. **0-1 knapsack problem**

**CODE:**

public class Knapsack {

public static int knapsack(int[] weights, int[] values, int capacity) {

int n = weights.length;

int[][] dp = new int[n + 1][capacity + 1];

for (int i = 1; i <= n; i++) {

for (int w = 0; w <= capacity; w++) {

if (weights[i - 1] <= w) {

dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);

} else {

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

}

}

}

return dp[n][capacity];

}

public static void main(String[] args) {

int[] weights = {4,5,1};

int[] values = {1,2,3};

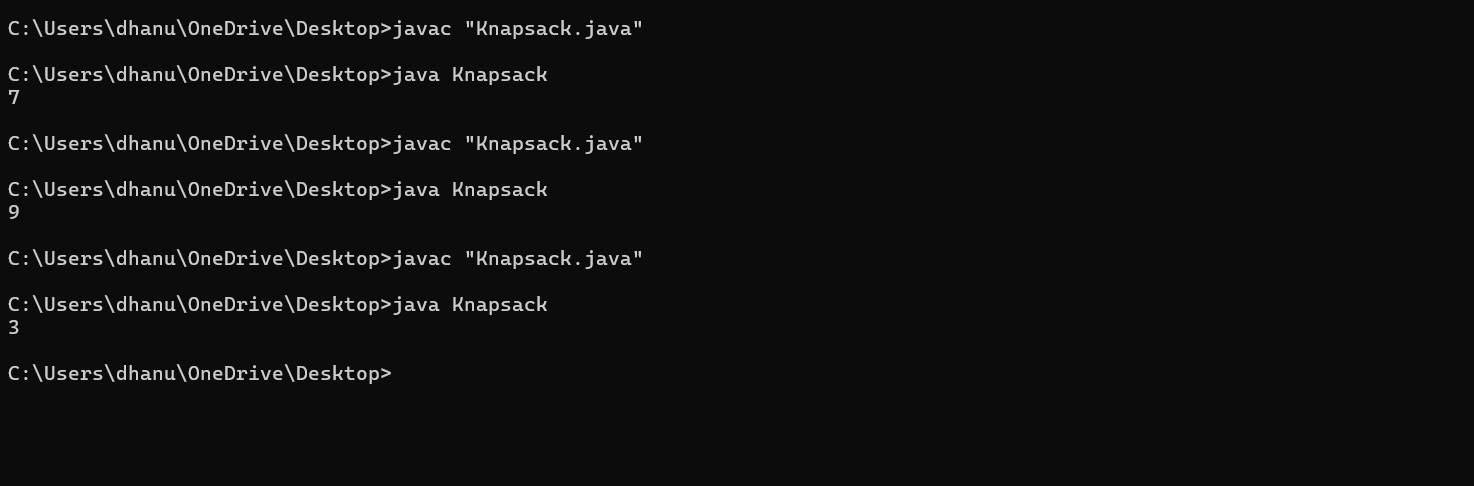
int capacity = 4;

System.out.println(knapsack(weights, values, capacity));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n×W)**

1. **Floor in sorted array**

**CODE:**

public class FloorInSortedArray {

public static int findFloor(int[] arr, int target) {

int low = 0, high = arr.length - 1;

int floor = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] <= target) {

floor = arr[mid];

low = mid + 1;

} else {

high = mid - 1;

}

}

return floor;

}

public static void main(String[] args) {

int[] sortedArray = {23,24,34,43};

int target = 4;

int result = findFloor(sortedArray, target);

if (result != -1) {

System.out.println("The floor of " + target + " is: " + result);

} else {

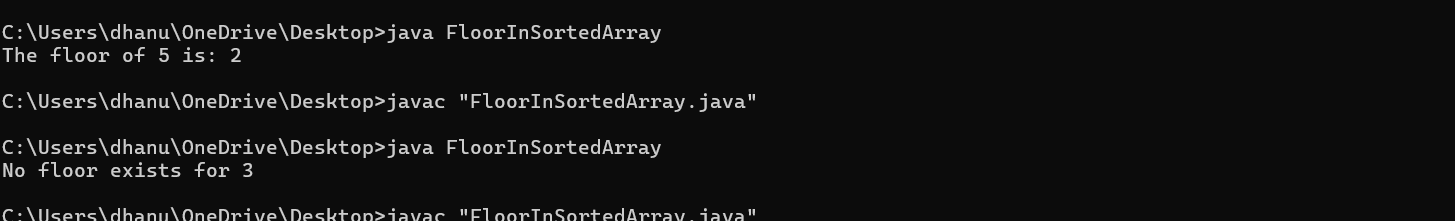
System.out.println("No floor exists for " + target);

}

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(logn)**

1. **Check equal arrays**

**CODE:**

import java.util.Arrays;

public class CheckEqualArrays {

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

if (arr1.length != arr2.length) {

return false;

}

Arrays.sort(arr1);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

public static void main(String[] args) {

int[] array1 = {1, 2, 3, 4};

int[] array2 = {4, 3, 2};

if (areArraysEqual(array1, array2)) {

System.out.println("The arrays are equal.");

} else {

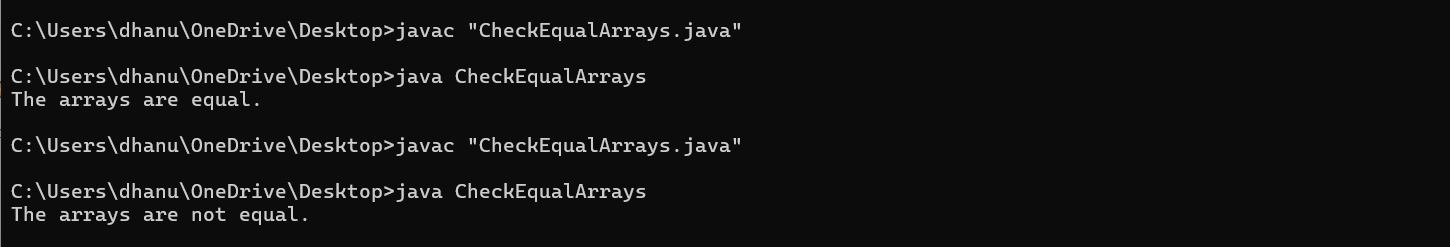
System.out.println("The arrays are not equal.");

}

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n1​logn1​+n2​logn2​)**

1. **Palindrome linked list**

**CODE:**

class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class PalindromeLinkedList {

public static boolean isPalindrome(ListNode head) {

if (head == null || head.next == null) {

return true;

}

ListNode slow = head, fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

ListNode secondHalf = reverseList(slow);

ListNode firstHalf = head;

while (secondHalf != null) {

if (firstHalf.val != secondHalf.val) {

return false;

}

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

return true;

}

private static ListNode reverseList(ListNode head) {

ListNode prev = null, curr = head;

while (curr != null) {

ListNode nextTemp = curr.next;

curr.next = prev;

prev = curr;

curr = nextTemp;

}

return prev;

}

public static void main(String[] args) {

ListNode head = new ListNode(1);

head.next = new ListNode(2);

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

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

if (isPalindrome(head)) {

System.out.println("The linked list is a palindrome.");

} else {

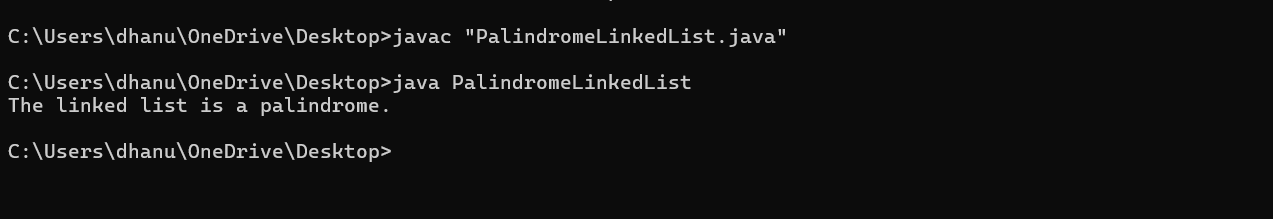
System.out.println("The linked list is not a palindrome.");

}

}

}

**OUTPUT:**

****

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

1. **Balanced tree check**

**CODE:**

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int val) {

this.val = val;

this.left = null;

this.right = null;

}

}

public class BalancedTreeCheck {

public static boolean isBalanced(TreeNode root) {

return checkHeight(root) != -1;

}

private static int checkHeight(TreeNode node) {

if (node == null) {

return 0;

}

int leftHeight = checkHeight(node.left);

if (leftHeight == -1) {

return -1;

}

int rightHeight = checkHeight(node.right);

if (rightHeight == -1) {

return -1;

}

if (Math.abs(leftHeight - rightHeight) > 1) {

return -1;

}

return Math.max(leftHeight, rightHeight) + 1;

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

if (isBalanced(root)) {

System.out.println("The tree is balanced.");

} else {

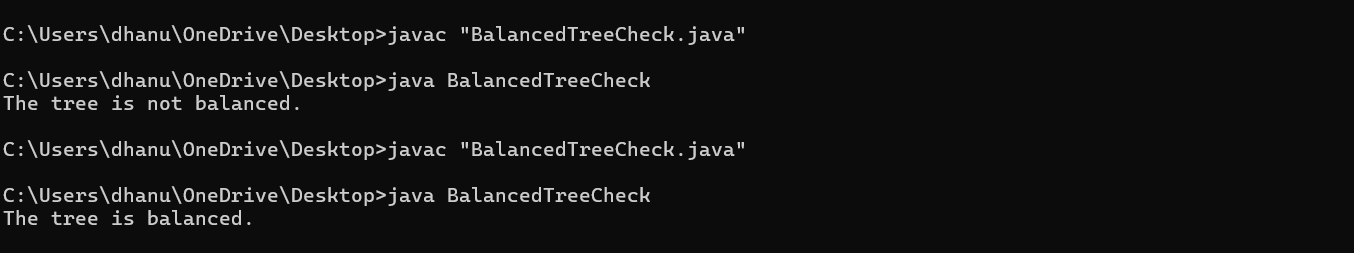
System.out.println("The tree is not balanced.");

}

}

}

**OUTPUT:**

****

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

1. **Triplet sum in array**

**CODE:**

import java.util.Arrays;

public class TripletSumInArray {

public static boolean findTriplet(int[] arr, int targetSum) {

Arrays.sort(arr);

for (int i = 0; i < arr.length - 2; i++) {

int left = i + 1;

int right = arr.length - 1;

while (left < right) {

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

if (currentSum == targetSum) {

System.out.println("Triplet found: " + arr[i] + ", " + arr[left] + ", " + arr[right]);

return true;

} else if (currentSum < targetSum) {

left++;

} else {

right--;

}

}

}

return false;

}

public static void main(String[] args) {

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

int targetSum = 13;

if (!findTriplet(arr, targetSum)) {

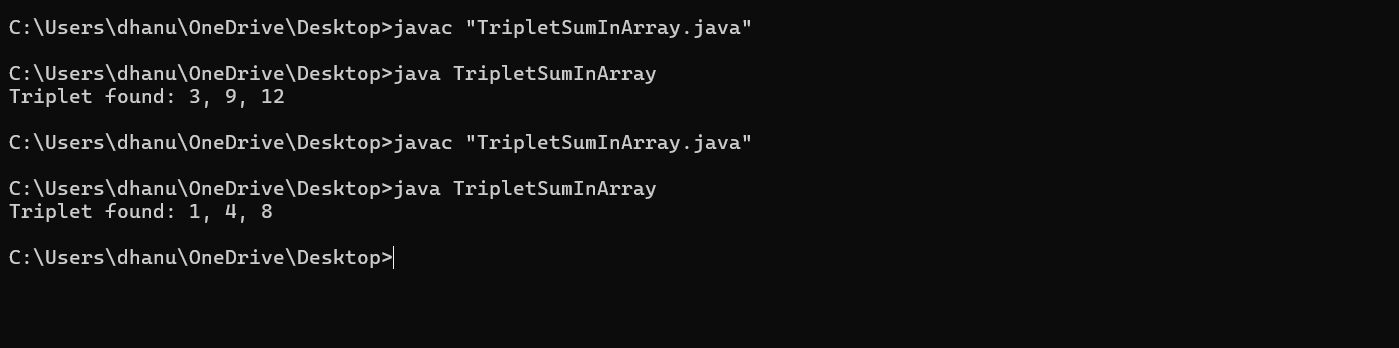
System.out.println("No triplet found with the given sum.");

}

}

}

**OUTPUT:**

****

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