**Coding practice Problems:** 11.11.2024    
 **Name: Sujitha M  
 Dept: CSE**

**1. 0-1 knapsack problem**

public class Knapsack {

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

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

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

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

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

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

} else {

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

}

}

}

return dp[n][W];

}

public static void main(String args[]) {

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

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

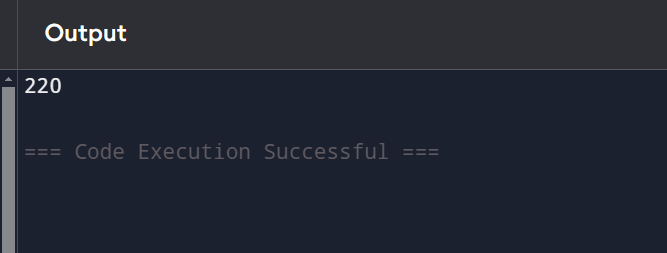
int W = 50;

int n = profit.length;

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

}

}



2. 2. Floor in sorted array

public class FloorInSortedArray {

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

int left = 0, right = arr.length - 1;

int floor = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

return arr[mid];

} else if (arr[mid] < x) {

floor = arr[mid];

left = mid + 1;

} else {

right = mid - 1;

}

}

return floor;

}

public static void main(String[] args) {

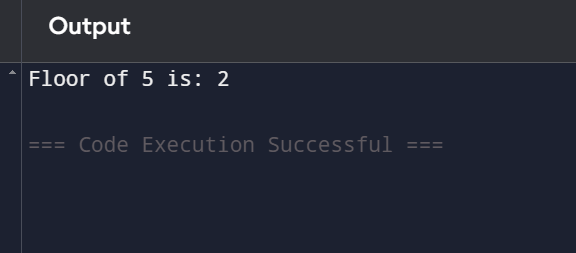
int[] arr = {1, 2, 8, 10, 10, 12, 19};

int x = 5;

System.out.println("Floor of " + x + " is: " + findFloor(arr, x));

}

}



3. Check Equal Arrays

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] arr1 = new int[n];

int[] arr2 = new int[n];

for (int i = 0; i < n; i++) arr1[i] = scanner.nextInt();

for (int i = 0; i < n; i++) arr2[i] = scanner.nextInt();

Arrays.sort(arr1);

Arrays.sort(arr2);

if (Arrays.equals(arr1, arr2)) {

System.out.println("Arrays are equal");

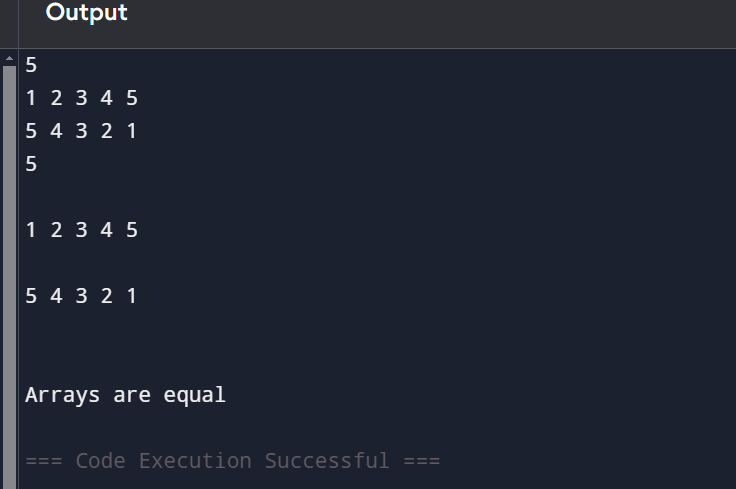
} else {

System.out.println("Arrays are not equal");

}

}

}



4. Palindrome linked list

import java.util.Scanner;

class Main {

static class LinkedList {

Node head;

class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

void push(int new\_data) {

Node new\_node = new Node(new\_data);

new\_node.next = head;

head = new\_node;

}

boolean isPalindrome() {

Node slow = head, fast = head;

Node prev = null, next = null;

if (head == null || head.next == null) return true;

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

fast = fast.next.next;

next = slow.next;

slow.next = prev;

prev = slow;

slow = next;

}

if (fast != null) slow = slow.next;

while (slow != null) {

if (slow.data != prev.data) return false;

slow = slow.next;

prev = prev.next;

}

return true;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

LinkedList list = new LinkedList();

int n = scanner.nextInt();

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

list.push(scanner.nextInt());

}

if (list.isPalindrome()) {

System.out.println("Palindrome");

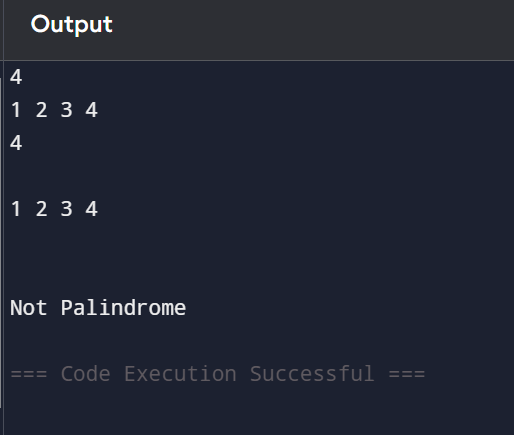
} else {

System.out.println("Not Palindrome");

}

}

}



5.Balanced Tree Check

import java.util.Scanner;

class Main {

static class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

static class BinaryTree {

Node root;

boolean isBalanced(Node node) {

return checkHeight(node) != -1;

}

int checkHeight(Node 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) {

Scanner scanner = new Scanner(System.in);

BinaryTree tree = new BinaryTree();

System.out.println("Enter the number of nodes in the tree:");

int n = scanner.nextInt();

System.out.println("Enter the values for the nodes (Pre-order):");

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

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

if (tree.isBalanced(tree.root)) {

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

} else {

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

}

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else {

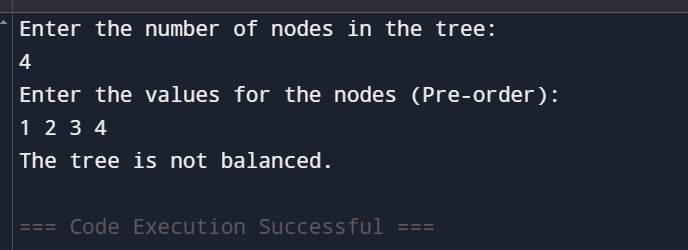
root.right = insert(root.right, data);

}

return root;

}

}



6. Triplet sum

import java.util.Scanner;

import java.util.Arrays;

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int sum = scanner.nextInt();

int[] arr = new int[n];

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

arr[i] = scanner.nextInt();

}

Arrays.sort(arr);

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

int left = i + 1;

int right = n - 1;

while (left < right) {

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

if (currentSum == sum) {

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

return;

} else if (currentSum < sum) {

left++;

} else {

right--;

}

}

}

System.out.println("No triplet found");

}

}

