**Date :11/11/2024**

**Name: Kowsick K**

**regNo:22CS083**

**Dept : CSE**

**1.0 - 1 Knapsack Problem**

class Main {

public 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; i++) {

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

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

dp[i][w] = 0;

else 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[] val = {60, 100, 120};

int[] wt = {10, 20, 30};

int W = 50;

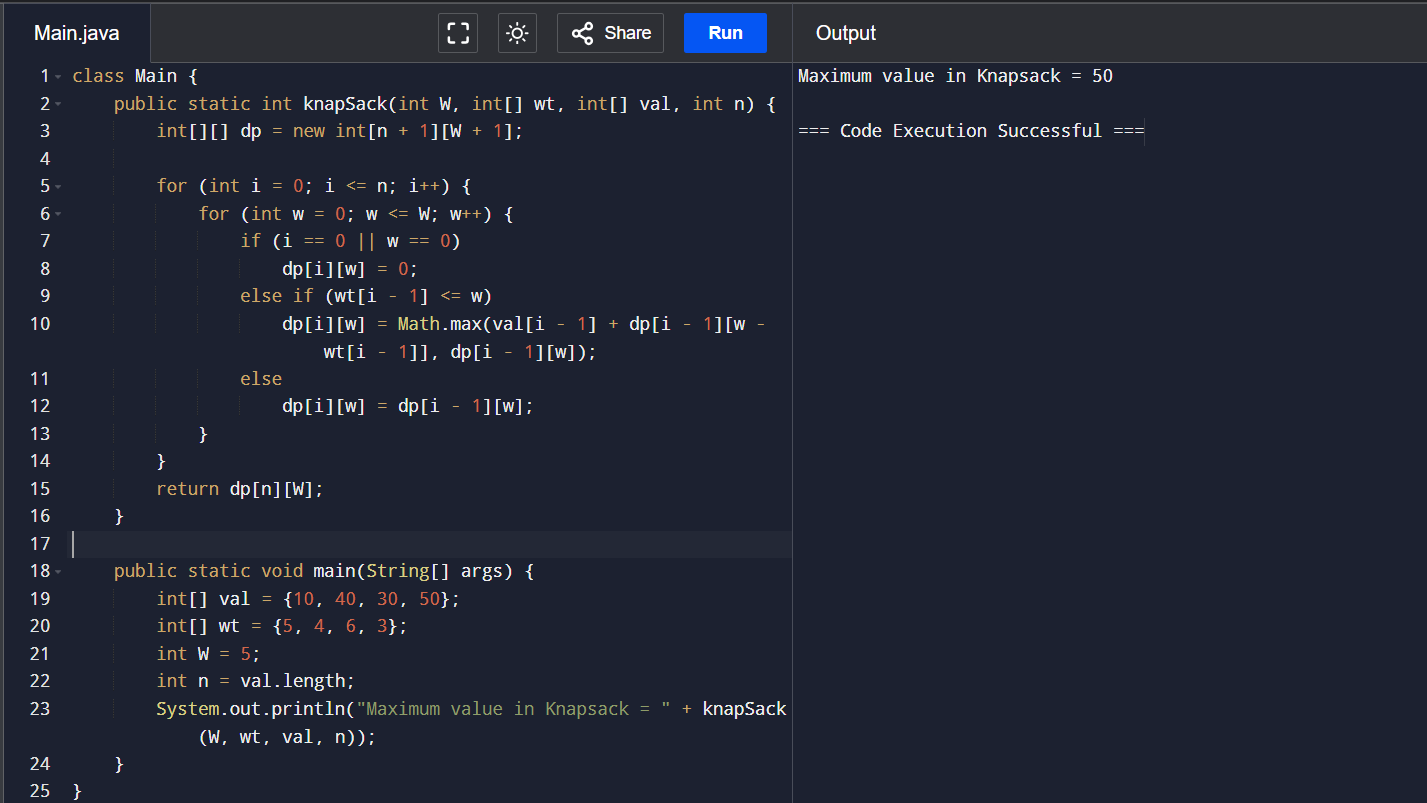
int n = val.length;

System.out.println("Maximum value in Knapsack = " + knapSack(W, wt, val, n));

}

}

Output:



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

**Space Complexity:** O(n)

**2.Floor in sorted array**

class Main {

public static int floorSearch(int[] arr, int low, int high, int key) {

if (key < arr[low])

return -1;

int mid;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == key)

return mid;

if (arr[mid] < key) {

if (mid + 1 <= high && arr[mid + 1] > key)

return mid;

low = mid + 1;

} else

high = mid - 1;

}

return high;

}

public static void main(String[] args) {

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

int key = 5;

int index = floorSearch(arr, 0, arr.length - 1, key);

if (index != -1)

System.out.println("Floor of " + key + " is " + arr[index]);

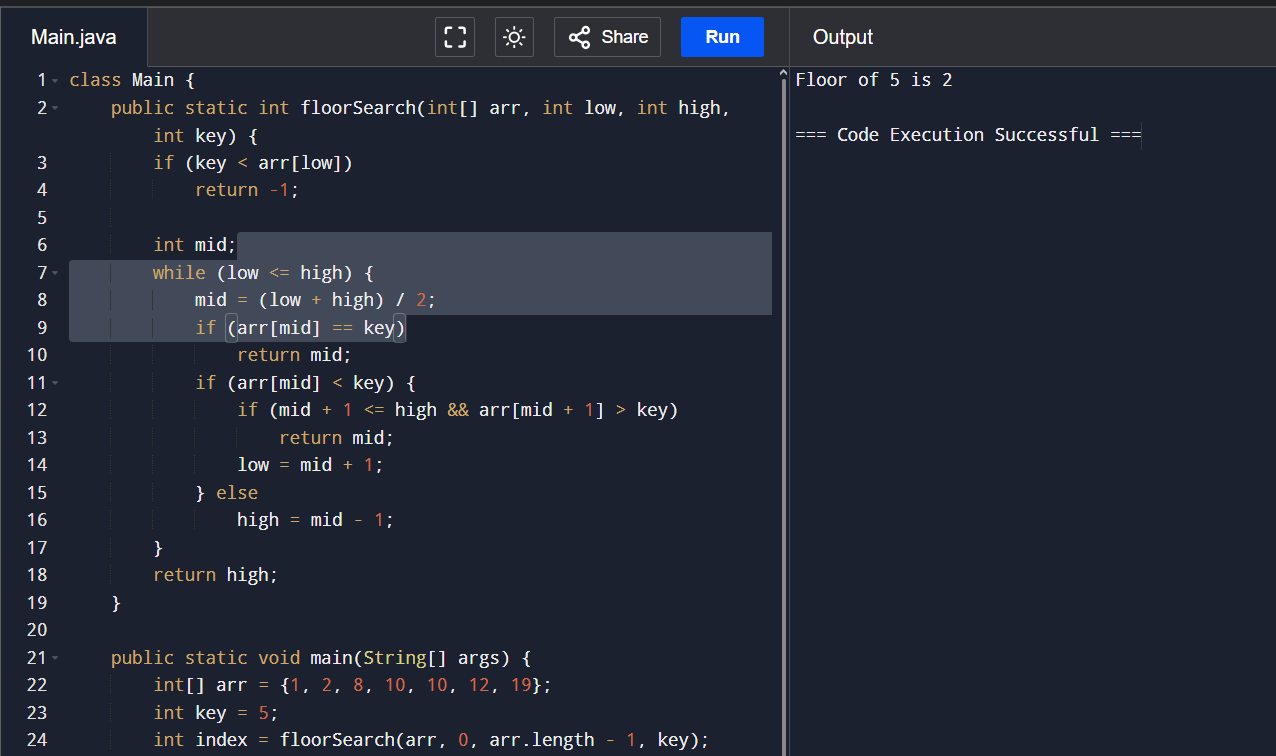
else

System.out.println("Floor doesn't exist.");

}

}

Output:



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

**Space Complexity: O(1)**

**3.Check equal arrays**

public class Main {

private static void areArraysEqual(int arr1[], int arr2[]) {

boolean isEqual = true;

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

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

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

isEqual = false;

break;

}

}

} else {

isEqual = false;

}

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

}

public static void printArrays(int arr1[], int arr2[]) {

for (int num : arr1)

System.out.print(num + " ");

System.out.println();

for (int num : arr2)

System.out.print(num + " ");

System.out.println();

}

public static void main(String[] args) {

int arr1[] = { 10, 30, 12 };

int arr2[] = { 10, 30, 12 };

printArrays(arr1, arr2);

areArraysEqual(arr1, arr2);

System.out.println();

int arr3[] = { 10, 30, 12 };

int arr4[] = { 45, 50, 55, 60, 65 };

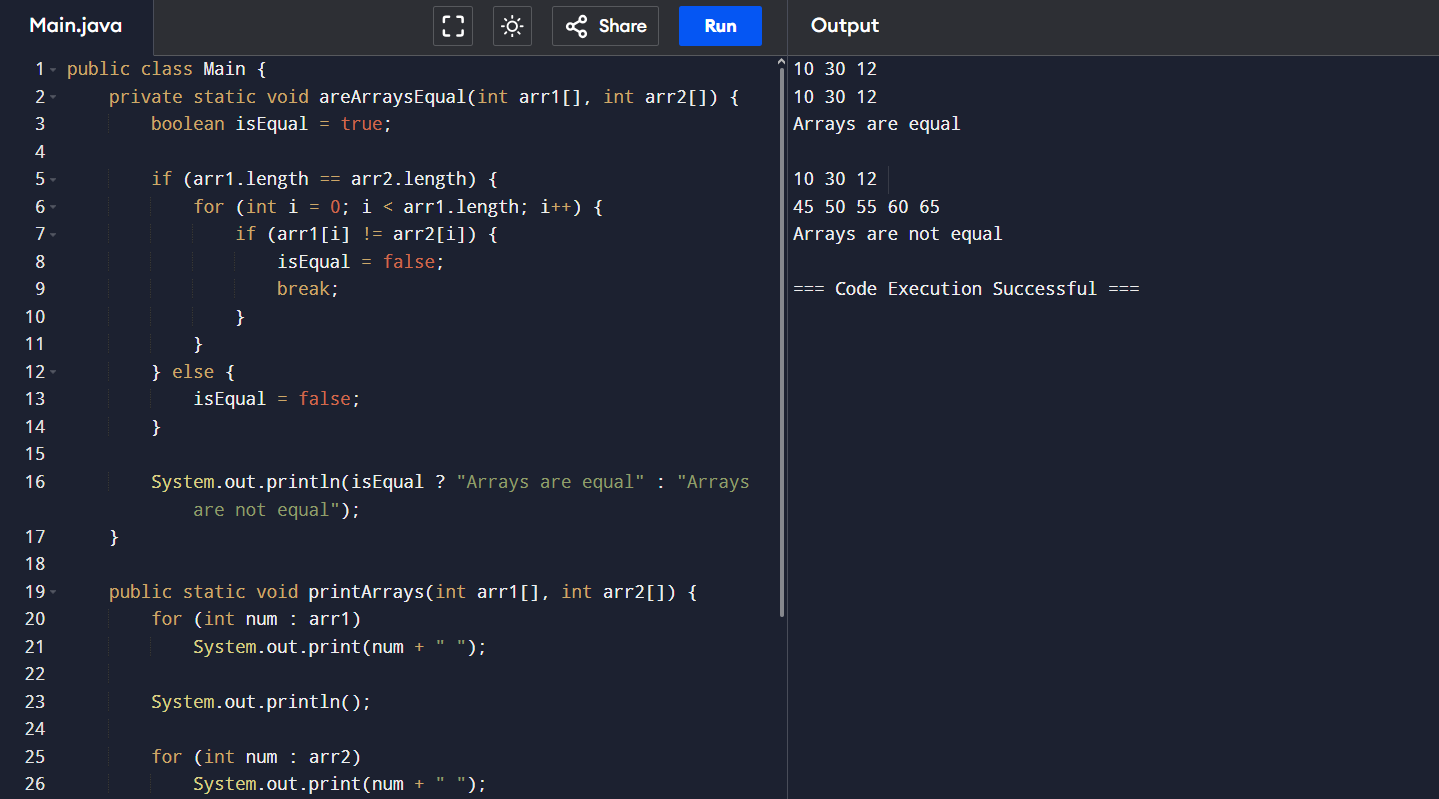
printArrays(arr3, arr4);

areArraysEqual(arr3, arr4);

}

}

Output:



***Time Complexity :*** *O(n)****Auxiliary Space :*** *O(1)*

**4.Palindrome linked list**

class Solution {

Node reverse\_list(Node head) {

Node prev = null;

Node curr = head;

Node next;

while (curr != null) {

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

return prev;

}

boolean is\_identical(Node n1, Node n2) {

for (; n1 != null && n2 != null; n1 = n1.next, n2 = n2.next)

if (n1.data != n2.data) return false;

return true;

}

boolean isPalindrome(Node head) {

int size = 0;

Node ptr;

for (ptr = head; ptr != null; ptr = ptr.next) size++;

if (size < 2) return true;

ptr = head;

int mid\_pt = (size - 1) / 2;

while (mid\_pt > 0) {

ptr = ptr.next;

mid\_pt--;

}

Node head2 = ptr.next;

ptr.next = null;

head2 = reverse\_list(head2);

boolean ret = is\_identical(head, head2);

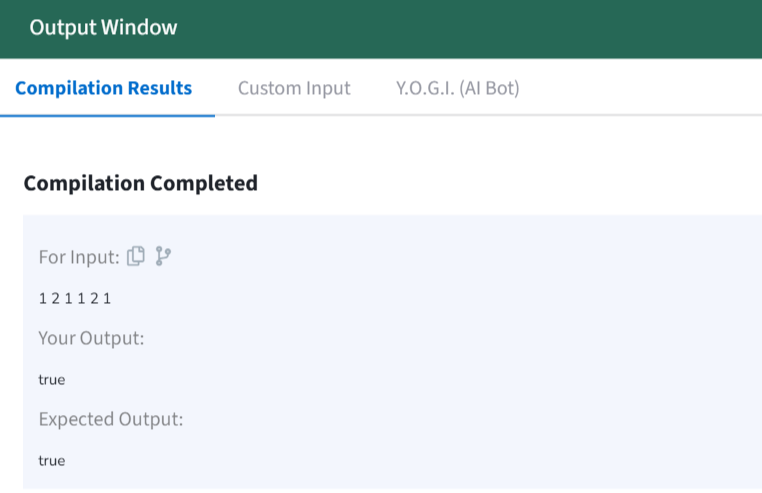
head2 = reverse\_list(head2);

ptr.next = head2;

return ret;

}

}



**Expected Time Complexity**: O(n)  
**Expected Auxiliary Space**: O(1)

**5.Check for Balanced Binary Tree**

import java.util.\*;

class TreeNode {

int value;

TreeNode left;

TreeNode right;

TreeNode(int value) {

this.value = value;

left = right = null;

}

}

class Tree {

TreeNode root;

public TreeNode insertLevelOrder(String[] arr, TreeNode root, int i) {

if (i < arr.length) {

if (!arr[i].equals("N")) {

TreeNode temp = new TreeNode(Integer.parseInt(arr[i]));

root = temp;

root.left = insertLevelOrder(arr, root.left, 2 \* i + 1);

root.right = insertLevelOrder(arr, root.right, 2 \* i + 2);

}

}

return root;

}

public boolean isBalanced(TreeNode root) {

if (root == null)

return true;

return Math.abs(maxDepth(root.left) - maxDepth(root.right)) <= 1 &&

isBalanced(root.left) &&

isBalanced(root.right);

}

private int maxDepth(TreeNode root) {

if (root == null)

return 0;

return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));

}

}

Output:

True

**6.Triplet sum in array**

**Code:**

class Main {

public static void main(String[] args) {

String[] arr = {"1", "2", "N", "N", "3"};

Tree g = new Tree();

g.root = g.insertLevelOrder(arr, g.root, 0);

if (g.isBalanced(g.root)) {

System.out.println("1");

} else {

System.out.println("0");

}

}

}

import java.util.\*;

class Main {

public List<List<Integer>> findTripletsWithZeroSum(int[] numbers) {

Set<List<Integer>> triplets = new HashSet<>();

Arrays.sort(numbers);

int length = numbers.length;

for (int first = 0; first < length - 2; first++) {

int second = first + 1;

int third = length - 1;

while (second < third) {

int sum = numbers[first] + numbers[second] + numbers[third];

if (sum == 0) {

triplets.add(Arrays.asList(numbers[first], numbers[second], numbers[third]));

second++;

} else if (sum > 0) {

third--;

} else {

second++;

}

}

}

return new ArrayList<>(triplets);

}

public static void main(String[] args) {

TripletSum tripletSum = new TripletSum();

int[] numbers = {-1, 0, 1, 2, -1, -4};

List<List<Integer>> tripletResults = tripletSum.findTripletsWithZeroSum(numbers);

System.out.println("Triplets with sum zero:");

for (List<Integer> triplet : tripletResults) {

System.out.println(triplet);

}

}

}

