**Bhuvaneswari P AIDS DSA Practice 2 11/11/2024**

**1.0/1 Knapsack Problem**

class Solution {

public int knapSack(int capacity, int[] wt, int[] val) {

int n = val.length;

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

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

for (int w = 1; w <= capacity; 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][capacity];

}

public static void main(String[] args) {

Solution solution = new Solution();

int capacity = 50;

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

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

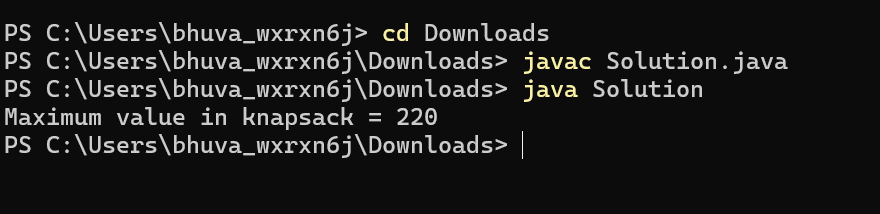
int maxVal = solution.knapSack(capacity, wt, val);

System.out.println("Maximum value in knapsack = " + maxVal);

}

}

**Output :**



Time Complexity : O(n\*capacity)

Space Complexity : O(n\*capacity)

**2. Floor in sorted array**

class Example {

public int findLargestLessThanOrEqualToK(int[] arr, int k) {

int left = 0;

int right = arr.length - 1;

int result = -1;

while (left <= right) {

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

if (arr[mid] <= k) {

result = mid;

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static void main(String[] args) {

Example example = new Example();

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

int k = 2;

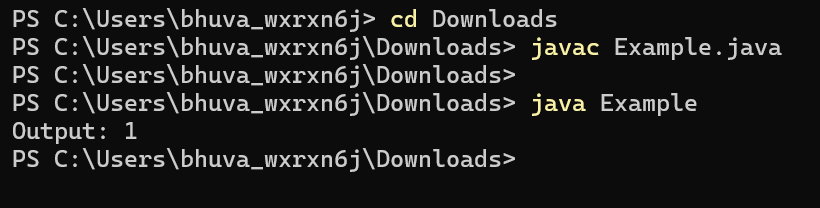
int index = example.findLargestLessThanOrEqualToK(arr, k);

System.out.println("Output: " + index);

}

}

**Output:**



Time Complexity : O(log n)

Space Complexity : O(1)

**3. Check equal arrays**

import java.util.HashMap;

public class CheckEqualArray {

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

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

return false;

}

HashMap<Integer, Integer> freqMap = new HashMap<>();

for (int num : arr1) {

freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);

}

for (int num : arr2) {

if (!freqMap.containsKey(num)) {

return false;

}

freqMap.put(num, freqMap.get(num) - 1);

if (freqMap.get(num) == 0) {

freqMap.remove(num);

}

}

return freqMap.isEmpty();

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 5, 4, 0};

int[] arr2 = {2, 4, 5, 0, 1};

if (areArraysEqual(arr1, arr2)) {

System.out.println("true");

} else {

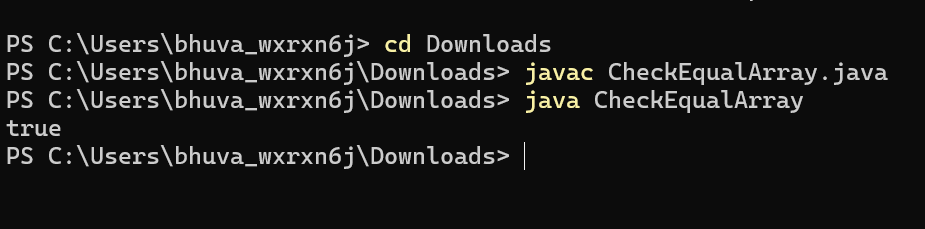
System.out.println("false");

}

}

}

**Output:**

****

Time Complexity : O(n)

Space Complexity : O(n)

**4. Palindrome Linked List**

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public class PalindromeLinkedList {

Node head;

public boolean isPalindrome() {

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

Node slow = head, fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

Node secondHalf = reverse(slow);

Node firstHalf = head;

while (secondHalf != null) {

if (firstHalf.data != secondHalf.data) return false;

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

return true;

}

private Node reverse(Node head) {

Node prev = null, current = head, next;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

return prev;

}

public void addNode(int data) {

if (head == null) {

head = new Node(data);

} else {

Node temp = head;

while (temp.next != null) temp = temp.next;

temp.next = new Node(data);

}

}

public static void main(String[] args) {

PalindromeLinkedList list = new PalindromeLinkedList();

list.addNode(1);

list.addNode(2);

list.addNode(1);

list.addNode(1);

list.addNode(2);

list.addNode(1);

System.out.println(list.isPalindrome());

PalindromeLinkedList list2 = new PalindromeLinkedList();

list2.addNode(1);

list2.addNode(2);

list2.addNode(3);

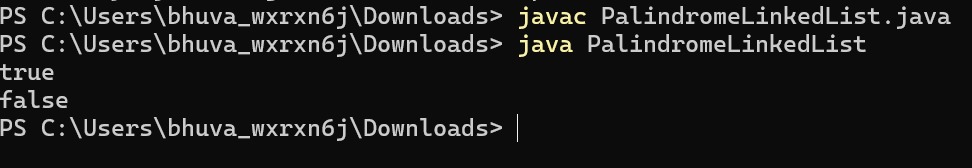
list2.addNode(4);

System.out.println(list2.isPalindrome());

}

}

**Output :**



Time Complexity: O(n)

Space Complexity: O(1)

**5. Balanced tree check**

class BinaryTree {

static class Node {

int value;

Node left, right;

Node(int value) {

this.value = value;

left = right = null;

}

}

private static int checkBalance(Node root) {

if (root == null) return 0;

int leftHeight = checkBalance(root.left);

if (leftHeight == -1) return -1;

int rightHeight = checkBalance(root.right);

if (rightHeight == -1) return -1;

if (Math.abs(leftHeight - rightHeight) > 1) return -1;

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

}

public static boolean isBalanced(Node root) {

return checkBalance(root) != -1;

}

public static void main(String[] args) {

Node tree1 = new Node(1);

tree1.left = new Node(2);

tree1.left.right = new Node(3);

System.out.println(isBalanced(tree1));

Node tree2 = new Node(10);

tree2.left = new Node(20);

tree2.right = new Node(30);

tree2.left.left = new Node(40);

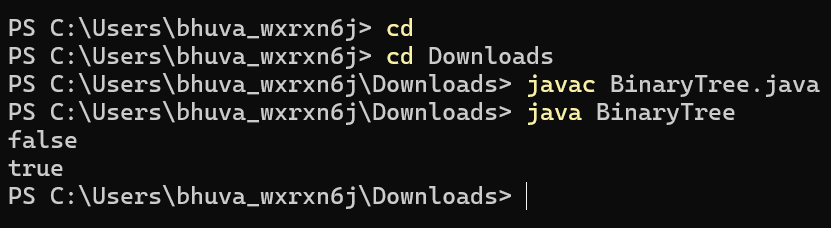
tree2.left.right = new Node(60);

System.out.println(isBalanced(tree2));

}

}

Output:



Time complexity: O(h)

Space Complexity:O(n)

**6.TripletSum**

import java.util.Arrays;

public class TripletSum {

public static int findTriplet(int arr[], int n, int x) {

Arrays.sort(arr);

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

int left = i + 1;

int right = n - 1;

while (left < right) {

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

if (sum == x) {

return 1;

} else if (sum < x) {

left++;

} else {

right--;

}

}

}

return 0;

}

public static void main(String[] args) {

int arr1[] = {1, 4, 45, 6, 10, 8};

int n1 = 6, x1 = 13;

System.out.println(findTriplet(arr1, n1, x1));

int arr2[] = {1, 2, 4, 3, 6, 7};

int n2 = 6, x2 = 10;

System.out.println(findTriplet(arr2, n2, x2));

int arr3[] = {40, 20, 10, 3, 6, 7};

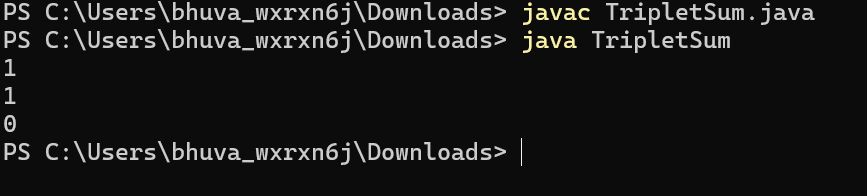
int n3 = 6, x3 = 24;

System.out.println(findTriplet(arr3, n3, x3));

}

}

Output:



Time Complexity:O(n^2)

Space Complexity:O(1)