**DSA Practice 2 (11/11/2024)**

**By Jashwanth SA**

**CSE C, 22CS066**

1. **0-1 Knapsack problem:**

import java.util.\*;

class Main {

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

int n = val.length;

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

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

for (int j = 1; j <= capacity; j++) {

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

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

} else {

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

}

}

}

return dp[n][capacity];

}

public static void main(String[] args) {

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

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

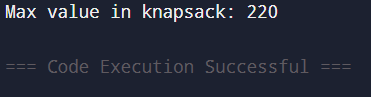
int capacity = 50;

System.out.println("Max value in knapsack: " + knapSack(capacity, val, wt));

}

}

Output:



Time complexity: O(n\*capacity)

1. **Floor in sorted array:**

class Main {

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

if (k < arr[0]) return -1;

for (int i = 0; i < arr.length - 1; i++) {

if (arr[i] <= k && arr[i + 1] > k) return i;

}

return arr.length - 1;

}

public static void main(String[] args) {

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

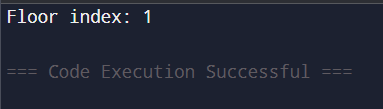
int k = 5;

System.out.println("Floor index: " + findFloor(arr, k));

}

}

Output:



Time complexity: O(n)

1. **Check Equal Arrays:**

import java.util.HashMap;

class Main {

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

if (arr1.length != arr2.length) return false;

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

for (int num : arr1) {

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

}

for (int num : arr2) {

if (!map.containsKey(num) || map.get(num) == 0) return false;

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

}

return true;

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

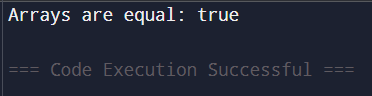
int[] arr2 = {5, 4, 3, 2, 1};

System.out.println("Arrays are equal: " + check(arr1, arr2));

}

}

Output:



Time complexity: O(n+m)

1. **Palindrome Linked List:**

import java.util.ArrayList;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class Main {

public static boolean isPalindrome(Node head) {

ArrayList<Integer> list = new ArrayList<>();

while (head != null) {

list.add(head.data);

head = head.next;

}

int n = list.size();

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

if (!list.get(i).equals(list.get(n - i - 1))) return false;

}

return true;

}

public static void main(String[] args) {

Node head = new Node(1);

head.next = new Node(2);

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

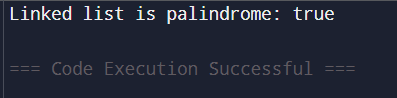
head.next.next.next = new Node(1);

System.out.println("Linked list is palindrome: " + isPalindrome(head));

}

}

Output:



Time complexity: O(n)

1. **Balanced tree check:**

class TreeNode {

int data;

TreeNode left, right;

TreeNode(int data) {

this.data = data;

this.left = this.right = null;

}

}

class Main {

public static boolean isBalanced(TreeNode root) {

if (root == null) return true;

int leftHeight = getHeight(root.left);

int rightHeight = getHeight(root.right);

return Math.abs(leftHeight - rightHeight) <= 1 && isBalanced(root.left) && isBalanced(root.right);

}

private static int getHeight(TreeNode root) {

if (root == null) return 0;

int leftHeight = getHeight(root.left);

int rightHeight = getHeight(root.right);

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);

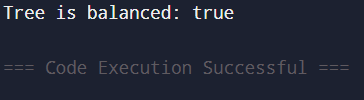
root.right.right = new TreeNode(6);

System.out.println("Tree is balanced: " + isBalanced(root));

}

}

Output:



Time complexity: O(n)

1. **Triple sum in array:**

import java.util.Arrays;

class Main {

public static boolean find3Numbers(int[] arr, int n, int x) {

Arrays.sort(arr);

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

int l = i + 1, r = n - 1;

while (l < r) {

int curr\_sum = arr[i] + arr[l] + arr[r];

if (curr\_sum == x) {

return true;

} else if (curr\_sum < x) {

l++;

} else {

r--;

}

}

}

return false;

}

public static void main(String[] args) {

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

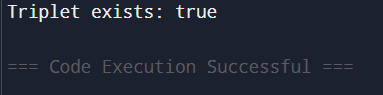
int x = 22;

System.out.println("Triplet exists: " + find3Numbers(arr, arr.length, x));

}

}

Output:



Time complexity: O(n^2)