Saravanan BS-AI&DS-DSA-Practice-2

1. 0-1 knapsack problem:

class Solution {

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 w = 0; w <= capacity; w++) {

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

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

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

}

}

}

return dp[n][capacity];

}

}

Result:

Input: capacity = 4, val[] = [1, 2, 3], wt[] = [4, 5, 1]   
Output: 3

Input: capacity = 3, val[] = [1, 2, 3], wt[] = [4, 5, 6]   
Output: 0

Input: capacity = 5, val[] = [10, 40, 30, 50], wt[] = [5, 4, 6, 3]   
Output: 50

Time Complexity: O(n x w)

2. Floor in sorted array:

class Solution {

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

int low, high;

low = 0;

high = arr.length;

int index;

index = -1;

int mid;

while(low <= high){

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

if (arr[mid] == k){

return mid;

}

else if(arr[mid] < k){

index = mid;

low = mid + 1;

}

else{

high = mid -1;

}

}

return index;

}

}

Result:

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 0  
Output: -1

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 5  
Output: 1

Input: arr[] = [1, 2, 8], k = 1  
Output: 0

Time complexity: O(log(N))

3. Check equal arrays:

class Solution {

// Function to check if two arrays are equal or not.

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

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

return false;

}

HashMap<Integer, Integer> map1 = new HashMap<>();

HashMap<Integer, Integer> map2 = new HashMap<>();

for (int i : arr1) {

map1.put(i, map1.getOrDefault(i, 0) + 1);

}

for (int i : arr2) {

map2.put(i, map2.getOrDefault(i, 0) + 1);

}

for (int key : map1.keySet()) {

if (!map2.containsKey(key) || !map1.get(key).equals(map2.get(key))) {

return false;

}

}

return true;

}

}

Result:

Input: arr1[] = [1, 2, 5, 4, 0], arr2[] = [2, 4, 5, 0, 1]  
Output: true

Input: arr1[] = [1, 2, 5], arr2[] = [2, 4, 15]  
Output: false

Time Complexity: O(N)

4. Palindrome linked list:

/\* Structure of class Node is

class Node

{

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}\*/

class Solution {

// Function to check whether the list is palindrome.

boolean isPalindrome(Node head) {

Stack<Integer> st = new Stack<>();

Node temp = head;

while(temp != null){

st.push(temp.data);

temp = temp.next;

}

temp = head;

while(temp != null){

if(temp.data != st.pop()){

return false;

}

temp = temp.next;

}

return true;

}

}

Result:

Input: LinkedList: 1->2->1->1->2->1  
Output: true

Input: LinkedList: 1->2->3->4  
Output: false

Time Complexity: O(N)

5. Balanced tree check:

/\* A binary tree node class

class Node

{

int data;

Node left,right;

Node(int d)

{

data = d;

left = right = null;

}

} \*/

class Tree

{

public int height(Node root){

if(root == null){

return 0;

}

int left = height(root.left);

int right = height(root.right);

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

}

//Function to check whether a binary tree is balanced or not.

boolean isBalanced(Node root)

{

if(root == null){

return true;

}

int lh = height(root.left);

int rh = height(root.right);

if (Math.abs(lh - rh) > 1){

return false;

}

return (isBalanced(root.left) && isBalanced(root.right));

}

}

Result:

Input:  
 1  
 /  
 2  
 \  
 3   
Output: 0

Input:  
 10  
 / \  
 20 30   
 / \  
 40 60  
Output: 1

Time complexity: O(N2)

6. Triplet sum in array:

class Solution {

// Should return true if there is a triplet with sum equal

// to x in arr[], otherwise false

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

Arrays.sort(arr);

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

int l = i+1;

int r = n - 1;

while(l < r){

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

if(curr == x){

return true;

}

else if(curr > x){

r--;

}

else{

l++;

}

}

}

return false;

}

}  
Result:

Input:n = 6, x = 13, arr[] = [1,4,45,6,10,8]  
Output: 1

Input: n = 6, x = 10, arr[] = [1,2,4,3,6,7]  
Output: 1

Input: n = 6, x = 24, arr[] = [40,20,10,3,6,7]  
Output: 0