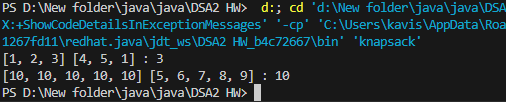
**0-1.Knapsack Problem**

**CODE :**

import java.util.Arrays;public class knapsack{ public static void main(String[] args) { knapsack(4,new int[]{1,2,3},new int[]{4,5,1}); knapsack(10, new int[]{10, 10, 10, 10, 10}, new int[]{5, 6, 7, 8, 9}); } static void knapsack(int capacity, int val[], int wt[]) { System.out.print(Arrays.toString(val) + " " + Arrays.toString(wt) + " : "); int[][] dp = new int[val.length][capacity+1]; for(int[] i : dp){ Arrays.fill(i,-1); } System.out.println(helper(val.length-1 , capacity , val , wt ,dp)); } public static int helper(int i , int w , int[] val , int[] wt,int[][]dp){ if(i==0){ if(wt[i]<=w){ return val[0]; }else{ return 0; } } if(dp[i][w]!=-1) return dp[i][w]; int nottake = 0 + helper(i-1 , w , val ,wt,dp); int take = Integer.MIN\_VALUE; if(wt[i]<=w){ take = val[i] + helper(i-1,w-wt[i] , val , wt,dp); } return dp[i][w] = Math.max(take,nottake); }}

**OUTPUT :**

****

TIME COMPLEXITY : O(n\*W)

SPACE COMPLEXITY : O(n\*W)

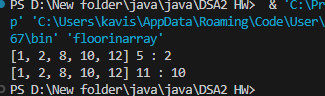
**Floor in sorted array**

**CODE :**

import java.util.\*;public class floorinarray { public static void main(String[] args) { floor(new int[]{1,2,8,10,12} , 5); floor(new int[]{1,2,8,10,12} , 11);

} public static void floor(int[] arr , int x){ System.out.print(Arrays.toString(arr) + " " + x + " : " ); int i = 0 , j = arr.length-1; int mid = (i+j)/2; boolean flag = false; if(x>arr[j]){ flag = true; System.out.println(arr[j]); } else if(x<arr[0]){ flag = true; System.out.println(-1); } while(i<j && !flag){ mid = (i+j)/2; // System.out.println(mid + " " + i + " " + j); if(mid>0 && arr[mid-1] < x && arr[mid]>=x){ flag = true; System.out.println(arr[mid-1]); break; } else if(mid<arr.length && arr[mid]<=x && arr[mid+1]>x){ flag = true; System.out.println(arr[mid]); break; } else if(arr[mid] < x){ i = mid+1; }else{ j = mid-1; } } if(!flag) System.out.println(-1); }}

**OUTPUT :**

****

TIME COMPLEXITY : O(log n)

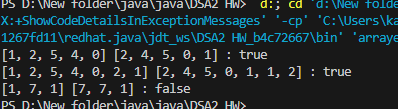
SPACE COMPLEXITY : O(1)

**Check equal arrays**

**CODE:**

import java.util.\*;public class arrayequal { public static void main(String[] args) { equal(new int[]{1, 2, 5, 4, 0} ,new int[]{2, 4, 5, 0, 1}); equal(new int[]{1, 2, 5, 4, 0, 2, 1} ,new int[]{2, 4, 5, 0, 1, 1, 2}); equal(new int[]{1, 7, 1} ,new int[]{7, 7, 1}); } public static void equal(int[] arr1, int[] arr2){ System.out.print(Arrays.toString(arr1) + " " + Arrays.toString(arr2) + " : "); Map<Integer,Integer> hm = new HashMap<>(); for(int i = 0 ; i< arr1.length ; i++){ hm.put(arr1[i],hm.getOrDefault(arr1[i],0)+1); } for(int i = 0 ; i< arr1.length ; i++){ hm.put(arr2[i],hm.getOrDefault(arr2[i],0)-1); } boolean flag = false; for(Map.Entry<Integer,Integer> e : hm.entrySet()){ if(e.getValue()!=0){ flag = true; System.out.println("false"); break; } } if(!flag) System.out.println("true"); }}

**OUTPUT:**

****

TIME COMPLEXITY : O(n)

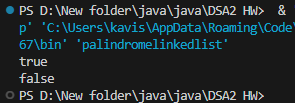
SPACE COMPLEXITY : O(n)

**Palindrome linked list**

**CODE:**

import java.util.\*;public class palindromelinkedlist { public static void main(String[] args) { LinkedList<Integer> l = new LinkedList<>(); l.add(1); l.add(2); l.add(2); l.add(1); palin(l); LinkedList<Integer> l1 = new LinkedList<>(); l1.add(1); l1.add(3); l1.add(2); l1.add(1); palin(l1); } public static void palin(LinkedList<Integer> l) { boolean flag = false; int i = 0 , j = l.size()-1; while(i<j){ if(l.get(i) != l.get(j)){ flag = true; System.out.println("false"); break; } i++; j--; } if(!flag) System.out.println("true"); }}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

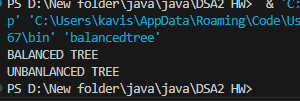
SPACE COMPLEXITY : O(1)

**Balanced tree check**

**CODE :**

class Node{ int data; Node left; Node right; Node(int data){ this.data = data; this.left = null; this.right = null; }}public class balancedtree { public static void main(String[] args) { Node root = new Node(0); root.left = new Node(1); root.right = new Node(2); root.left.left = new Node(3); root.left.right = new Node(4); System.out.println(helper(root)!=-1 ? "BALANCED TREE" : "UNBANLANCED TREE"); Node r1 = new Node(0); r1.left = new Node(1); r1.right = new Node(2); r1.left.right = new Node(4); r1.left.left = new Node(3); r1.left.left.left = new Node(5); System.out.println(helper(r1)!=-1? "BALANCED TREE" : "UNBANLANCED TREE"); } public static int helper(Node root){ if(root==null) return 0; int lh = helper(root.left); if(lh==-1) return -1; int rh = helper(root.right); if(rh==-1) return -1; int hh = Math.abs(lh-rh); if(hh>1) return -1; return 1+Math.max(lh,rh); }}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

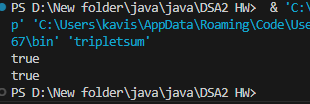
SPACE COMPLEXITY : O(n)

**Triplet sum in array**

**CODE:**

import java.util.Arrays;public class tripletsum { public static void main(String[] args) { System.out.println(find(new int[]{1,4,45,6,10,8},13)); System.out.println(find(new int[]{1,2,4,3,6,7},10)); } public static boolean find(int arr[], int x) { int n = arr.length ; Arrays.sort(arr); for(int i = 0 ; i<n ; i++){ int j = i+1; int k = n-1; while(j<k){ int sum = arr[i] + arr[j] + arr[k]; if(sum<x){ j++; }else if(sum>x){ k--; }else{ return true; } } } return false; }}

**OUTPUT :**

****

TIME COMPLEXITY : O(n^2)

SPACE COMPLEXITY : O(1)