Arrays Assignment 20BD1A1290-Jakkanagari Harika

Problem 1:

There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the i th kid has, and an integer extracandies, denoting the number of extra candies that you have. Return a boolean array result of length n, where result[i] is true if, after giving the i th kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise. Note that multiple kids can have the greatest number of candies.

Solution:

//Java Program

import java.util.\*;

class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

int i,j;

int max=a[0];

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

a[i]=sc.nextInt();

max=Math.max(max,a[i]);

}

int excan=sc.nextInt();

boolean res[]=new boolean[n];

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

int k=a[i]+excan;

if(k>=max){

res[i]=true;

}

else{

res[i]=false;

}

}

System.out.println(Arrays.toString(res));

}

}

Example:

Input: n=5,candies = [2,3,5,1,3], extraCandies = 3

Output: [true,true,true,false,true]

Problem 2:

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i th line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

Solution:

Java Program

Import java.util.\*;

class MaxWater{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

int l=0,r=a.length-1;

int max=0;

while(l<r){

int area=Math.min(a[l],a[r])\*(r-l);

max=Math.max(area,max);

if(a[l]<a[r]){

l++;

}

Else{

r--;

}

}

System.out.println(max);

}

}

Example :

Input: height = [1,8,6,2,5,4,8,3,7] Output: 49

Problem 3:

Solution:

Java Program

import java.util.\*;

class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

int target=sc.nextInt();

System.out.println(method(a,target));

}

public int threeSumClosest(int[] nums, int target) {

Arrays.sort(nums);

int closestSum = nums[0] + nums[1] + nums[2];

for (int i = 0; i < nums.length - 2; i++) {

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (Math.abs(sum - target) < Math.abs(closestSum - target)) {

closestSum = sum;

}

if (sum < target) {

j++;

} else {

k--;

}

}

}

return closestSum;

}

}

Example :

Input: nums = [-1,2,1,-4], target = 1 Output: 2

Program 4:

A permutation of an array of integers is an arrangement of its members into a sequence or linear order. ● For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]. The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order). ● ● ● For example, the next permutation of arr = [1,2,3] is [1,3,2]. Similarly, the next permutation of arr = [2,3,1] is [3,1,2]. While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement. Given an array of integers nums, find the next permutation of nums. The replacement must be in place and use only constant extra memory.

Solution:

//Java Program

import java.util.\*;

class Permutations{  
 public static void main(String[] args){

Scanner sc=new Scanner(System.in);

Int n=sc.nextInt();

Int a[]=new int[n];

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

a[i]=sc.nextInt();

}

Method(a);

System.out.println(Arrays.toString(a));

}

public static void Method(int[] nums) {

int i = nums.length - 2;

while (i >= 0 && nums[i] >= nums[i + 1])

i--;

}

if (i >= 0) {

int j = nums.length - 1;

while (j >= 0 && nums[j] <= nums[i]) {

j--;

}

swap(nums, i, j);

}

reverse(nums, i + 1);

}

private void swap(int[] nums, int i, int j) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

private void reverse(int[] nums, int start) {

int end = nums.length - 1;

while (start < end) {

swap(nums, start, end);

start++;

end--;

}

}

Example :

Input: nums = [1,2,3] Output: [1,3,2]

Program 5:

There is an integer array nums sorted in ascending order (with distinct values). Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2]. Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums. You must write an algorithm with O(log n) runtime complexity.

Solution:

//Java Program

import java.util.\*;

class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

int t=sc.nextInt();

int ans=-1;

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

if(a[i]==t){

ans=i;

}

}

System.out.println(ans);

}

}

Example :

Input: nums = [4,5,6,7,0,1,2], target = 0 Output: 4

Program 6:

Given a collection of numbers, nums, that might contain duplicates, return all possible unique permutations in any order.

Solution:  
//Java Program

import java.util.\*;

public class UniquePermutations {

public static List<List<Integer>> permuteUnique(int[] nums) {

List<List<Integer>> result = new ArrayList<>();

Arrays.sort(nums);

backtrack(nums, new boolean[nums.length], new ArrayList<>(), result);

return result;

}

private static void backtrack(int[] nums, boolean[] used, List<Integer> permutation, List<List<Integer>> result) {

if (permutation.size() == nums.length) {

result.add(new ArrayList<>(permutation));

return;

}

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

if (used[i] || (i > 0 && nums[i] == nums[i - 1] && !used[i - 1])) {

continue;

}

used[i] = true;

permutation.add(nums[i]);

backtrack(nums, used, permutation, result);

used[i] = false;

permutation.remove(permutation.size() - 1);

}

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

Int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

List<List<Integer>> res = permuteUnique(a);

System.out.println(res);

}

}

Example :

Input: nums = [1,1,2] Output: [[1,1,2], [1,2,1], [2,1,1]]

Program 7:

Given an integer array nums, find the subarray with the largest sum, and return its sum.

Solution:

import java.util.\*;

class LargestSum{  
 public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int array[]=new int[n];

for(int i=0;i<n;i++){  
 array[i]=sc.nextInt();

}

System.out.println(largestSum(array,n));

}

public static int largestSum(int a[],int n){

int cumax=a[0];

int max=a[0];

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

curmax=Math.max(a[i],cumax+a[i]);

if(cumax>max){  
 max=cumax;

}

}

return max;

}

}

Example :

Input: nums = [-2,1,-3,4,-1,2,1,-5,4] Output: 6

Problem 8

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Solution:  
import java.util.\*;

class Findingnonoverlapping{  
 public static void main(String[] args){  
 Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

Interval a[]=new Interval[n];

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

int s=sc.nextInt();

int d=sc.nextInt();

a[i]=new Interval(s,d);

}

Findintervals(a);

}

public static void Findinterval(Interval a[]){

if(a.length<=0){

return;

}

Arrays.sort(a,new Comparator<Interval>(){

public int compare(Interval n1,Interval n2){

return n1.start-n2.start;

}

});

Int ind=0;

for(int i=1;i<a.length;i++){

if(a[ind].end>=a[i].start){

a[ind].end=Math.max(a[ind].end,a[i].end);

}

else{

ind++;

a[ind]=a[i];

}

}

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

System.out.println(“[“ +a[i].start+”,”+”a[i].end+”]”);

}

}

}

class Interval{  
 int start,end;

Interval(int start,int end){

This.start=start;

This.end=end;

}

}

Example :

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Program 9:

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.

Solution:

import java.util.\*;

class Sorting{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

int c0=0;

int c1=0;

int c2=0;

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

int s=sc.nextInt();

if(c0==0){

c0++;

}

else if(c1==1){

c1++;

}

else if(c2==1){

c2++;

}

}

int ind=0;

While(c0>0){

a[ind]=0;

c0--;

ind++;

}

While(c1>0){

a[ind]=1;

ind++;

c1--;

}

While(c2>0){

a[ind]=2;

ind++;

c2--;

}

System.out.println(Arrays.toString(a));

}

}

Example :

Input: nums = [2,0,2,1,1,0]Output: [0,0,1,1,2,2]

Program 10:

You are given an integer array prices where prices[i] is the price of a given stock on the i th day. On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day. Find and return the maximum profit you can achieve.

Solution:

Import java.util.\*;

class MaxProfit{

public static void main(String[] args){

Scanner s=new Scanner(System.in);

int n=sc.nextInt();

int array[]=new int[n];

for(int i=0;i<n;i++){  
 array[i]=sc.nextInt();

}

System.out.println(maxprofit(a));

}

Public static int maxprofit(int a[]){

int lsf = Integer.MAX\_VALUE;

int op = 0;

int pist = 0;

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

if(prices[i] < lsf){

lsf = prices[i];

}

pist = prices[i] - lsf;

if(op < pist){

op = pist;

}

}

return op;

}

Example :

Input: prices = [7,1,5,3,6,4]Output: 7

Program 11:

Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

Solution:

Import java.util.\*;

class MaxProfit{

public static void main(String[] args){

Scanner s=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){  
 a[i]=sc.nextInt();

}

int ind=0;

int peak=a[0];

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

if(a[i]>peak){  
 peak=Math.max(a[i],peak);

if(peak==a[i]){

ind=i;

}

}

}

System.out.println(ind);

}

}

Example :

Input: nums = [1,2,3,1] Output: 2

Program 12:

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

One simple solution:

Import java.util.\*;

Class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int m=sc.nextInt();

int a1[]=new int[n];

int a2[]=new int[m];

int res[]=new int[n+m];

int ind=0;

for(int i=0;i<n;i++){  
 res[ind]=sc.nextInt();

ind++;

}

For(int j=0;j<m;j++){

res[ind]=sc.nextInt();

ind++;

}

Arrays.sort(res);

if(res.length%2==0){

int mid=res.length/2;

int median=((res[mid]+res[mid+1])/2);

System.out.println(median);

}

Else{  
 System.out.println(res[mid]/2);

}

Another method:

import java.util.\*;

class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int m=sc.nextInt();

int nums1[]=new int[m];

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

nums1[i]=sc.nextInt();

}

int n=sc.nextInt();

int nums2[]=new int[n];

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

nums2[i]=sc.nextInt();

}

System.out.println(median(nums1,nums2));

}

static double median(int a[],int b[]){

int m=a.length;

int n=b.length;

int start=0;

int high=m;

while(start<=high){

int mid=(start+high)/2;

int lasize=mid;

int lbsize=(m+n+1)/2-lasize;

int la=(lasize>0) ? a[lasize-1]:Integer.MIN\_VALUE;

int lb=(lbsize>0) ? b[lbsize-1]:Integer.MIN\_VALUE;

int ra=(lasize<m) ? a[lasize]:Integer.MAX\_VALUE;

int rb=(lbsize<n) ? b[lbsize]:Integer.MAX\_VALUE;

if(la<=rb && lb<=ra){

if((m+n)%2==0){

return (Math.max(la,lb)+Math.min(ra,rb))/2.0;

}

return Math.max(la,lb);

}

else if(la>rb){

high=mid-1;

}

else{

start=mid+1;

}

}

return 0.0;

}

}

Example :

Input: nums1 = [1,3], nums2 = [2] Output: 2.00000

Problem 13:

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i th line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

Solution:

Java Program

Import java.util.\*;

class MaxWater{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

int l=0,r=a.length-1;

int max=0;

while(l<r){

int area=Math.min(a[l],a[r])\*(r-l);

max=Math.max(area,max);

if(a[l]<a[r]){

l++;

}

Else{

r--;

}

}

System.out.println(max);

}

}

Example :

Input: height = [1,8,6,2,5,4,8,3,7] Output: 49

Problem 14

Given a binary array nums, return the maximum number of consecutive 1's in the array.

Solution:

import java.util.\*;

class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

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

a[i]=sc.nextInt();

}

System.out.println(method(a,n));

}

Public static int method(int a[], int n)

{

int c=0;

int res=0

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

{

if (a[i]==0){

c = 0;

}

else

{

c++;

r = Math.max(res, c);

}

}

return r;

}

}

Example :

Input: nums = [1,1,0,1,1,1] Output: 3

Program 15:

Given an integer array nums and an integer k, return the kth largest element in the array. Note that it is the kth largest element in the sorted order, not the kth distinct element. You must solve it in O(n) time complexity

Solution:

Import java.util.\*;

Class Test{

Public static void main(String[] args){

Scanner sc=new Scanner(System.in);

Int n=sc.nextInt();

Int a[]=new int[n];

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

a[i]=sc.nextInt();

}

Arrays.sort(a);

Int k=sc.nextInt();

System .out.println(a[n-k]);

}

}

Example :

Input: nums = [3,2,1,5,6,4], k = 2 Output: 5