

DSA CODING PROBLEMS

Name : SRIKANTH M

Dept: AI&DS

Date:14/11/2024

1.Stock buy and sell

The cost of stock on each day is given in an array A[] of size N. Find all the segments of days on which you buy and sell the stock such that the sum of difference between sell and buy prices is maximized. Each segment consists of indexes of two elements, first is index of day on which you buy stock and second is index of day on which you sell stock.

Note: Since there can be multiple solutions, the driver code will print 1 if your answer is correct, otherwise, it will return 0. In case there's no profit the driver code will print the string "No Profit" for a correct solution.

Example 1:

Input:

N = 7

A[] = {100,180,260,310,40,535,695}

Output:

1

Explanation:

One possible solution is (0 3) (4 6)

We can buy stock on day 0,
and sell it on 3rd day, which will
give us maximum profit. Now, we buy
stock on day 4 and sell it on day 6.

CODE:

```
import java.io.*;
import java.util.*;

public class BuyAndSell {

    public static void main (String[] args) throws IOException {
```

```

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
int t = Integer.parseInt(br.readLine().trim());
while(t-->0){
    int n = Integer.parseInt(br.readLine().trim());
    int A[] = new int[n];
    String inputLine[] = br.readLine().trim().split(" ");
    for(int i=0; i<n; i++){
        A[i] = Integer.parseInt(inputLine[i]);
    }
    int p = 0;
    for(int i=0; i<n-1; i++)
        p += Math.max(0,A[i+1]-A[i]);
    Solution obj = new Solution();
    ArrayList<ArrayList<Integer>> ans = obj.stockBuySell(A, n);
    if(ans.size()==0)
        System.out.print("No Profit");
    else{
        int c=0;
        for(int i=0; i<ans.size(); i++)
            c += A[ans.get(i).get(1)]-A[ans.get(i).get(0)];
        if(c==p)
            System.out.print(1);
        else
            System.out.print(0);
    }System.out.println();
System.out.println("~");
}
}

```

```

}

class Solution{
    ArrayList<ArrayList<Integer>> stockBuySell(int A[], int n) {
        ArrayList<ArrayList<Integer>> res = new ArrayList<>();
        int i = 0;
        while(i < n-1){
            while(i < n-1 && A[i+1] <= A[i]){
                i++;
            }
            if(i == n-1) break;
            int buy = i++;
            while(i < n && A[i] >= A[i-1]){
                i++;
            }
            int sell = i-1;
            ArrayList<Integer> pair = new ArrayList<>();
            pair.add(buy);
            pair.add(sell);
            res.add(pair);
        }
        return res;
    }
}

```

Time Complexity : $O(n)$

Space Complexity : $O(n)$

OUTPUT:

```
C:\Windows\System32\cmd.e  X + v
D:\coding-java\Day-5>javac BuyAndSell.java
D:\coding-java\Day-5>java BuyAndSell
1
5
2 4 8 1 10
1
~
```

2.Coin Change (Count Ways)

Given an integer array coins[] representing different denominations of currency and an integer sum, find the number of ways you can make sum by using different combinations from coins[].

Note: Assume that you have an infinite supply of each type of coin. And you can use any coin as many times as you want.

Answers are guaranteed to fit into a 32-bit integer.

Examples:

Input: coins[] = [1, 2, 3], sum = 4

Output: 4

Explanation: Four Possible ways are: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3].

CODE:

```
import java.io.*;
import java.util.*;
class CoinChange {
    public static void main(String args[]) throws IOException {
        BufferedReader read = new BufferedReader(new
        InputStreamReader(System.in));
        int t = Integer.parseInt(read.readLine());
        while (t-- > 0) {
            String inputLine[] = read.readLine().trim().split(" ");
```

```

        int n = inputLine.length;
        int arr[] = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = Integer.parseInt(inputLine[i]);
        }
        int sum = Integer.parseInt(read.readLine());
        Solution ob = new Solution();
        System.out.println(ob.count(arr, sum));
    }
}

class Solution {
    public int count(int coins[], int sum) {
        int[] dp = new int[sum+1];
        dp[0] = 1;
        for(int num : coins){
            for(int j = num; j<=sum;j++){
                dp[j] += dp[j - num];
            }
        }
        return dp[sum];
    }
}

```

Time Complexity : $O(n \cdot \text{sum})$

Space Complexity : $O(\text{sum})$

OUTPUT:

```
D:\coding-java\Day-5>javac CoinChange.java
```

```
D:\coding-java\Day-5>java CoinChange
```

```
1
```

```
1 2 3
```

```
4
```

```
4
```

3.First and Last Occurrences

Given a sorted array arr with possibly some duplicates, the task is to find the first and last occurrences of an element x in the given array.

Note: If the number x is not found in the array then return both the indices as -1.

Examples:

Input: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Output: [2, 5]

Explanation: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

CODE:

```
import java.io.*;
```

```
import java.util.*;
```

```
class GFG {
```

```
    ArrayList<Integer> find(int arr[], int x) {
```

```
        ArrayList<Integer> res = new ArrayList<>();
```

```
        int f = findFirst(arr, x);
```

```
        int l = findLast(arr, x);
```

```
        res.add(f);
```

```
        res.add(l);
```

```
        return res;
```

```
    }
```

```
    private int findFirst(int[] arr, int x) {
```

```
int low = 0, high = arr.length - 1;
int first = -1;
while (low <= high) {
    int mid = low + (high - low) / 2;
    if (arr[mid] == x) {
        first = mid;
        high = mid - 1;
    } else if (arr[mid] < x) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}
return first;
}
```

```
private int findLast(int[] arr, int x) {
    int low = 0, high = arr.length - 1;
    int last = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x) {
            last = mid;
            low = mid + 1;
        } else if (arr[mid] < x) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
}
```

```

    }
    return last;
}
}

class FirstAndLast {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int testcases = Integer.parseInt(br.readLine());
        while (testcases-- > 0) {
            String line1 = br.readLine();
            String[] a1 = line1.trim().split("\\s+");
            int n = a1.length;
            int arr[] = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = Integer.parseInt(a1[i]);
            }
            int x = Integer.parseInt(br.readLine());
            GFG ob = new GFG();
            ArrayList<Integer> ans = ob.find(arr, x);
            System.out.println(ans.get(0) + " " + ans.get(1));
            System.out.println("~");
        }
    }
}

```

Time Complexity : $O(\log n)$

Space Complexity : $O(1)$

OUTPUT:


```
D:\coding-java\Day-5>javac FirstAndLast.java

D:\coding-java\Day-5>java FirstAndLast
1
1 2 3
4
-1 -1
~
```

4.Find Transition Point

Given a sorted array, arr[] containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed. If arr does not have any 1, return -1. If array does not have any 0, return 0.

Examples:

Input: arr[] = [0, 0, 0, 1, 1]

Output: 3

Explanation: index 3 is the transition point where 1 begins.

CODE:

```
import java.io.*;
import java.util.*;
class TransitionPoint {
    public static void main(String args[]) throws IOException {
        BufferedReader read = new BufferedReader(new
        InputStreamReader(System.in));
        int t = Integer.parseInt(read.readLine());
        while (t > 0) {
            String inputLine[] = read.readLine().trim().split(" ");
            int n = inputLine.length;
            int arr[] = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = Integer.parseInt(inputLine[i]);
```

```

    }

    Solution obj = new Solution();

    System.out.println(obj.transitionPoint(arr));

    System.out.println("~");

    t--;

    }

}

}

class Solution {

    int transitionPoint(int arr[]) {

        int n = arr.length;

        int low = 0;

        int high = n - 1;

        if (arr[0] == 1) return 0;

        if (arr[n - 1] == 0) return -1;

        while (low <= high) {

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

            if (arr[mid] == 1 && (mid == 0 || arr[mid - 1] == 0)) {

                return mid;

            } else if (arr[mid] == 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return -1;

    }

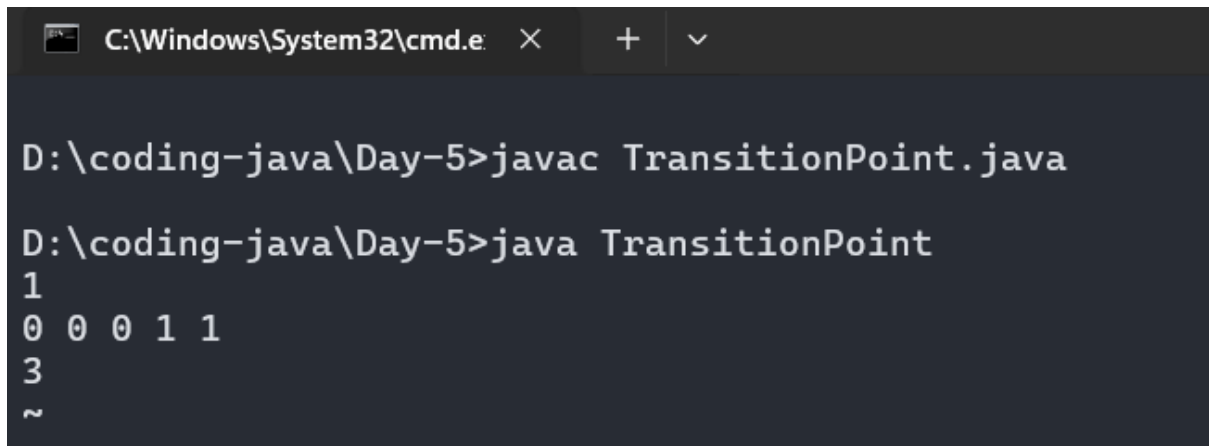
}

```

Time Complexity : $O(\log n)$

Space Complexity : $O(1)$

OUTPUT:



```
C:\Windows\System32\cmd.e  X  +  v

D:\coding-java\Day-5>javac TransitionPoint.java

D:\coding-java\Day-5>java TransitionPoint
1
0 0 0 1 1
3
~
```

5.Maximum Index

Given an array arr of positive integers. The task is to return the maximum of $j - i$ subjected to the constraint of $\text{arr}[i] \leq \text{arr}[j]$ and $i \leq j$.

Examples:

Input: $\text{arr}[] = [1, 10]$

Output: 1

Explanation: $\text{arr}[0] \leq \text{arr}[1]$ so $(j-i)$ is $1-0 = 1$.

CODE:

```
import java.io.*;
import java.util.*;

public class MaximumIndex {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int t = Integer.parseInt(scanner.nextLine().trim());

        while (t-- > 0) {

            String line = scanner.nextLine().trim();
```

```

String[] numsStr = line.split(" ");
int[] nums = new int[numsStr.length];
for (int i = 0; i < numsStr.length; i++) {
    nums[i] = Integer.parseInt(numsStr[i]);
}

Solution ob = new Solution();

System.out.println(ob.maxIndexDiff(nums));

System.out.println("~");
}

}

}

class Solution {
    int maxIndexDiff(int[] arr) {
        int n = arr.length;
        if (n == 1) return 0;
        int[] leftMin = new int[n];
        int[] rightMax = new int[n];
        leftMin[0] = arr[0];
        for (int i = 1; i < n; i++) {
            leftMin[i] = Math.min(arr[i], leftMin[i - 1]);
        }
        rightMax[n - 1] = arr[n - 1];
        for (int j = n - 2; j >= 0; j--) {
            rightMax[j] = Math.max(arr[j], rightMax[j + 1]);
        }
        int i = 0, j = 0, maxDiff = 0;
        while (i < n && j < n) {
            if (leftMin[i] <= rightMax[j]) {

```

```

        maxDiff = Math.max(maxDiff, j - i);

        j++;
    } else {
        i++;
    }
}

return maxDiff;
}
}

```

Time Complexity : $O(n)$

Space Complexity : $O(n)$

OUTPUT:

```

D:\coding-java\Day-5>javac MaximumIndex.java

D:\coding-java\Day-5>java MaximumIndex
1
1 10
1
~

```

6.First Repeating Element

Given an array `arr[]`, find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.

Note:- The position you return should be according to 1-based indexing.

Examples:

Input: `arr[] = [1, 5, 3, 4, 3, 5, 6]`

Output: 2

Explanation: 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.

Input: arr[] = [1, 2, 3, 4]

Output: -1

Explanation: All elements appear only once so answer is -1.

Constraints:

$1 \leq \text{arr.size} \leq 10^6$

$0 \leq \text{arr}[i] \leq 10^6$

CODE:

```
import java.io.*;
import java.util.*;
class FirstRepeat {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int t = Integer.parseInt(br.readLine().trim());
        while (t-- > 0) {
            String line = br.readLine();
            String[] tokens = line.split(" ");
            ArrayList<Long> array = new ArrayList<>();
            for (String token : tokens) {
                array.add(Long.parseLong(token));
            }
            int[] arr = new int[array.size()];
            int idx = 0;
            for (long i : array) arr[idx++] = (int)i;
            Solution obj = new Solution();
            System.out.println(obj.firstRepeated(arr));
            System.out.println("~");
        }
    }
}
```

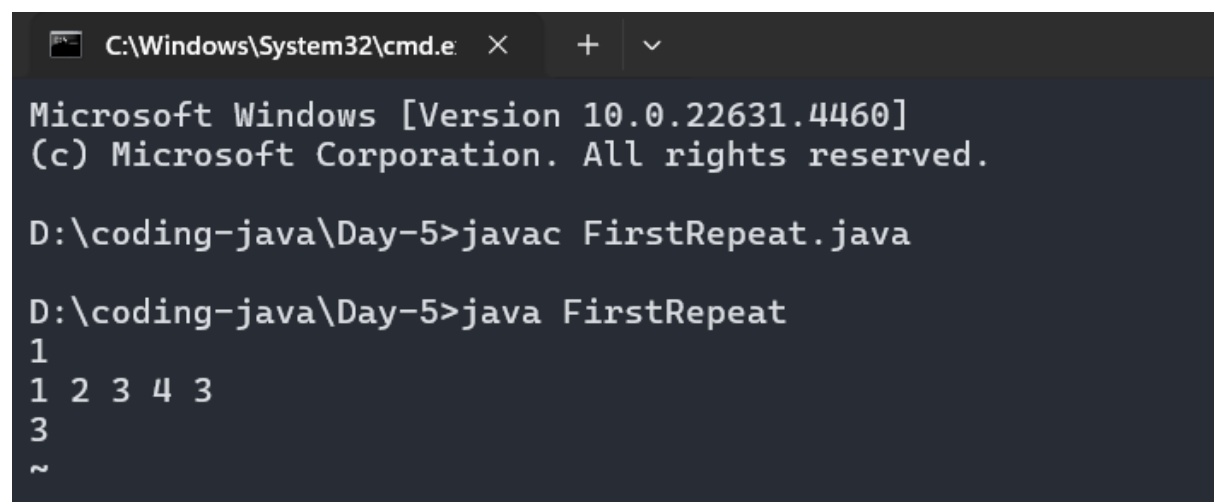
```
}
```

```
class Solution {  
    public static int firstRepeated(int[] arr) {  
        HashMap<Integer, Integer> map = new HashMap<>();  
        int minI = Integer.MAX_VALUE;  
        for(int i=0;i<arr.length;i++){  
            if(map.containsKey(arr[i])){  
                minI = Math.min(minI, map.get(arr[i]));  
            }  
            else{  
                map.put(arr[i],i);  
            }  
        }  
        return (minI == Integer.MAX_VALUE) ? -1 : minI + 1;  
    }  
}
```

Time Complexity : $O(n)$

Space Complexity : $O(n)$

OUTPUT:



```
C:\Windows\System32\cmd.e  X  +  v  
Microsoft Windows [Version 10.0.22631.4460]  
(c) Microsoft Corporation. All rights reserved.  
  
D:\coding-java\Day-5>javac FirstRepeat.java  
  
D:\coding-java\Day-5>java FirstRepeat  
1  
1 2 3 4 3  
3  
~
```

7.Remove Duplicates Sorted Array

Given a **sorted** array **arr**. Return the size of the modified array which contains only distinct elements.

Note:

1. Don't use set or HashMap to solve the problem.
2. You **must** return the modified array **size only** where distinct elements are present and **modify** the original array such that all the distinct elements come at the beginning of the original array.

Examples :

Input: arr = [2, 2, 2, 2, 2]

Output: [2]

Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should **return 1** after modifying the array, the driver code will print the modified array elements.

CODE:

```
import java.io.*;
import java.util.*;

public class RemoveDuplicates {

    public static void main(String[] args) throws Exception {

        Scanner sc = new Scanner(System.in);

        int t = Integer.parseInt(sc.nextLine());

        while (t-- > 0) {

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

            String input = sc.nextLine();

            StringTokenizer st = new StringTokenizer(input);

            while (st.hasMoreTokens()) {

                arr.add(Integer.parseInt(st.nextToken()));

            }

            Solution ob = new Solution();
```



```

        int ans = ob.remove_duplicate(arr);
        for (int i = 0; i < ans; i++) {
            System.out.print(arr.get(i) + " ");
        }
        System.out.println();
    System.out.println("~");
}

    sc.close();
}
}

class Solution {
    public int remove_duplicate(List<Integer> arr) {
        if (arr.size() == 0) return 0;
        int j = 0;
        for (int i = 1; i < arr.size(); i++) {
            if (!arr.get(i).equals(arr.get(j))) {
                j++;
                arr.set(j, arr.get(i));
            }
        }
        return j + 1;
    }
}

```

Time Complexity : $O(n)$

Space Complexity : $O(n)$

OUTPUT:

```
C:\Windows\System32\cmd.e  ×  +  ∨

D:\coding-java\Day-5>javac RemoveDuplicates.java

D:\coding-java\Day-5>java RemoveDuplicates
1
2 2 2 2 2
2
~
```

8.Wave Array

Given a **sorted** array **arr[]** of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that $arr[1] \geq arr[2] \leq arr[3] \geq arr[4] \leq arr[5] \dots$.

If there are multiple solutions, find the lexicographically smallest one.

Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array.

Examples:

Input: `arr[] = [1, 2, 3, 4, 5]`

Output: `[2, 1, 4, 3, 5]`

Explanation: Array elements after sorting it in the waveform are 2, 1, 4, 3, 5.

CODE:

```
import java.io.*;
import java.util.*;
import java.util.Arrays;
class IntArray {
    public static int[] input(BufferedReader br, int n) throws IOException {
        String[] s = br.readLine().trim().split(" ");
        int[] a = new int[n];
        for (int i = 0; i < n; i++) a[i] = Integer.parseInt(s[i]);
        return a;
    }
}
```

```

    }

    public static void print(int[] a) {
        for (int e : a) System.out.print(e + " ");
        System.out.println();
    }

    public static void print(ArrayList<Integer> a) {
        for (int e : a) System.out.print(e + " ");
        System.out.println();
    }
}

class WaveArray {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int t;
        t = Integer.parseInt(br.readLine());
        while (t-- > 0) {
            String inputLine[] = br.readLine().trim().split(" ");
            int n = inputLine.length;
            int arr[] = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = Integer.parseInt(inputLine[i]);
            }
            Solution obj = new Solution();
            obj.convertToWave(arr);
            StringBuffer sb = new StringBuffer("");
            for (int i : arr) {
                sb.append(i + " ");
            }

```

```

        System.out.println(sb.toString());
        System.out.println("~");
    }
}
}
class Solution {
    public static void convertToWave(int[] arr) {
        int n = arr.length;
        int temp;
        for(int i=0;i<n-1;i+=2){
            temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
        }
    }
}

```

Time Complexity : $O(n)$

Space Complexity : $O(1)$

OUTPUT:

```

D:\coding-java\Day-5>javac WaveArray.java
D:\coding-java\Day-5>java WaveArray
1
1 2 3 4 5
2 1 4 3 5
~

```