

DSA PRACTICE – DAY 5

Name: Dhejan R

Reg No: 22IT022

Date: 14/11/2024

1. Stock Buy and Sell

Code Solution:

```
class Solution {
    ArrayList<ArrayList<Integer>> stockBuySell(int A[], int n) {
        ArrayList<ArrayList<Integer>> result=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;
            i++;
            while (i<n && A[i]>=A[i-1]) {
                i++;}
            int sell=i-1;
            ArrayList<Integer> buySellPair = new ArrayList<>();
            buySellPair.add(buy);
            buySellPair.add(sell);
            result.add(buySellPair);
        }
        return result;
    }
}
```

Output:

The screenshot displays a coding platform interface. On the left, the problem 'Stock buy and sell' is shown with a difficulty of 'Medium', an accuracy of '29.18%', and 277K+ submissions. The problem description states: '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.' Below the description, the 'Output Window' shows 'Compilation Results' for 'Y.O.G.I. (AI Bot)', indicating 'Problem Solved Successfully'. A summary box shows 'Test Cases Passed: 142 / 142', 'Attempts: 1 / 1', and 'Accuracy: 100%'. On the right, the code editor shows the Java solution for the problem, which is identical to the code provided in the 'Code Solution' section.

Time complexity: $O(n)$

Space Complexity: $O(n)$

2. Minimize heights II

Code Solution:

```
class Solution {
    public int count(int coins[], int sum) {
        // code here.
        int[] dp=new int[sum+1];
        dp[0]=1;
        for (int coin:coins) {
            for (int j=coin; j<=sum; j++) {
                dp[j]+=dp[j-coin];
            }
        }
        return dp[sum];
    }
}
```

Output:

The screenshot displays a code editor interface for the 'Coin Change (Count Ways)' problem. The problem description states: '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 provided are `coins = [1, 2, 3]` and `sum = 4`, with the output being 4.

The code editor shows the following Java code:

```
28
29
30 // User function Template for Java
31
32 class Solution {
33     public int count(int coins[], int sum) {
34         // code here.
35         int[] dp=new int[sum+1];
36         dp[0]=1;
37         for (int coin:coins) {
38             for (int j=coin; j<=sum; j++) {
39                 dp[j]+=dp[j-coin];
40             }
41         }
42         return dp[sum];
43     }
44 }
```

The 'Output Window' shows 'Compilation Completed' for the input `1 2 3` and `4`, with the output being `4`.

Time Complexity: $O(m*n)$

Space Complexity: $O(m)$

3. First and Last Occurrences

Code Solution:

```
class GFG {
    ArrayList<Integer> find(int arr[], int x) {
        // code here
        ArrayList<Integer> res=new ArrayList<>(Collections.nCopies(2,-1));
        int count=0;
        for (int i=0; i<arr.length; i++) {
            if (arr[i]==x) {
                if (res.get(0)==-1) {
                    res.set(0,i);
                }
                count++;
            }
        }

        if (count>0) {
            res.set(1, res.get(0)+count-1);
        }
        return res;
    }
}
```

Output:

First and Last Occurrences

Difficulty: Medium Accuracy: 37.36% Submissions: 271K+ Points: 4

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]

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed: **1120 / 1120**

Attempts: Correct / Total
1 / 1

Accuracy: 100%

```
1 // Driver Code Starts
2 // User function Template for Java
3
4
5
6
7
8
9
10
11 class GFG {
12     ArrayList<Integer> find(int arr[], int x) {
13         // code here
14         ArrayList<Integer> res=new ArrayList<>(Collections.nCopies(2,-1));
15         int count=0;
16         for (int i=0; i<arr.length; i++) {
17             if (arr[i]==x) {
18                 if (res.get(0)==-1) {
19                     res.set(0,i);
20                 }
21                 count++;
22             }
23         }
24
25         if (count>0) {
26             res.set(1, res.get(0)+count-1);
27         }
28         return res;
29     }
30 }
31
32 // Driver Code Ends
```

Time complexity: $O(n)$

Space Complexity: $O(1)$

4. Find Transition Point

Code Solution:

```
class Solution {
    int transitionPoint(int arr[]) {
        // code here
        int left=0;
        int right=arr.length-1;

        while(left<=right){
            int mid=left+(right-left)/2;
            if(arr[mid]==1){
                if (mid==0 || arr[mid-1]==0){
                    return mid;
                }
            }
            else {
                right=mid-1;
            }
        }
        else{
            left=mid+1;
        }
    }
    return -1;
}
```

Output:

The screenshot displays a coding platform interface. On the left, the problem 'Find Transition Point' is shown with its difficulty (Easy), accuracy (37.9%), and submission statistics (268K+). The problem description states: '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 provided are: Input: arr[] = [0, 0, 0, 1, 1], Output: 3. The 'Output Window' shows 'Problem Solved Successfully' with 1115/1115 test cases passed and 100% accuracy. On the right, the C++ code solution is displayed, which is a binary search algorithm to find the first index of 1 in a sorted array of 0s and 1s.

```
26
27
28 class Solution {
29     int transitionPoint(int arr[]) {
30         // code here
31         int left=0;
32         int right=arr.length-1;
33
34         while(left<=right){
35             int mid=left+(right-left)/2;
36             if(arr[mid]==1){
37                 if (mid==0 || arr[mid-1]==0){
38                     return mid;
39                 }
40             }
41             else {
42                 right=mid-1;
43             }
44         }
45         else{
46             left=mid+1;
47         }
48     }
49     return -1;
50 }
```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

5. First Repeating Element

Code Solution:

```
class Solution {
    // Function to return the position of the first repeating element.
    public static int firstRepeated(int[] arr) {
        // Your code here
        HashMap<Integer,Integer> map=new HashMap<>();
        int minIndex=Integer.MAX_VALUE;
        for (int i=arr.length-1; i>=0; i--) {
            if (map.containsKey(arr[i])) {
                minIndex=i;
            } else {
                map.put(arr[i], i);
            }
        }
        return minIndex==Integer.MAX_VALUE?-1:minIndex+1;
    }
}
```

Output:

First Repeating Element

Difficulty: Easy Accuracy: 32.57% Submissions: 268K+ Points: 2

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

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed	Attempts : Correct / Total
1115 / 1115	1 / 1
	Accuracy : 100%

Time Complexity: $O(n)$

Space Complexity: $O(n)$

6. Remove Duplicates Sorted Array

Code Solution:

```
class Solution {
    // Function to remove duplicates from the given array
    public int remove_duplicate(List<Integer> arr) {
```

```

// Code Here
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));
    }
}
while (arr.size()>j+1) {
    arr.remove(arr.size()-1);
}

return j+1;
}
}

```

Output:

Remove Duplicates Sorted Array

Difficulty: Easy Accuracy: 38.18% Submissions: 259K+ Points: 2

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 = [1,1,2]

Output: [1,2]

Return: 2

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

[Suggest Feedback](#)

Test Cases Passed	Attempts : Correct / Total
1115 / 1115	1 / 1
	Accuracy : 100%

```

// User function Template for Java
class Solution {
    // Function to remove duplicates from the given array
    public int remove_duplicate(List<Integer> arr) {
        // Code Here
        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));
            }
        }
        while (arr.size()>j+1) {
            arr.remove(arr.size()-1);
        }

        return j+1;
    }
}

```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

7. Maximum Index

Code Solution

```

class Solution {
    // Function to find the maximum index difference.
    int maxIndexDiff(int[] arr) {
        // Your code here
        int n=arr.length;
        int[] LMin=new int[n];

```

```

int[] RMax=new int[n];
LMin[0]=arr[0];
for (int i=1; i<n; i++) {
    LMin[i]=Math.min(arr[i], LMin[i-1]);
}
RMax[n-1]=arr[n-1];
for (int j=n-2; j>=0; j--) {
    RMax[j]=Math.max(arr[j], RMax[j+1]);
}
int i=0, j=0;
int maxDiff=-1;
while (i<n && j<n) {
    if (LMin[i]<=RMax[j]) {
        maxDiff=Math.max(maxDiff, j-i);
        j++;
    } else {
        i++;
    }
}
return maxDiff;
}
}

```

Output:

Maximum Index

Difficulty: Medium Accuracy: 24.5% Submissions: 258K+ Points: 4

Given an array `arr` of positive integers. The task is to return the maximum of `j-i` subjected to the constraint of `arr[i] ≤ arr[j]` and `i ≤ j`.

Examples:

Input: `arr[] = [1, 10]`

Output: 1

Explanation: `arr[0] ≤ arr[1]` so `(j-i)` is `1-0 = 1`.

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed	Attempts : Correct / Total
1115 / 1115	1 / 5
	Accuracy : 20%

```

28
29
30 // User function Template for Java
31 class Solution {
32     // Function to find the maximum index difference.
33     int maxIndexDiff(int[] arr) {
34         // Your code here
35         int n=arr.length;
36         int[] LMin=new int[n];
37         int[] RMax=new int[n];
38         LMin[0]=arr[0];
39         for (int i=1; i<n; i++) {
40             LMin[i]=Math.min(arr[i], LMin[i-1]);
41         }
42         RMax[n-1]=arr[n-1];
43         for (int j=n-2; j>=0; j--) {
44             RMax[j]=Math.max(arr[j], RMax[j+1]);
45         }
46         int i=0, j=0;
47         int maxDiff=-1;
48         while (i<n && j<n) {
49             if (LMin[i]<=RMax[j]) {
50                 maxDiff=Math.max(maxDiff, j-i);
51                 j++;
52             } else {
53                 i++;
54             }
55         }
56         return maxDiff;
57     }
58 }

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

7.

Code Solution

```
class Solution {
    public static void convertToWave(int[] arr) {
        // code here
        int n=arr.length;
        for (int i=0; i<n-1; i+=2) {
            if (arr[i]<arr[i+1]) {
                int temp=arr[i];
                arr[i]=arr[i+1];
                arr[i+1]=temp;
            }
            if (i+1<n-1 && arr[i+1]>arr[i+2]) {
                int temp=arr[i+1];
                arr[i+1]=arr[i+2];
                arr[i+2]=temp;
            }
        }
    }
}
```

Output:

The screenshot displays a coding platform interface for the 'Wave Array' problem. The problem description states: '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] >= arr[2] <= arr[3] >= arr[4] <= arr[5]..... 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:'. The interface shows the solution code in Java, which is the same as the one provided in the previous block. The 'Compilation Results' section indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1120 / 1120' and 'Attempts: Correct / Total: 1 / 1'. The 'Accuracy' is listed as '100%'. The 'Suggest Feedback' link is also visible.

Time Complexity: $O(n)$

Space Complexity: $O(1)$