

DSA PRACTICE - 5

NAME : SIDHARTH N

ROLL NO : 22AD126

1. Buy and Sell Stock

Output Window

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

Problem Solved Successfully

Test Cases Passed: 142 / 142

Attempts: Correct / Total: 1 / 1

Accuracy: 100%

Points Scored: 4 / 4

Time Taken: 0.09

Your Total Score: 30

Solve Next

You and your books Stock Buy and Sell - Multiple Transaction Allowed Max Sum without Adjacents

```
1 // User function Template for Java
2
3 class Solution {
4     // function to find the days of buying and selling stock for max profit.
5     ArrayList<ArrayList<Integer>> stockBuySell(int a[], int n) {
6         ArrayList<ArrayList<Integer>> result = new ArrayList<>();
7
8         for (int i = 0; i < n - 1; i++) {
9             if (a[i] < a[i + 1]) {
10                 int buyDay = i;
11
12                 int sellDay = buyDay;
13                 for (int j = buyDay + 1; j < n; j++) {
14                     if (j == n - 1 || a[j] > a[j + 1]) {
15                         sellDay = j;
16                         break;
17                     }
18                 }
19
20                 ArrayList<Integer> pair = new ArrayList<>();
21                 pair.add(buyDay);
22                 pair.add(sellDay);
23                 result.add(pair);
24
25                 i = sellDay;
26             }
27         }
28
29         return result;
30     }
31 }
```

Time Complexity : $O(n^2)$

2. Wave Array

Output Window

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

Problem Solved Successfully

Test Cases Passed: 1120 / 1120

Attempts: Correct / Total: 1 / 1

Accuracy: 100%

Points Scored: 2 / 2

Time Taken: 0.81

Your Total Score: 32

Solve Next

Three way partitioning Sort by Absolute Difference Convert an array to reduced form

```
1 // User function Template for Java
2
3 class Solution {
4     public static void convertToWave(int[] arr) {
5         // code here
6         int n = arr.length;
7         int mid = n / 2;
8         for (int i = 1; i < n; i++) {
9             if (i % 2 == 1) {
10                 int temp = arr[i];
11                 arr[i] = arr[i - 1];
12                 arr[i - 1] = temp;
13             }
14         }
15     }
16 }
```

Time Complexity : $O(n)$

3.Remove duplicates sorted array

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed: 1115 / 1115

Attempts: Correct / Total: 1 / 2

Accuracy: 50%

Points Scored: 2 / 2

Time Taken: 1.36

Your Total Score: 34

Solve Next

[Sort subsequence of size 3](#) [Sort 0s, 1s and 2s](#) [Sort Unsorted Subarray](#)

```
1 // Driver Code Starts
2
3 // User function Template for Java
4
5 class Solution {
6     // Function to remove duplicates from the given array
7     public int remove_duplicate(List<Integer> arr) {
8         // Code here
9         int temp = 0;
10        int ans = 1;
11        for(int i = 1; i<arr.size(); i++){
12            if(!arr.get(i).equals(arr.get(temp))){
13                temp++;
14                arr.set(temp, arr.get(i));
15                ans++;
16            }
17        }
18        return ans;
19    }
20 }
```

Time complexity : $O(n)$

4.Maximum Index

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`.

Input: `arr[] = [34, 8, 10, 3, 2, 80, 30, 33, 1]`
Output: 6
Explanation: In the given array `arr[1] < arr[7]` satisfying the required condition (`arr[i] ≤ arr[j]`) thus giving the maximum difference of `j - i` which is `6(7-1)`.

Expected Time Complexity: $O(n)$
Expected Auxiliary Space: $O(n)$

Constraints:

```
1 // Driver Code Starts
2
3 // User function Template for Java
4
5 class Solution {
6     int maxIndexDiff(int[] arr) {
7         int n = arr.length;
8         if (n < 2) {
9             return 0;
10        }
11
12        int[] leftMin = new int[n];
13        int[] rightMax = new int[n];
14
15        leftMin[0] = arr[0];
16        for (int i = 1; i < n; i++) {
17            leftMin[i] = Math.min(leftMin[i - 1], arr[i]);
18        }
19
20        rightMax[n - 1] = arr[n - 1];
21        for (int j = n - 2; j >= 0; j--) {
22            rightMax[j] = Math.max(rightMax[j + 1], arr[j]);
23        }
24
25        int i = 0, j = 0, maxDiff = 0;
26        while (i < n && j < n) {
27            if (leftMin[i] < rightMax[j]) {
28                maxDiff = Math.max(maxDiff, j - i);
29                j++;
30            } else {
31                i++;
32            }
33        }
34
35        return maxDiff;
36    }
37 }
```

Time Complexity : $O(n)$

5. First repeating element

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 4

Accuracy : 25%

Points Scored
2 / 2

Time Taken
1.81

Your Total Score: 40

Solve Next

Sorted subsequence of size 3 Array Duplicates

Frequencies of Limited Range Array Elements

```
1 // Driver Code Starts
2
3
4
5 // User function Template for Java
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43 class Solution {
44     public static int firstRepeated(int[] arr) {
45         Map<Integer, Integer> hashmap = new HashMap<>();
46         int ele = -1;
47
48         for (int i : arr) {
49             hashmap.put(i, hashmap.getOrDefault(i, 0) + 1);
50         }
51
52         for (int i = 0; i < arr.length; i++) {
53             if (hashmap.get(arr[i]) >= 2) {
54                 return i+1;
55             }
56         }
57         return -1;
58     }
59 }
60
```

Time complexity : $O(n)$

6. First Transition point

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1

Accuracy : 100%

Points Scored
2 / 2

Time Taken
0.48

Your Total Score: 42

Solve Next

Index of an Extra Element Missing in Array Left most and right most index

```
1 // Driver Code Starts
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28 class Solution {
29     int transitionPoint(int arr[]) {
30         // code here
31         for (int i = 0; i < arr.length; i++) {
32             if (arr[i] == 1) {
33                 return i;
34             }
35         }
36         return -1;
37     }
38 }
39
```

Time Complexity : $O(n)$

7. First and Last Occurrences

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]

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

Input: arr[] = [1, 3, 5, 5, 5, 5, 7, 123, 125], x = 7

Output: [6, 6]

Explanation: First and last occurrence of 7 is at index 6

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

Output: [-1, -1]

Explanation: No occurrence of 4 in the array, so, output is [-1, -1]

Java (1.8) Average Time: 15m Start Timer

```
1 // User function Template for Java
2
3 class GFG {
4
5     ArrayList<Integer> find(int arr[], int x) {
6         int n = arr.length;
7         int i = 0;
8         int j = n - 1;
9         int first = -1;
10        int last = -1;
11        boolean found_first = false;
12        boolean found_last = false;
13
14        while (i <= j) {
15            if (!found_first) {
16                if (arr[i] == x) {
17                    first = i;
18                    found_first = true;
19                } else {
20                    i++;
21                }
22            }
23            if (!found_last) {
24                if (arr[j] == x) {
25                    last = j;
26                    found_last = true;
27                } else {
28                    j--;
29                }
30            }
31            if (found_first && found_last) break;
32        }
33
34        ArrayList<Integer> result = new ArrayList<>();
35        result.add(first);
36        result.add(last);
37        return result;
38    }
39 }
```

Time complexity : $O(n)$