22IT023 DHONI R

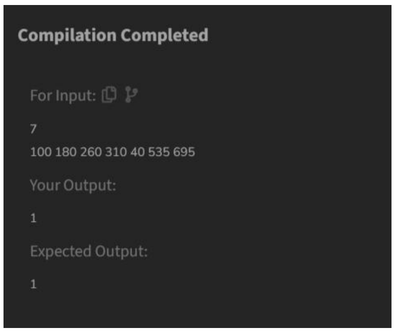
14/11/24

DSA Practice

1) Stock buy and Sell

```
class Solution{
   //Function to find the days of buying and selling stock for max profit.
   ArrayList<ArrayList<Integer> > stockBuySell(int A[], int n){
      // code here


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

      for(int i=0;i<n-1;i++){
         if(A[i+1]>A[i])
         {
            ArrayList<Integer> al = new ArrayList<>();

            al.add(i);
            al.add(i+1);
            ans.add(al);
         }
      }
      return ans;
   }
}
```

Output



```
Compilation Completed

For Input:
7
100 180 260 310 40 535 695
Your Output:
1
Expected Output:
1
```

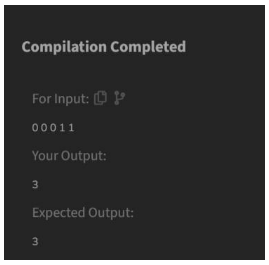
Time Complexity : O(N)

2) **Find Transition Point**

```
class Solution {
   int transitionPoint(int arr[]) {
      // code here

      int left, right, mid, n, index;

      n = arr.length;
      left = 0;
      right = n-1;
      index = -1;
```

```
        while(left <= right){
            mid = (left+right)/2;

            if(arr[mid] == 1){
                index = mid;
                right = mid-1;
            }
            else if(arr[mid] == 0){
                left = mid+1;
            }
        }

        return index;
    }
}
```

Output



```
Compilation Completed

For Input:
0 0 0 1 1
Your Output:
3
Expected Output:
3
```

Time Complexity : O(Log n)

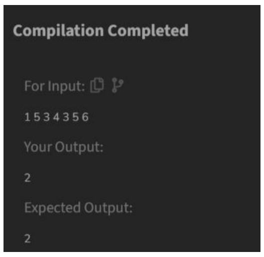
3) First Repeating Elements

Code:

```
class Solution {
    // Function to return the position of the first repeating element.
    public static int firstRepeated(int[] arr) {
        HashMap<Integer, Integer> map = new HashMap<>();
        int index = -1;
        for(int x : arr)
            map.put(x, map.getOrDefault(x,0)+1);
        for(int i = 0; i<arr.length; i++ ){
            if(map.get(arr[i])>1){
                index = i+1;
                break;
            }

        }
        return index;
    }
}
```

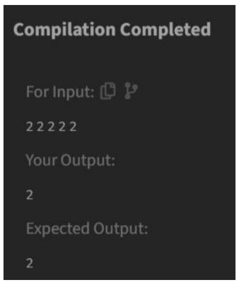
Time Complexity : O(N)

Output:


```
Compilation Completed

For Input:
1 5 3 4 3 5 6
Your Output:
2
Expected Output:
2
```

4) Remove duplicates sorted array

Code:
```
class Solution {
   // Function to remove duplicates from the given array
   public int remove_duplicate(List<Integer> arr) {
      // Initializing pointer i to the first element of the array
      int i = 0, n = arr.size();

      // Iterating through the array
      for (int j = 1; j < n; j++) {
         // If the current element is not equal to the previous element,
         // then increment i and update arr[i] with the current element
         if (!arr.get(j).equals(arr.get(i))) {
            i++;
            arr.set(i, arr.get(j));
         }
      }
      // Returning the length of the array after removing duplicates
      return i + 1;
   }
}
```
Output:


```
Compilation Completed

For Input:
2 2 2 2 2
Your Output:
2
Expected Output:
2
```

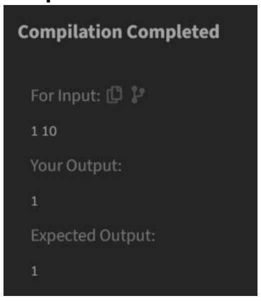
Time Complexity : O(N)

5) Maximum index

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

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

Output:



```
Compilation Completed

For Input:
1 10
Your Output:
1
Expected Output:
1
```

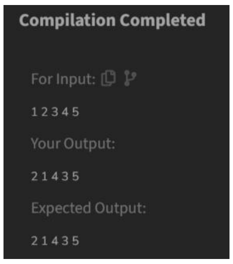
Time Complexity : O(N)

6) Wave Array

Code:

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

Output:



```
Compilation Completed

For Input:
1 2 3 4 5
Your Output:
2 1 4 3 5
Expected Output:
2 1 4 3 5
```

Time Complexity : O(N)

7) First and last occurance
Code:

```
public class FirstLast {nb
public static int findFirstOccurrence(int[] arr, int x) {
int left = 0, right = arr.length - 1;
int result = -1;

while (left <= right) {
int mid = left + (right - left) / 2;

if (arr[mid] == x) {
result = mid;
right = mid - 1;
} else if (arr[mid] < x) {
left = mid + 1;
} else {
right = mid - 1;
}
}
return result;
}
public static int findLastOccurrence(int[] arr, int x) {
int left = 0, right = arr.length - 1;
int result = -1;
while (left <= right) {
int mid = left + (right - left) / 2;

if (arr[mid] == x) {
result = mid;
left = mid + 1;
} else if (arr[mid] < x) {

left = mid + 1;
} else {
right = mid - 1;
}
}
return result;
}
public static int[] findFirstAndLast(int[] arr, int x) {
int[] result = new int[2];
result[0] = findFirstOccurrence(arr, x);
result[1] = findLastOccurrence(arr, x);
return result;
}
public static void main(String[] args) {
int[] arr = {1, 3, 5, 5, 5, 5, 67, 123, 125};
int x = 5;
int[] result = findFirstAndLast(arr, x);
System.out.println("First and last occurrences of " + x + ": [" + result[0] + ", " +
result[1] + "]");
}
}
```

Output: First and last occurrences of 5: [2, 5]

Time Complexity :O(Logn)

8) Coin Change (Count ways)

Code:
```
import java.util.*;

class CoinChange {
static int count(int[] coins, int sum, int n, int[][] dp) {
if (sum == 0)
return dp[n][sum] = 1;
if (n == 0 || sum < 0)
return 0;
if (dp[n][sum] != -1)
return dp[n][sum];
return dp[n][sum] = count(coins, sum - coins[n - 1], n, dp) + count(coins, sum, n - 1,
dp);
}

public static void main(String[] args) {

int tc = 1;
while (tc != 0) {
int n = 3, sum = 4;
int[] coins = {1, 2, 3};
int[][] dp = new int[n + 1][sum + 1];
for (int[] row : dp)
Arrays.fill(row, -1);
int res = count(coins, sum, n, dp);
System.out.println(res);
tc--;
}
}
}
```

Output: 4
Time Complexity: O(n2)