**14-11-2024 DSA PRACTICE PROBLEMS SET 5**

**1. Stock Buy and Sell**

**CODE:**

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]>=A[i+1]){

i++;

}

if (i == n - 1) break;

int buy = i++;

while (i < n && A[i]>=A[i-1]){

i++;

}

int sell = i-1;

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

seg.add(buy);

seg.add(sell);

res.add(seg);

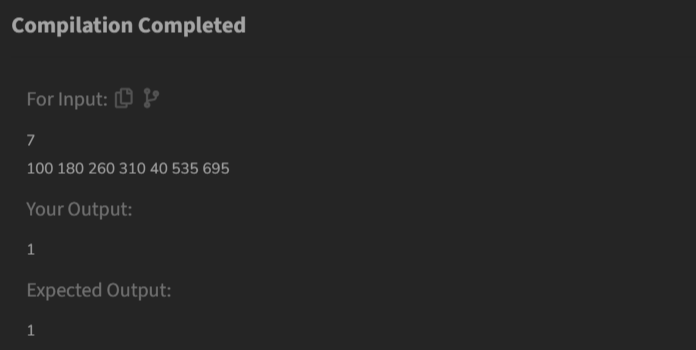
}

return res;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**2. Coin Change**

**CODE:**

class Solution {

public static boolean makeChanges(int N, int K, int target, int[] coins) {

boolean[][] dp = new boolean[target + 1][K + 1];

dp[0][0] = true;

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

for (int j = 0; j <= K; j++) {

for (int coin : coins) {

if (coin <= i && j > 0

&& dp[i - coin][j - 1]) {

dp[i][j] = true;

break;

}

}

}

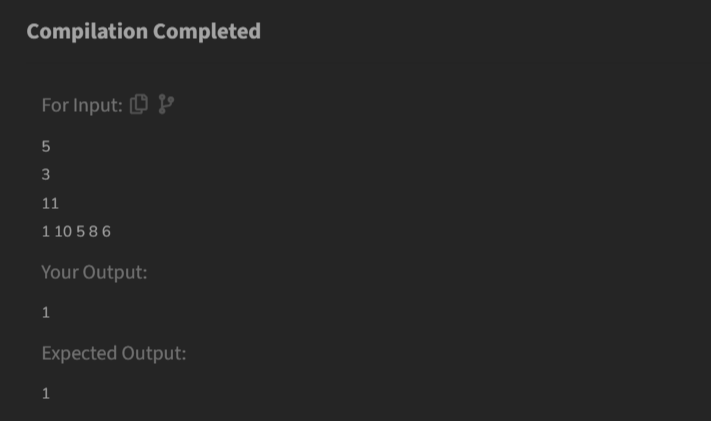
}

return dp[target][K];

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(NKT)**

**3. First And Last Occurrences**

**CODE:**

//{ Driver Code Starts

// Initial Template for Java

import java.io.\*;

import java.util.\*;

// } Driver Code Ends

// User function Template for Java

class GFG {

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

int l = arr.length;

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

int low = 0;

int high = l-1;

int res=-1;

while (low<=high){

int mid = (low+high)/2;

if (arr[mid]<x){

low = mid+1;

}

else if (arr[mid]>x){

high = mid-1;

}

else{

res = mid;

high = mid-1;

}

}

ans.add(res);

low = 0;

high = l-1;

res=-1;

while (low<=high){

int mid = (low+high)/2;

if (arr[mid]<x){

low = mid+1;

}

else if (arr[mid]>x){

high = mid-1;

}

else{

res = mid;

low = mid+1;

}

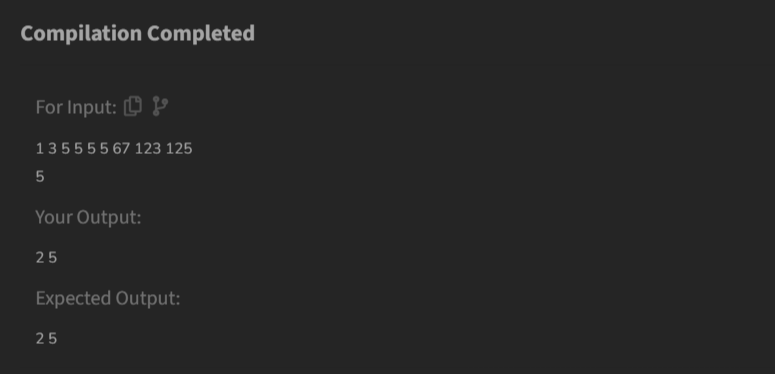
}

ans.add(res);

return ans;

}

}

**OUTPUT:** 

**TIME COMPLEXITY: O(log(N))**

**4. Find Transition Point**

**CODE:**

class Solution {

int transitionPoint(int arr[]) {

int l = arr.length;

int low = 0;

int high = l-1;

while(low<=high){

int mid = (low+high)/2;

if (arr[mid]==0){

low = mid+1;

}

else{

if (arr[mid]==1){

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

return mid;

}

else{

high = mid-1;

}

}

}

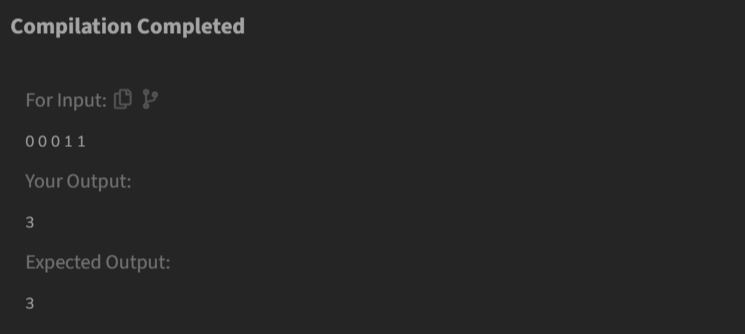
}

return -1;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(log(N))**

**5. First Repeating Element**

**CODE:**

class Solution {

public static int firstRepeated(int[] arr) {

HashSet<Integer> hash = new HashSet<>();

int l = arr.length;

int Min=-1;

for (int i=l-1;i>=0;i--){

if (hash.contains(arr[i])){

Min = i;

}

else{

hash.add(arr[i]);

}

}

if(Min!=-1){

return Min+1;

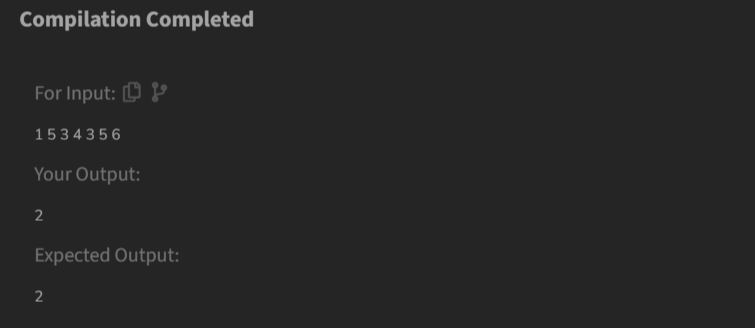
}

return -1;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**6. Remove Duplicates Sorted Array**

**CODE:**

class Solution {

public int remove\_duplicate(List<Integer> arr) {

int n = arr.size();

if (n <= 1)

return n;

int ind = 1;

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

if (arr.get(i) != arr.get(i-1)) {

arr.set(ind,arr.get(i));

ind++;

}

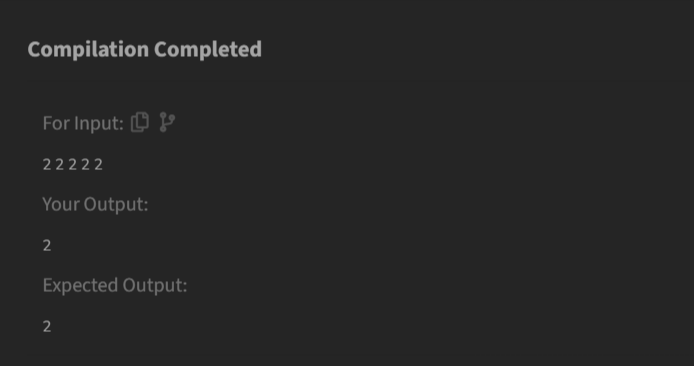
}

return ind;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**7. Maximum Index**

**CODE:**

class Solution {

int maxIndexDiff(int[] arr) {

Stack<Integer> st = new Stack<>();

int N = arr.length;

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

if(st.isEmpty() || arr[st.peek()]>arr[i])

st.push(i);

}

int max = 0;

int dif;

int i = N-1;

while(i>=0){

if(!st.isEmpty() && arr[st.peek()] <= arr[i]){

dif = i - st.pop();

if(dif>max){

max = dif;

}

continue;

}

i--;

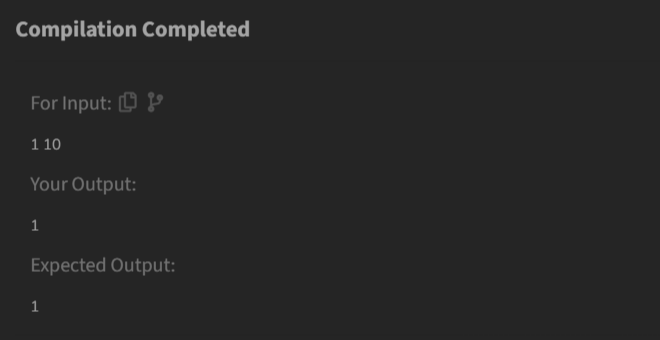
}

return max;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**8. Wave Array**

**CODE:**

class Solution {

public static void convertToWave(int[] arr) {

int n = arr.length;

for (int i = 0; i < n; i+=2){

if(i > 0 && arr[i - 1] > arr[i]){

int temp = arr[i];

arr[i] = arr[i-1];

arr[i-1] = temp;

}

if(i < n-1 && arr[i + 1] > arr[i]){

int temp = arr[i];

arr[i] = arr[i+1];

arr[i+1] = temp;

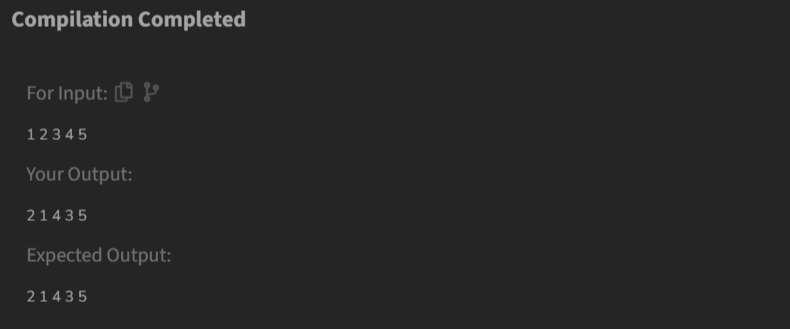
}

}

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**