DAY 5- GEEKSFORGEEKS

1.Stock buy and sell

Code:

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> pair = new ArrayList<>();

pair.add(buy);

pair.add(sell);

result.add(pair);

}

return result;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(n)

2.Coin changes(count ways)

Code:

class Solution {

public int count(int coins[], int sum) {

long[] dp=new long[sum+1];

dp[0]=1;

for(int i:coins){

for(int j=i;j<=sum;j++){

dp[j]+=dp[j-i];

}

}

return (int) dp[sum];

// code here.

}

}

A screenshot of a black screen

Description automatically generated

Time complexity: O(n)

3. First and Last occurrences

Code:

import java.util.ArrayList;

class Solution {

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

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

int first = binarySearch(arr, x, true);

if (first == -1) {

result.add(-1);

result.add(-1);

return result;

}

int last = binarySearch(arr, x, false);

result.add(first);

result.add(last);

return result;

}

private int binarySearch(int arr[], int x, boolean findFirst) {

int low = 0, high = arr.length - 1;

int result = -1;

while (low <= high) {

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

if (arr[mid] == x) {

result = mid;

if (findFirst) {

high = mid - 1;

}

else {

low = mid + 1;

}

}

else if (arr[mid] < x) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return result;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(log N)

4.First Repeating Element

Code:

class Solution {

public static int firstRepeated(int[] arr) {

Map<Integer,Integer> map=new HashMap<>();

for(int i=0;i<arr.length;i++){

if(map.containsKey(arr[i])){

return map.get(arr[i])+1;

}

else{

map.put(arr[i],i);

}

}

return -1;

// Your code here

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(n)

5. Remove duplicates from sorted Array

class Solution {

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

if(arr.size()<=1){

return arr.size();

}

int index=0;

for(int i=0;i<arr.size();i++){

if(!arr.get(i).equals(arr.get(index))){

index++;

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

}

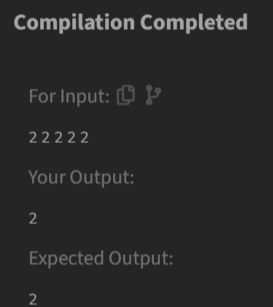
}

return index+1;

// Code Here

}

}



Time complexity: O(n)

6.Maximum Index:

class Solution {

int maxIndexDiff(int[] arr) {

int n = arr.length;

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;

int maxDiff = -1;

while (i < n && j < n) {

if (leftMin[i] < rightMax[j]) {

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

j++;

} else {

i++;

}

}

return maxDiff;

}

}

A screenshot of a black screen

Description automatically generated

Time complexity: O(n)

7.Transition Point

class Solution {

int transitionPoint(int arr[]) {

for(int i=0;i<arr.length;i++){

if(arr[i]==1){

return i;

}

}

return -1;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(n)