Q1) <https://www.hackerearth.com/problem/algorithm/range-sum-4/>

Q2) <https://www.codechef.com/problems/BLONDIE>

Q3)<https://leetcode.com/problems/shortest-unsorted-continuous-subarray/submissions/>

Q4) <https://leetcode.com/problems/subarray-sum-equals-k/>

Q5) <https://leetcode.com/problems/subarray-sums-divisible-by-k/>

Q6) https://practice.geeksforgeeks.org/problems/equilibrium-point-1587115620/1

Q7) https://practice.geeksforgeeks.org/problems/longest-span-with-same-sum-in-two-binary-arrays5142/1/?category[]=prefix-sum&category[]=prefix-sum&difficulty[]=1&page=1&query=category[]prefix-sumdifficulty[]1page1category[]prefix-sum

Blondie is a bounty hunter. Today, he is on a mission to shoot criminals in a particular city. A city is divided into NN areas, numbered 11 through NN. It is known that an area numbered ii contains AiAi criminals. But the problem here is that for some areas, the number of criminals is unknown. Blondie needs to know the number of criminals in each area in advance, as he is very good with the gun, so he only needs a single bullet for one criminal, that’s why he needs to know the count. He doesn’t like to carry more bullets with him.

The areas are arranged linearly in order. And the good part is that for each area with unknown criminals, the number of criminals can be calculated by following these simple steps:

*Step 1*: Count the criminals shot before  
*Step 2*: Count the areas visited before  
*Step 3*: Divide criminals found in *Step 1* by areas found in *Step 2*  
*Step 4*: Round it down to the nearest integer  
*Step 5*: That’s it

Calculate the number of bullets Blondie needs for each area.

2. #include <iostream>

using namespace std;

int main() {

long long int n,t;

cin>>t;

while(t--){

long long int sum=0;

cin>>n;

long long int a[n];

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

cin>>a[i];

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

if(a[i]==-1){

a[i]=sum/(i);

}

cout<<a[i]<<" ";

sum+=a[i];

}

cout<<endl;

}

}

. Given an integer array nums, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order.

Return the shortest such subarray and output its length.

**Example 1:**

**Input:** nums = [2,6,4,8,10,9,15]

**Output:** 5

**Explanation:** You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.

**Example 2:**

**Input:** nums = [1,2,3,4]

**Output:** 0

3. class Solution {

public:

int findUnsortedSubarray(vector<int>& N) {

int len = N.size() - 1, left = -1, right = -1,

maxN = N[0], minN = N[len];

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

int a = N[i], b = N[len-i];

if (a < maxN) right = i;

else maxN = a;

if (b > minN) left = i;

else minN = b;

}

return max(0, left + right - len + 1);

}

};

4.

Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals to *k*.

**Example 1:**

**Input:** nums = [1,1,1], k = 2

**Output:** 2

**Example 2:**

**Input:** nums = [1,2,3], k = 3

**Output:** 2

class Solution {

public:

int subarraySum(vector<int>& nums, int k) {

int count=0;

unordered\_map<int,int> hash;

int sum=0;

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

{

sum+=nums[i];

if(sum==k)

count++;

if(hash.find(sum-k)!= hash.end()){

count+=hash[sum-k];

}

hash[sum]++;

}

return count;

}

};

5. Given an array A of integers, return the number of (contiguous, non-empty) subarrays that have a sum divisible by K.

**Example 1:**

**Input:** A = [4,5,0,-2,-3,1], K = 5

**Output:** 7

**Explanation:** There are 7 subarrays with a sum divisible by K = 5:

[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]

class Solution {

public:

int subarraysDivByK(vector<int>& A, int K) {

unordered\_map<int,int>mp;

mp[0]=1;

int sum=0;

int ans=0;

int rem=0;

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

sum+=A[i];

rem = sum%K;

if(rem<0) rem+=K;

if(mp.find(rem)!=mp.end()){

ans+=mp[rem];

mp[rem]++;

}

else{

mp[rem]++;

}

}

return ans;

}

};

6. Given an array A of n positive numbers. The task is to find the first Equilibrium Point in the array.   
Equilibrium Point in an array is a position such that the sum of elements before it is equal to the sum of elements after it.

**Example 1:**

**Input:**

n = 1

A[] = {1}

**Output:** 1

**Explanation:** Since its the only

element hence its the only equilibrium

point.

* Initialize left\_sum = 0
* Find the sum of the array as **sum.**
* For i = 1 to end of the array, do the following:
* Update sum to get the right sum.
* sum = sum - arr[i] // sum is the right sum.
* If left\_sum == sum, return current index.
* Update left\_sum = left\_sum + arr[i]
* Return -1 and exit the algorithm. // Equilibrium index is not found.

#include <stdio.h>

int equilibrium\_index(int arr[], int n)

{

int sum = 0;

int leftsum = 0;

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

sum += arr[i];

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

sum -= arr[i];

if (leftsum == sum)

return i;

leftsum += arr[i];

}

return -1;

}

int main()

{

int n;

printf(“\nEnter the number of elements : “);

scanf(“%d”,&n);

int arr[n];

printf(“\nInput the array elements : “);

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

{

scanf(“%d”,&arr[i]);

}

printf(“\nEquilibrium Index : %d\n”, equilibrium\_index(arr, n));

return 0;

}}

7. Longest Span with same Sum in two Binary arrays

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 22 Apr, 2021

Given two binary arrays arr1[] and arr2[] of same size n. Find length of the longest common span (i, j) where j >= i such that arr1[i] + arr1[i+1] + …. + arr1[j] = arr2[i] + arr2[i+1] + …. + arr2[j].  
Expected time complexity is Θ(n).  
**Examples :**

#include<bits/stdc++.h>

using namespace std;

// Returns length of the longest common subarray

// with same sum

int longestCommonSum(bool arr1[], bool arr2[], int n)

{

    // Initialize result

    int maxLen = 0;

    // One by one pick all possible starting points

    // of subarrays

    for (int i=0; i<n; i++)

    {

       // Initialize sums of current subarrays

       int sum1 = 0, sum2 = 0;

       // Conider all points for starting with arr[i]

       for (int j=i; j<n; j++)

       {

           // Update sums

           sum1 += arr1[j];

           sum2 += arr2[j];

           // If sums are same and current length is

           // more than maxLen, update maxLen

           if (sum1 == sum2)

           {

             int len = j-i+1;

             if (len > maxLen)

                maxLen = len;

           }

       }

    }

    return maxLen;

}

// Driver program to test above function

int main()

{

    bool  arr1[] = {0, 1, 0, 1, 1, 1, 1};

    bool  arr2[] = {1, 1, 1, 1, 1, 0, 1};

    int n = sizeof(arr1)/sizeof(arr1[0]);

    cout << "Length of the longest common span with same "

            "sum is "<< longestCommonSum(arr1, arr2, n);

    return 0;

}