prefix sum sol

- Equilibrium Index Element : https://www.interviewbit.com/problems/balance-array/

```
int solve(int* A, int n1) {

    long long int* evenSum = (long long int *)malloc(sizeof(long long int) * n1);
    long long int* oddSum = (long long int *)malloc(sizeof(long long int) * n1);

    int i;

    evenSum[0] = A[0];
    oddSum[0] = 0;

    /* Prefix arrays for odd and even index */
    for(i=1; i < n1; i++) {

        if(i%2==0) {
            evenSum[i] = evenSum[i-1] + A[i];
            oddSum[i] = oddSum[i-1];
        } else {
            evenSum[i] = evenSum[i-1];
            oddSum[i] = oddSum[i-1] + A[i];
        }

    }

    int ans = 0;

    for(i=0;i<n1;i++) {

        /* Store even and odd sum before the current index i */
        long long int prevEvenSum;
        long long int prevOddSum;
```

```
        if(i > 0) {
            prevEvenSum = evenSum[i-1];
            prevOddSum = oddSum[i-1];
        } else {
            prevEvenSum = 0;
            prevOddSum = 0;
        }


        /* calculate the even and odd sum by subtracting the sum till i index from total
sum */
        /* note - because the current element is removed, even sum will become odd sum
and odd sum will become even sum for next elements */
        long long int nextEvenSum = oddSum[n1-1] - oddSum[i];
        long long int nextOddSum = evenSum[n1-1] - evenSum[i];

        if(prevEvenSum + nextEvenSum == prevOddSum + nextOddSum) {
            ans++;
        }

    }

    return ans;
}
int solve(int* A, int n1) {

    long long int* evenSum = (long long int *)malloc(sizeof(long long int) * n1);
    long long int* oddSum = (long long int *)malloc(sizeof(long long int) * n1);

    int i;

    evenSum[0] = A[0];
    oddSum[0] = 0;

    /* Prefix arrays for odd and even index */
    for(i=1; i < n1; i++) {
```

```
        if(i%2==0) {
            evenSum[i] = evenSum[i-1] + A[i];
            oddSum[i] = oddSum[i-1];
        } else {
            evenSum[i] = evenSum[i-1];
            oddSum[i] = oddSum[i-1] + A[i];
        }

    }

    int ans = 0;

    for(i=0;i<n1;i++) {

        /* Store even and odd sum before the current index i */
        long long int prevEvenSum;
        long long int prevOddSum;

        if(i > 0) {
            prevEvenSum = evenSum[i-1];
            prevOddSum = oddSum[i-1];
        } else {
            prevEvenSum = 0;
            prevOddSum = 0;
        }


        /* calculate the even and odd sum by subtracting the sum till i index from total
sum */
        /* note - because the current element is removed, even sum will become odd sum
and odd sum will become even sum for next elements */
        long long int nextEvenSum = oddSum[n1-1] - oddSum[i];
        long long int nextOddSum = evenSum[n1-1] - evenSum[i];

        if(prevEvenSum + nextEvenSum == prevOddSum + nextOddSum) {
```

```
            ans++;
        }

    }

    return ans;
}
```

- . Pick from both sides!: https://www.interviewbit.com/problems/pick-from-both-sides/

```
public class Solution {
    public int solve(ArrayList<Integer> A, int B) {
        int size = A.size();
        int maxSum = 0;
        for(int i=0; i<B; i++) {
            maxSum += A.get(i);
        }
        if(B == size) {
            return maxSum;
        }

        int localSum = maxSum;
        int startPointer = B - 1;
        int endPointer = size - 1;

        for(int i=startPointer; i>=0; i--) {
            localSum -= A.get(i);
            localSum += A.get(endPointer);

            maxSum = Math.max(localSum, maxSum);
            endPointer--;
        }


        return maxSum;
    }
```

```
}
```

- https://leetcode.com/problems/minimum-operations-to-make-array-equal/

```
class Solution {
    public int minOperations(int n) {
        int ans=0;
        for(int i=0;i<n/2;i++){
            int x=(2 * i) + 1;
            ans+=n-x;
        }
        return ans;
    }
}
```

- 303. Range Sum Query - Immutable : https://leetcode.com/problems/range-sum-query-immutable/'

```
class NumArray {
    private int[] sumArray;
    public NumArray(int[] nums) {
        sumArray = new int[nums.length + 1];
        for (int i = 0; i < nums.length; i++) {
            sumArray[i + 1] = sumArray[i] + nums[i];
```

```
        }

    }



    public int sumRange(int left, int right) {

        return sumArray[right + 1] - sumArray[left];

    }

}
```

- **Equilibrium Point :** **https://practice.geeksforgeeks.org/problems/equilibrium-point-1587115620/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article**

```
//{ Driver Code Starts
import java.io.*;
import java.util.*;
import java.util.stream.*;

class Main {

    public static void main(String[] args) throws IOException {
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        int t =
            Integer.parseInt(br.readLine().trim()); // Inputting the testcases
        while (t-- > 0) {

            //taking input n
            int n = Integer.parseInt(br.readLine().trim());
```

```
            long arr[] = new long[n];
            String inputLine[] = br.readLine().trim().split(" ");

            //adding elements to the array
            for (int i = 0; i < n; i++) {
                arr[i] = Long.parseLong(inputLine[i]);
            }

            Solution ob = new Solution();

            //calling equilibriumPoint() function
            System.out.println(ob.equilibriumPoint(arr, n));
        }
    }
}
// } Driver Code Ends


// class Solution {


//     // a: input array
//     // n: size of array
//     // Function to find equilibrium point in the array.
//     public static int equilibriumPoint(long arr[], int n) {

//         // Your code here
//     }
// }
int equilibriumPoint(long long arr[], int n) {
```

- Product of Array Except Self : https://leetcode.com/problems/product-of-array-except-self/description/

```
        long long s1,s2;
        s1=s2=0;
        int i,j;
        i=-1;
```

```
        j=n;
        while(i<=j){
            if(s1==s2){
                if(j==(i+2)){
                    return j;
                }
                s1+=arr[++i];
                s2+=arr[--j];
            }else if(s1<s2){
                s1+=arr[++i];
            }else{
                s2+=arr[--j];
            }
        }
        return -1;
    }

class Solution {

    public int[] productExceptSelf(int[] nums) {

        int n = nums.length;

        int ans[] = new int[n];



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

            int pro = 1;

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

                if(i == j) continue;

                pro *= nums[j];
```

```
            }

            ans[i] = pro;

        }



        return ans;

    }

}
```