

# PREFIX SUM ASSIGNMENT

## Q1. Running Sum of 1d Array

**1480. Running Sum of 1d Array**

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]...nums[i])`.

Return the running sum of `nums`.

**Example 1:**

Input: `nums = [1,2,3,4]`  
Output: `[1,3,6,10]`  
Explanation: Running sum is obtained as follows: `[1, 1+2, 1+2+3, 1+2+3+4]`.

**Example 2:**

Input: `nums = [1,1,1,1,1]`  
Output: `[1,2,3,4,5]`  
Explanation: Running sum is obtained as follows: `[1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]`.

**Example 3:**

Input: `nums = [3,1,2,10,1]`  
Output: `[3,4,6,16,17]`

```
class Solution {
    public int[] runningSum(int[] nums) {
        int sum = 0;
        for(int i = 0; i < nums.length; i++){
            sum += nums[i];
            nums[i] = sum;
        }
        return nums;
    }
}
```

Testcase Result: Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

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

Output: `[1,3,6,10]`

Console Run Submit

## Q2. Sum of All Odd Length Subarrays

**1588. Sum of All Odd Length Subarrays**

Given an array of positive integers `arr`, return the sum of all possible **odd-length subarrays** of `arr`.

A **subarray** is a contiguous subsequence of the array.

**Example 1:**

Input: `arr = [1,4,2,5,3]`  
Output: 58  
Explanation: The odd-length subarrays of `arr` and their sums are:  
`[1] = 1`  
`[4] = 4`  
`[2] = 2`  
`[5] = 5`  
`[3] = 3`  
`[1,4,2] = 7`  
`[4,2,5] = 11`  
`[2,5,3] = 10`  
`[1,4,2,5,3] = 15`  
If we add all these together we get `1 + 4 + 2 + 5 + 3 + 7 + 11 + 10 + 15 = 58`

**Example 2:**

```
class Solution {
    public int sumOddLengthSubarrays(int[] arr) {
        int sum = 0, n = arr.length;
        for (int i = 0; i < n; i++){
            int count = ((i+1)*(n-i)+1)/2;
            sum += count * arr[i];
        }
        return sum;
    }
}
```

Testcase Result: Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: `arr = [1,4,2,5,3]`

Output: 58

Console Run Submit

### Q3. Find the Highest Altitude

The screenshot shows the LeetCode website for problem 1732, "Find the Highest Altitude". The problem is categorized as "Easy" with 2.4K likes and 215 comments. The description states: "There is a biker going on a road trip. The road trip consists of  $n + 1$  points at different altitudes. The biker starts his trip on point 0 with altitude equal 0. You are given an integer array `gain` of length  $n$  where `gain[i]` is the net gain in altitude between points  $i$  and  $i + 1$  for all  $(0 \leq i < n)$ . Return the highest altitude of a point."

Example 1:  
Input: `gain = [-5,1,5,0,-7]`  
Output: 1  
Explanation: The altitudes are `[0,-5,-4,1,1,-6]`. The highest is 1.

Example 2:  
Input: `gain = [-4,-3,-2,-1,4,3,2]`  
Output: 0  
Explanation: The altitudes are `[0,-4,-7,-9,-10,-6,-3,-1]`. The highest is 0.

Constraints:

- $1 \leq n \leq 10^5$
- $-10^4 \leq \text{gain}[i] \leq 10^4$

The solution is implemented in C++:

```
class Solution {
public:
    int largestAltitude(vector<int>& gain) {
        vector<int> alt(gain.size()+1);
        alt[0] = 0;
        for(int i = 0; i < gain.size(); i++)
            alt[i+1] = alt[i] + gain[i];
        return *max_element(alt.begin(), alt.end());
    }
};
```

The test case shows the input `gain = [-5,1,5,0,-7]` and the output `1`. The status is "Accepted" with a runtime of 0 ms.

### Q4. Find the Middle Index in Array

The screenshot shows the LeetCode website for problem 1991, "Find the Middle Index in Array". The problem is categorized as "Easy" with 1.2K likes and 49 comments. The description states: "Given a 0-indexed integer array `nums`, find the leftmost `middleIndex` (i.e., the smallest amongst all the possible ones). A `middleIndex` is an index where `nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1]`. If `middleIndex == 0`, the left side sum is considered to be 0. Similarly, if `middleIndex == nums.length - 1`, the right side sum is considered to be 0. Return the leftmost `middleIndex` that satisfies the condition, or -1 if there is no such index."

Example 1:  
Input: `nums = [2,3,-1,8,4]`  
Output: 3  
Explanation: The sum of the numbers before index 3 is: `2 + 3 + -1 = 4`. The sum of the numbers after index 3 is: `4 = 4`.

Example 2:  
Input: `nums = [1,-1,4]`  
Output: 2

The solution is implemented in Java:

```
class Solution {
    public int findMiddleIndex(int[] nums) {
        int leftSum = 0, rightSum = 0;
        for(int n : nums) rightSum += n;
        for(int i = 0; i < nums.length; i++) {
            rightSum -= nums[i];
            if(leftSum == rightSum) return i;
            leftSum += nums[i];
        }
        return -1;
    }
}
```

The test case shows the input `nums = [2,3,-1,8,4]` and the output `3`. The status is "Accepted" with a runtime of 0 ms.

## Q5. Find the Pivot Integer

**2485. Find the Pivot Integer**

Given a positive integer  $n$ , find the **pivot integer**  $x$  such that:

- The sum of all elements between  $1$  and  $x$  inclusively equals the sum of all elements between  $x$  and  $n$  inclusively.

Return the **pivot integer**  $x$ . If no such integer exists, return  $-1$ . It is guaranteed that there will be at most one pivot index for the given input.

**Example 1:**

Input:  $n = 8$   
Output:  $6$   
Explanation:  $6$  is the pivot integer since:  $1 + 2 + 3 + 4 + 5 + 6 = 6 + 7 + 8 = 21$ .

**Example 2:**

Input:  $n = 1$   
Output:  $1$   
Explanation:  $1$  is the pivot integer since:  $1 = 1$ .

```
1 import java.lang.*;
2 class Solution {
3     public int pivotInteger(int n) {
4         int a = (int)Math.sqrt((n*n + n)/2);
5         if(2*a*a==n*n+n)
6             return a;
7         return -1;
8     }
9 }
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$n =$

8

Output

6

Console Run Submit

## Q6. Left and Right Sum Differences

**1934. Confirmation Rate**

Medium

SQL Schema

Table: Signups

Column Name	Type
user_id	int
time_stamp	datetime

user\_id is the primary key for this table. Each row contains information about the signup time for the user with ID user\_id.

Table: Confirmations

Column Name	Type
user_id	int
time_stamp	datetime
action	ENUM

```
1 # Write your MySQL query statement below
2 select s.user_id, CASE WHEN c.time_stamp is NULL then 0.00
3 else ROUND(sum(c.action='confirmed')/COUNT(*),2)
4 END AS confirmation_rate
5 from Signups s
6 left join Confirmations c
7 on s.user_id =c.user_id
8 group by s.user_id
```

Testcase Result

Accepted Runtime: 243 ms

Case 1

Input

Signups =

user_id	time_stamp
3	2020-03-21 10:16:13
7	2020-01-04 13:57:59
2	2020-07-20 22:00:44

Console Run Submit

## Q7. Sum of Absolute Differences in a Sorted Array

The screenshot shows the LeetCode problem page for "1685. Sum of Absolute Differences in a Sorted Array". The problem is rated "Medium" and has 1.1K likes and 32 comments. The description states: "You are given an integer array `nums` sorted in **non-decreasing** order. Build and return an integer array `result` with the same length as `nums` such that `result[i]` is equal to the **summation of absolute differences** between `nums[i]` and all the other elements in the array. In other words, `result[i]` is equal to  $\sum(|nums[i] - nums[j]|)$  where  $0 \leq j < nums.length$  and  $j \neq i$  (0-indexed)." Example 1: Input: `nums = [2,3,5]`, Output: `[4,3,5]`. Example 2: Input: `nums = [1,4,6,8,10]`, Output: `[24,15,13,15,21]`. The solution is written in Java using a prefix sum approach. The test case shows the input `[2,3,5]` and the output `[4,3,5]`, with the result marked as "Accepted".

1685. Sum of Absolute Differences in a Sorted Array

Medium 1.1K 32

Companies

You are given an integer array `nums` sorted in **non-decreasing** order.

Build and return an integer array `result` with the same length as `nums` such that `result[i]` is equal to the **summation of absolute differences** between `nums[i]` and all the other elements in the array.

In other words, `result[i]` is equal to  $\sum(|nums[i] - nums[j]|)$  where  $0 \leq j < nums.length$  and  $j \neq i$  (0-indexed).

Example 1:

Input: `nums = [2,3,5]`  
Output: `[4,3,5]`  
Explanation: Assuming the arrays are 0-indexed, then  
`result[0] = |2-2| + |2-3| + |2-5| = 0 + 1 + 3 = 4`,  
`result[1] = |3-2| + |3-3| + |3-5| = 1 + 0 + 2 = 3`,  
`result[2] = |5-2| + |5-3| + |5-5| = 3 + 2 + 0 = 5`.

Example 2:

Input: `nums = [1,4,6,8,10]`  
Output: `[24,15,13,15,21]`

```
1 class Solution {
2     public int[] getSumAbsoluteDifferences(int[] nums) {
3         int n = nums.length;
4         int[] res = new int[n];
5         int[] prefixSum = new int[n + 1];
6         for (int i = 0; i < n; ++i) {
7             prefixSum[i + 1] = prefixSum[i] + nums[i];
8         }
9         for (int i = 0; i < n; ++i) {
10            res[i] = i * nums[i] - prefixSum[i] + (prefixSum[n] - prefixSum[i] - (n - i) *
11                nums[i]);
12        }
13        return res;
14    }
15 }
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`nums = [2,3,5]`

Output

`[4,3,5]`

Console Run Submit

## Q8. Find the Score of All Prefixes of an Array

The screenshot shows the LeetCode problem page for "2640. Find the Score of All Prefixes of an Array". The problem is rated "Medium" and has 194 likes and 11 comments. The description defines a "conversion array" `conver` of an array `arr` as `conver[i] = arr[i] + max(arr[0..i])`, where `max(arr[0..i])` is the maximum value of `arr[j]` over  $0 \leq j \leq i$ . It also defines the "score" of an array `arr` as the sum of the values of the conversion array of `arr`. Given a 0-indexed integer array `nums` of length `n`, return an array `ans` of length `n` where `ans[i]` is the score of the prefix `nums[0..i]`. Example 1: Input: `nums = [2,3,7,5,10]`, Output: `[4,10,24,36,56]`. Explanation: For the prefix `[2]`, the conversion array is `[4]` hence the score is 4. For the prefix `[2, 3]`, the conversion array is `[4, 6]` hence the score is 10. For the prefix `[2, 3, 7]`, the conversion array is `[4, 6, 14]` hence the score is 24. For the prefix `[2, 3, 7, 5]`, the conversion array is `[4, 6, 14, 12]` hence the score is 36. For the prefix `[2, 3, 7, 5, 10]`, the conversion array is `[4, 6, 14, 12, 20]` hence the score is 56. The solution is written in Java using a single pass to calculate the maximum and the score. The test case shows the input `[2,3,7,5,10]` and the output `[4,10,24,36,56]`, with the result marked as "Accepted".

2640. Find the Score of All Prefixes of an Array

Medium 194 11

Companies

We define the **conversion array** `conver` of an array `arr` as follows:

- `conver[i] = arr[i] + max(arr[0..i])` where `max(arr[0..i])` is the maximum value of `arr[j]` over  $0 \leq j \leq i$ .

We also define the **score** of an array `arr` as the sum of the values of the conversion array of `arr`.

Given a 0-indexed integer array `nums` of length `n`, return an array `ans` of length `n` where `ans[i]` is the score of the prefix `nums[0..i]`.

Example 1:

Input: `nums = [2,3,7,5,10]`  
Output: `[4,10,24,36,56]`  
Explanation:  
For the prefix `[2]`, the conversion array is `[4]` hence the score is 4  
For the prefix `[2, 3]`, the conversion array is `[4, 6]` hence the score is 10  
For the prefix `[2, 3, 7]`, the conversion array is `[4, 6, 14]` hence the score is 24  
For the prefix `[2, 3, 7, 5]`, the conversion array is `[4, 6, 14, 12]` hence the score is 36  
For the prefix `[2, 3, 7, 5, 10]`, the conversion array is `[4, 6, 14, 12, 20]` hence the score is 56

```
1 class Solution {
2     public long[] findPrefixScore(int[] nums) {
3         int n = nums.length;
4         long[] score = new long[n + 1];
5         for (int i = 0; i < n; ++i) {
6             int max = Math.max(max, nums[i]);
7             score[i + 1] = score[i] + max + nums[i];
8         }
9         return Arrays.copyOfRange(score, 1, n + 1);
10    }
11 }
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`nums = [2,3,7,5,10]`

Output

`[4,10,24,36,56]`

Console Run Submit