Prefix Sum Array

Given an array arr[] of size N, the task is to generate the *prefix sum array* of the given array.

Prefix Sum Array: The prefix sum array of any array, arr[ ] is defined as an array of same size say, prefixSum[ ] such that the value at any index **i** in prefixSum[ ] is sum of all elements from indexes **0 to i** in arr[].

That is,

```
prefixSum[i] = arr[0] + arr[1] + arr[2] + . . . . + arr[i]

for all0 <= i <= N.
```

Examples:

```
Input  : arr[] = {10, 20, 10, 5, 15}
Output : prefixSum[] = {10, 30, 40, 45, 60}

Explanation : While traversing the array, update
the element by adding it with its previous element.
prefixSum[0] = 10,
prefixSum[1] = prefixSum[0] + arr[1] = 30,
prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.
```

Below function generates a prefix sum array for a given array arr[] of size N:

```
void fillPrefixSum(int arr[], int N, int prefixSum[])
{
    prefixSum[0] = arr[0];

    // Adding present element
    // with previous element
    for (int i = 1; i < N; i++)
        prefixSum[i] = prefixSum[i-1] + arr[i];
}
```

Finding sum in a Range: We can easily calculate the sum with-in a range [i, j] in an array using the prefix sum array. Since the array prefixSum[i] stores the sum of all elements upto **i**. Therefore, **prefixSum[j]** - **prefixSum[i]** will give:

```
sum of elements upto j-th index - sum of elements upto i-th
element
```

The above total sum will exclude the i-th element.

Therefore, we can get the sum of elements in range [i,j] by:

```
prefixSum[j] - prefixSum[i-1]
```

The above formula will not work in the case when **i** = **0**.

Therefore,

```
sumInRange = prefixSum[j]  , if i = 0

otherwise,

sumInRange = prefixSum[j] - prefixSum[i-1]  ,  if (i != 0).
```

Sample Problem: Consider an array of size N with all initial values as 0. Perform given 'm' add operations from index 'a' to 'b' and evaluate highest element in array. An add operation adds 100 to all elements from index a to b (both inclusive).

Example:

```
Input : n = 5 // We consider array {0, 0, 0, 0, 0}
        m = 3.
        a = 2, b = 4.
        a = 1, b = 3.
        a = 1, b = 2.
Output : 300

Explanation :
After I operation -
A : 0 100 100 100 0
```

```
After II operation -
A : 100 200 200 100 0

After III operation -
A : 200 300 200 100 0

Highest element : 300
```

Solution using Prefix Sum:

```
1 : Run a loop for 'm' times, inputting 'a' and 'b'.
2 : Add 100 at index 'a' and subtract 100 from index 'b+1'.
3 : After completion of 'm' operations, compute the prefix
sum array.
4 : Scan the largest element and we're done.
```

What we did was adding 100 at ‘a’ because this will add 100 to all elements while taking prefix sum array. Subtracting 100 from ‘b+1’ will reverse the changes made by adding 100 to elements from ‘b’ onward.

For better understanding :

```
After I operation -
A : 0 100 0 0 -100
Prefix Sum Array : 0 100 100 100 0

After II operation -
A : 100 100 0 -100 -100
Prefix Sum Array : 100 200 200 100 0

After III operation -
A : 200 100 -100 -100 -100
Prefix Sum Array : 200 300 200 100 0

Final Prefix Sum Array : 200 300 200 100 0

The required highest element : 300
```