

## Range Addition

Assume you have an array of length  $n$  initialized with all 0's and are given  $k$  update operations.

Each operation is represented as a triplet: **[startIndex, endIndex, inc]** which increments each element of subarray **A[startIndex ... endIndex]** (startIndex and endIndex inclusive) with **inc**.

Return the modified array after all  $k$  operations were executed.

### Example:

Given:

```
length = 5,  
updates = [  
    [1, 3, 2],  
    [2, 4, 3],  
    [0, 2, -2]  
]
```

Output:

```
[-2, 0, 3, 5, 3]
```

### Explanation:

Initial state:  
[ 0, 0, 0, 0, 0 ]

After applying operation [1, 3, 2]:  
[ 0, 2, 2, 2, 0 ]

After applying operation [2, 4, 3]:  
[ 0, 2, 5, 5, 3 ]

After applying operation [0, 2, -2]:  
[-2, 0, 3, 5, 3 ]

1. Thinking of using advanced data structures? You are thinking it too complicated.
2. For each update operation, do you really need to update all elements between  $i$  and  $j$ ?
3. Update only the first and end element is sufficient.
4. The optimal time complexity is  $O(k + n)$  and uses  $O(1)$  extra space.

### Credits:

Special thanks to [@vinod23](#) for adding this problem and creating all test cases.

## Solution 1

Just store every start index for each value and at end index plus one minus it  
for example it will look like:

[1 , 3 , 2] , [2, 3, 3] (length = 5)

res[ 0, 2, ,0, 0 -2 ]

res[ 0 ,2, 3, 0, -5]

sum 0, 2, 5, 5, 0

res[0, 2, 5, 5, 0]

```
public int[] getModifiedArray(int length, int[][] updates) {  
  
    int[] res = new int[length];  
    for(int[] update : updates) {  
        int value = update[2];  
        int start = update[0];  
        int end = update[1];  
  
        res[start] += value;  
  
        if(end < length - 1)  
            res[end + 1] -= value;  
    }  
  
    int sum = 0;  
    for(int i = 0; i < length; i++) {  
        sum += res[i];  
        res[i] = sum;  
    }  
  
    return res;  
}
```

written by [lidoo4](#) original link [here](#)

## Solution 2

From the hint, we only need to update first and end element, so we update the startIndex with inc, then update endIndex + 1 with -inc.

Using the example in the problem, We get vector nums = {-2, 2, 3, 2, -2, -3}, then we compute range sum ( [Range Sum Query - Immutable](#)), that is the final result = {-2, 0, 3, 5, 3}.

```
class Solution {
public:
    vector<int> getModifiedArray(int length, vector<vector<int>>& updates) {
        vector<int> res, nums(length + 1, 0);
        for (int i = 0; i < updates.size(); ++i) {
            nums[updates[i][0]] += updates[i][2];
            nums[updates[i][1] + 1] -= updates[i][2];
        }
        int sum = 0;
        for (int i = 0; i < length; ++i) {
            sum += nums[i];
            res.push_back(sum);
        }
        return res;
    }
};
```

written by [grandyang](#) original link [here](#)

### Solution 3

Just use the first position and the **last + 1** position. The last + 1 position add the opposite value. So when you add the numbers from start to end you will have correct result.

written by [willcomeback](#) original link [here](#)

From [Leetcode](#).