Pupils Plan \$25

GUC COLLEGIATE
PROGRAMMING CONTEST

Prefix Sums, Frequency Arrays & Max Subarray Sum





Problem: Range Sum Query (Immutable)

You're given an array of integers:

```
java

int[] nums = {-2, 0, 3, -5, 2, -1};
```

You will be asked multiple queries, each giving you two indices left and right.

For each query, return the sum of the elements from index left to right, inclusive.



Query 1:

```
java

left = 0, right = 2

Answer = nums[0] + nums[1] + nums[2] = -2 + 0 + 3 = 1
```

Query 2:

```
java

left = 2, right = 5

Answer = nums[2] + nums[3] + nums[4] + nums[5] = 3 + (-5) + 2 + (-1) = -1
```

Basic Approach

For each query, you loop over the range from left to right and compute the sum:

```
java

int sum = 0;
for (int i = left; i <= right; i++) {
    sum += nums[i];
}</pre>
```

- Correct, but inefficient if the number of queries is large.
- Time Complexity per query: O(R L + 1)

Optimized with Prefix Sum

Instead of recalculating each range, we preprocess the array:

Step 1: Build prefix sum array

```
java

int[] prefix = new int[nums.length];
prefix[0] = nums[0];
for (int i = 1; i < nums.length; i++) {
    prefix[i] = prefix[i - 1] + nums[i];
}</pre>
```

Step 2: Answer any query in O(1)

```
java

int sum = (left == 0) ? prefix[right] : prefix[right] - prefix[left - 1];
```





Problem Setup

You are given a 2D integer array logs where each logs[i] = [birthi, deathi] indicates the birth and death years of the ith person. The population of...

Can you solve this real interview question? Maximum Population Year -

Maximum Population Year

Suppose you're given an array of size n, initially filled with zeros:

```
java

□ Copy ₺ Edit

arr = [0, 0, 0, 0, 0]
```

You receive multiple update queries of the form:

add value v to all elements from index L to R (inclusive)

```
Update 1: add 3 from index 1 to 3 → arr = [0, 3, 3, 3, 0]

Update 2: add 2 from index 2 to 4 → arr = [0, 3, 5, 5, 2]
```

Direct Approach (Inefficient)

For each query, loop through the entire range and add the value:

```
java

for (int i = left; i <= right; i++) {
    arr[i] += value;
}</pre>
```

♦ Optimized Approach – Sweep Line / Difference Array

Instead of updating every index in a range, we:

- Add +v at index L
- Subtract -v at index R + 1
- Then take a prefix sum to apply all updates in one pass

Step 1: Initialize a difference array: ∜ Edit java int[] diff = new int[n + 1]; // one extra space for boundary handling **Step 2:** For each update: ₽ Edit java diff[left] += value; if (right + 1 < n) diff[right + 1] -= value;</pre> Step 3: Apply prefix sum on the difference array to get final result: ℃ Edit java int[] result = new int[n]; result[0] = diff[0]; for (int i = 1; i < n; i++) {</pre> result[i] = result[i - 1] + diff[i];

```
Update 1: add 3 from 1 to 3 → diff = [0, 3, 0, 0, -3, 0]
Update 2: add 2 from 2 to 4 → diff = [0, 3, 2, 0, -1, -3]
Prefix sum of diff = [0, 3, 5, 5, 2]
```

What is a Frequency Array?

A frequency array is a technique used to count how many times each value appears in a dataset. It's one of the fastest ways to answer frequency-related queries.

★ When to Use

- When your input values are bounded and small (e.g., values from 0 to 100)
- When you need to answer questions like:
 - How many times did x appear?
 - What is the most frequent number?

Using an Array (Best When Range is Small & Known)

If values are between 0 and 100:

```
java

int[] freq = new int[101]; // size = max possible value + 1

for (int x : nums) {
    freq[x]++;
}
```

Using a HashMap (Best When Values are Large or Negative)

If values are large (like up to 10°) or include negatives:

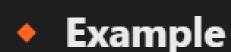
```
java

Map<Integer, Integer> freq = new HashMap<>();
for (int x : nums) {
    freq.put(x, freq.getOrDefault(x, 0) + 1);
}
```

Maximum Subarray Sum – Finding the Best Range

Problem Statement

Given an array of integers (can include negatives), find the maximum possible sum of a contiguous subarray.



java

int[] nums = $\{-2, 1, -3, 4, -1, 2, 1, -5, 4\}$;

Expected Output: 6

Subarray that gives the max sum: [4, -1, 2, 1]



Maximum Subarray

Can you solve this real interview question? Maximum Subarray - Given an integer array nums, find the subarray with the largest sum, and return its sum. Example 1: Input: nums = [-2,1,-3,4,-1,2,1,-5,4] Output: 6...

▲ LeetCode

Brute Force Idea (O(n²))

Try all possible subarrays and track the one with the max sum.

```
java
                                                                                   ゅ Edit
int maxSum = Integer.MIN_VALUE;
for (int i = 0; i < n; i++) {
   int sum = 0;
   for (int j = i; j < n; j++) {
       sum += nums[j];
        maxSum = Math.max(maxSum, sum);
```

♦ Kadane's Algorithm – O(n) Optimal Solution

We keep track of:

- currentSum: maximum subarray sum ending at current index
- maxSum: global maximum subarray sum

```
java

int currentSum = nums[0];
int maxSum = nums[0];

for (int i = 1; i < nums.length; i++) {
    currentSum = Math.max(nums[i], currentSum + nums[i]);
    maxSum = Math.max(maxSum, currentSum);
}</pre>
```

✓ Time Complexity: O(n)