

Pupils Plan S25



Prefix Sums, Frequency Arrays & Max Subarray Sum



1 Prefix Sum – Range Query Summation

■ Problem: Range Sum Query (Immutable)

You're given an array of integers:

java

Copy

Edit

```
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.



LeetCode

Range Sum Query - Immutable


Can you solve this real interview question? Range Sum Query - Immutable - Given an integer array `nums`, handle multiple queries of the following type: 1. Calculate the sum of the elements of `nums` between...

LeetCode

Query 1:

java

 Copy

 Edit

```
left = 0, right = 2
```

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

Query 2:

java

 Copy

 Edit

```
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

Copy

Edit

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

✓ 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

Copy

Edit

```
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];
}
```

Step 2: Answer any query in O(1)

java

Copy

Edit

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

2 Sweep Line – Range Updates Optimization



LeetCode

Maximum Population Year

Can you solve this real interview question? Maximum Population Year - You are given a 2D integer array logs where each logs[i] = [birth_i, death_i] indicates the birth and death years of the ith person. The population of...

LeetCode

Problem Setup

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

 Copy

 Edit

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

⚡ 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:

java

 Copy


 Edit

```
int[] diff = new int[n + 1]; // one extra space for boundary handling
```

Step 2: For each update:

java

 Copy


 Edit

```
diff[left] += value;  
if (right + 1 < n) diff[right + 1] -= value;
```

Step 3: Apply prefix sum on the difference array to get final result:

java

 Copy

 Edit

```
int[] result = new int[n];  
result[0] = diff[0];  
for (int i = 1; i < n; i++) {  
    result[i] = result[i - 1] + diff[i];  
}
```

Update 1: add 3 from 1 to 3 \rightarrow diff = [0, 3, 0, 0, -3, 0]

Update 2: add 2 from 2 to 4 \rightarrow 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

Copy

Edit

```
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 10^9) or include negatives:

java

Copy

Edit

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

4 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.

Example

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]



LeetCode

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 ($O(n^2)$)

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

java

 Copy

 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

Copy

Edit

```
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);
}
```

✓ Time Complexity: $O(n)$