Module 2 of 12

Prefix sums and difference arrays

Answer any range sum in O(1) after one O(n) pass, and apply many range updates with two writes each — the two halves of one idea.

Summing a range element by element is O(n) per query; with 10^5 queries over 10^5 numbers that is 10^{10} steps. A prefix array, built once in O(n), answers every range sum with one subtraction.

Run the idea backwards and you get the difference array: to add v to a whole range, write +v where it starts and −v just after it ends, then one prefix sum at the end applies every update at once. Together they turn "many queries" and "many updates" problems from O(n · q) into O(n + q).

The exercises are both halves at contest sizes — range-sum queries with sums past 2^{31}, and thousands of range additions — and the checkpoint finds the busiest hour of a booking timeline a million hours long.

CONTENTS

1	Prefix sums and difference arrays	15 min
✓	Checkpoint — Prefix sums and difference arrays	45 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 15 MIN

Prefix sums and difference arrays

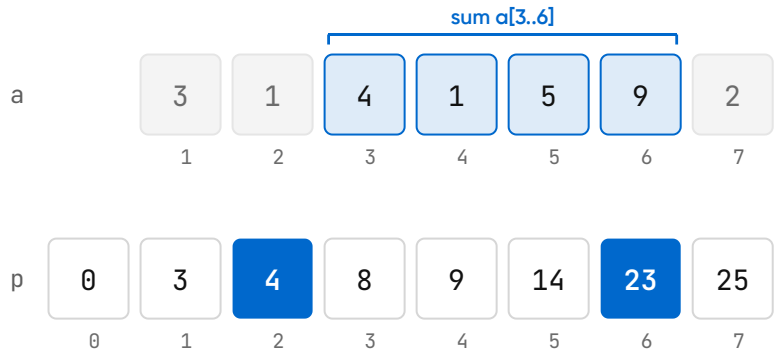
Two problems, one idea. **Many range-sum queries** on a fixed array, and **many range updates** followed by one read. Done element by element both are O(n · q) — 10^{10} at the limits.

Prefix sums: precompute, then subtract

```
vector<ll> p(n + 1, 0);                // p[0] = 0
for (int i = 1; i ⩽ n; ++i) p[i] = p[i - 1] + a[i];
// sum of a[l..r]
ll s = p[r] - p[l - 1];
```

- p[i] is the sum of the first i elements. Building it is one pass, O(n).
- A range is "everything up to r" minus "everything before l".
- `p[0] = 0` means l = 1 needs no special case.
- `long long`: a prefix of $2 \cdot 10^5$ values of 10^9 is $2 \cdot 10^{14}$.

Build the prefix array once, answer every range in O(1)



1. p[i] will hold a[1] + … + a[i]. Start with p[0] = 0, the sum of nothing — it is what lets a range starting at 1 need no special case.
2. p[1] = p[0] + a[1] = 0 + 3 = 3. One addition per slot: the previous total plus the new element.
3. p[2] = p[1] + a[2] = 3 + 1 = 4. One addition per slot: the previous total plus the new element.
4. p[3] = p[2] + a[3] = 4 + 4 = 8. One addition per slot: the previous total plus the new element.
5. The loop runs to the end the same way, so the whole prefix array costs O(n): p = [0, 3, 4, 8, 9, 14, 23, 25].
6. Now any range is two lookups. a[3..6] is everything up to 6 minus everything up to 2: p[6] − p[2] = 23 − 4 = 19, which is 4 + 1 + 5 + 9. O(1) per query.

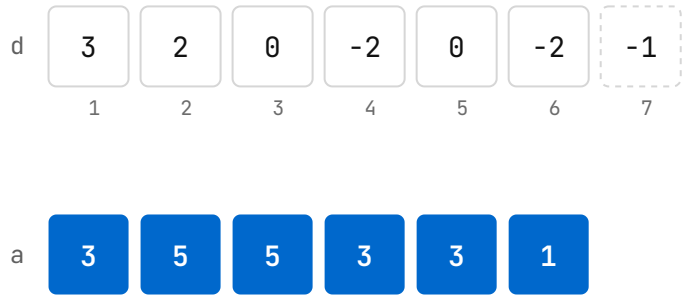
Difference arrays: update now, sum later

Run the idea backwards. To add v to a[l..r], mark only where the change **starts** and where it **stops**:

```
vector<ll> d(n + 2, 0);
d[l] += v;                              // the change starts at l
d[r + 1] -= v;                          // and stops after r
// after every update:
ll run = 0;
for (int i = 1; i ⩽ n; ++i) { run += d[i]; a[i] = run; }
```

- Two writes per update, however long the range.
- One running sum rebuilds the whole array: O(n + m) in total.
- `d` has n + 2 slots so an update ending at n still has somewhere to stop.

Many range updates, then one prefix sum: the difference array



1. Add v to every element of [l, r] by writing d[l] += v and d[r + 1] −= v. d has a slot n + 1 = 7, so an update that ends at n still has somewhere to stop.
2. Update +3 on 1..3: d[1] += 3 and d[4] −= 3. Two writes whatever the length of the range — d is now [3, 0, 0, −3, 0, 0, 0].
3. Update +2 on 2..5: d[2] += 2 and d[6] −= 2. Two writes whatever the length of the range — d is now [3, 2, 0, −3, 0, −2, 0].
4. Update +1 on 4..6: d[4] += 1 and d[7] −= 1. Two writes whatever the length of the range — d is now [3, 2, 0, −2, 0, −2, −1].
5. Then one prefix sum over d gives the array: a[1] = d[1] = 3. The +3 that starts at 1 is carried along by the running sum.
6. a[2] = a[1] + 2 = 5. d[2] nets every change here: +2 starts.
7. a[4] = a[3] − 2 = 3. d[4] nets every change here: +3 stops, +1 starts.
8. a[6] = 1. Each element is the sum of every change that started at or before it, minus every one that already stopped.
9. a = [3, 5, 5, 3, 3, 1], the same as applying each update element by element. 3 updates cost 6 writes plus one O(n) pass: O(n + q) instead of O(n · q).

Which one, when?

- Many queries, no updates → **prefix sums**.
- Many updates, then one read → **difference array**.
- Updates and queries mixed → neither; a Fenwick or segment tree (later in this track).
- Rectangles of a grid → 2D prefix sums: big block − top strip − left strip + corner.

GAME · QUICK FIRE

Prefix array, difference array, or neither?

Pick the tool that makes the problem fit the time limit.

1/5

2·10^5 queries "sum of a[l..r]" on an array that never changes.

A Prefix sums

B Difference array

C Sum each range directly

More practice

- Range Sum Query - Immutable — the prefix array itself.
- Find Pivot Index — left sum against right sum.
- Subarray Sum Equals K — prefixes in a hash map.
- Corporate Flight Bookings — a difference array.
- Car Pooling — range updates, then a scan.

EXERCISE 1

Range sum queries

Read an array of `n` integers and answer `q` queries: each gives `l r` (1-indexed) — print `a[l] + … + a[r]`.

Constraints: `1 ≤ n, q ≤ 2·10⁵`, `-10⁹ ≤ aᵢ ≤ 10⁹`, `1 ≤ l ≤ r ≤ n`.

Input: `n q`, the array, then `q` lines `l r`. **Output:** one sum per line.

```
5 3
1 2 3 4 5
1 5
2 3
4 4
```

prints

```
15
5
4
```

INPUT 1

```
5 3
1 2 3 4 5
1 5
2 3
4 4
```

OUTPUT 1

```
15
5
4
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, q;
    cin >> n >> q;
    vector<ll> p(n + 1, 0);
    for (int i = 1; i <= n; ++i) {
        ll x;
        cin >> x;
        p[i] = p[i - 1] + x;
    }
    while (q--) {
        int l, r;
        cin >> l >> r;
        cout << p[r] - p[l - 1] << '\n';
    }
    return 0;
}
```

EXERCISE 2

Many range additions

An array of `n` zeros receives `m` updates. Update `l r v` adds `v` to every element from `l` to `r` (1-indexed). Print the array after all of them.

Constraints: `1 ≤ n, m ≤ 2·10⁵`, `1 ≤ l ≤ r ≤ n`, `-10⁹ ≤ v ≤ 10⁹`.

Input: `n m`, then `m` lines `l r v`. **Output:** the `n` final values on one line, separated by spaces.

```
5 3
1 3 2
2 5 3
5 5 -1
```

prints

```
2 5 5 3 2
```

INPUT 1

```
5 3
1 3 2
2 5 3
5 5 -1
```

OUTPUT 1

```
2 5 5 3 2
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<ll> d(n + 2, 0);
    for (int k = 0; k < m; ++k) {
        int l, r;
        ll v;
        cin >> l >> r >> v;
        d[l] += v;
        d[r + 1] -= v;
    }
    ll run = 0;
    for (int i = 1; i <= n; ++i) {
        run += d[i];
        cout << run << (i == n ? '\n' : ' ');
    }
    return 0;
}
```

QUIZ

1. With `p[0] = 0` and `p[i] = p[i - 1] + a[i]` (1-indexed), the sum `a[l] + … + a[r]` is…

A. `p[r] - p[l]`

B. `p[r] - p[l - 1]`

C. `p[r + 1] - p[l]`

D. `p[r] + p[l - 1]`

2. To add `v` to every element of `[l, r]` with a difference array `d`, you write…

A. `d[l] += v; d[r] -= v;`

B. `d[l] += v; d[r + 1] -= v;`

C. `d[l - 1] += v; d[r] -= v;`

D. `d[i] += v` for every i in l..r

3. n = 10^5 values and q = 10^5 range-sum queries, no updates. Which costs the least?

A. Sum each range directly: O(n · q)

B. Prefix array: O(n + q)

C. Sort first: O(n log n + q · n)

D. A difference array

4. Updates and range-sum queries arrive interleaved — query, update, query… Does a plain prefix array still work?

A. Yes, in O(1) each

B. No — one update changes O(n) prefix values, so each update costs O(n); a Fenwick or segment tree handles both in O(log n)

C. Yes, if you use long long

D. Only for positive values

ANSWER KEY · PREFIX SUMS AND DIFFERENCE ARRAYS

1. B p[r] holds everything up to r, p[l − 1] everything before l; their difference is exactly l..r. `p[r] - p[l]` drops a[l] — the classic off-by-one.

2. B The running sum picks up +v at l and carries it to every later index; −v at r + 1 cancels it right after the range. Cancelling at r would leave a[r] without the update.

3. B One O(n) pass builds the prefix array, and each query is then two lookups. Directly summing is up to 10^{10} steps. A difference array is for many updates, not many queries.

4. B Prefix sums and difference arrays shine when all the updates come before all the reads (or there are none). Mixed online operations need a tree, later in this track.

CHECKPOINT · 45 MIN

Checkpoint — Prefix sums and difference arrays

The prefix sums checkpoint. It is timed: **45 minutes** from when you start it.

What it covers. Prefix arrays and their indexes (`p[r] - p[l - 1]`), difference arrays (`d[l] += v`, `d[r + 1] -= v`), when each applies and when neither does, 2D prefix sums, and counting subarrays with a hash map of prefixes.

How it works. Eleven questions and one program. Clear 70% of the questions and get the program accepted. The program's timeline is a million hours long and the bookings run into the thousands, so adding each booking hour by hour runs out of time.

Before you start, make sure you can say from memory:

- why the prefix array has a `p[0]` and the difference array a slot `n + 1`;
- which tool fits "many queries", "many updates", and both mixed;
- the four-term formula for a rectangle's sum.

EXERCISE

The busiest hour

A venue's timeline has hours `1` to `10⁶`. Each of `m` bookings `s e p` brings `p` people for every hour from `s` to `e`, both included. Print the largest number of people present in any one hour, and the earliest hour at which it happens.

Constraints: `1 ≤ m ≤ 2·10⁵`, `1 ≤ s ≤ e ≤ 10⁶`, `1 ≤ p ≤ 10⁹`.

Input: `m`, then `m` lines `s e p`. **Output:** `<people> <hour>`.

```
3
1 4 2
3 6 5
6 6 1
```

prints

```
7 3
```

(hours 3 and 4 hold 2 + 5 = 7 people; hour 3 is earlier.)

INPUT 1

```
3
1 4 2
3 6 5
6 6 1
```

OUTPUT 1

```
7 3
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    const int H = 1000000;
    int m;
    cin >> m;
    vector<ll> d(H + 2, 0);
    for (int k = 0; k < m; ++k) {
        int s, e;
        ll p;
        cin >> s >> e >> p;
        d[s] += p;
        d[e + 1] -= p;
    }
    ll run = 0, best = -1;
    int at = 0;
    for (int h = 1; h <= H; ++h) {
        run += d[h];
        if (run > best) {
            best = run;
            at = h;
        }
    }
    cout << best << ' ' << at << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. a = [3, 1, 4, 1, 5] (1-indexed). What is p[4] in the prefix array with p[0] = 0?

A. 4

B. 8

C. 9

D. 14

2. Same array. Which expression gives a[2] + a[3] + a[4]?

A. p[4] − p[2]

B. p[4] − p[1]

C. p[3] − p[1]

D. p[5] − p[2]

3. Why does the prefix array have n + 1 slots, with p[0] = 0?

A. Alignment

B. So a range starting at index 1 is p[r] − p[0] like every other range — no special case

C. To store the length

D. It does not need one

4. Why does the difference array need a slot n + 1?

A. It does not

B. An update ending at r = n writes −v at d[n + 1]; without that slot it writes past the end

C. For the answer

D. To hold the total

5. d = [0, 2, 3, 0, −2, −3] for indexes 0..5 (d[0] unused). What is the array a[1..4] after the running sum?

A. 2 5 5 3

B. 2 3 0 −2

C. 2 5 3 0

D. 5 5 5 5

6. n = 10^5 and q = 10^5 range-sum queries, each answered by summing the range directly. Roughly how many steps in the worst case?

A. 10^5

B. $2 \cdot 10^5$

C. 10^{10}

D. 10^{15}

7. All updates come first, then you print the array once. The right tool is…

A. A prefix array

B. A difference array

C. Sorting

D. A hash map

8. Values up to 10^9, n up to $2 \cdot 10^5$. What type must the prefix array be?

A. `int`

B. `long long`

C. `unsigned int`

D. `short`

9. Subarrays summing to exactly k (values may be negative) can be counted in O(n) by…

A. Sliding a window

B. Walking the prefix sums and counting, in a hash map, how often each earlier prefix equals the current prefix − k

C. Sorting the array

D. A difference array

10. In 2D, the sum of the rectangle with rows r1..r2 and columns c1..c2 from a prefix table P (with a zero row and column) is…

A. P[r2][c2] − P[r1 − 1][c1 − 1]

B. P[r2][c2] − P[r1 − 1][c2] − P[r2][c1 − 1] + P[r1 − 1][c1 − 1]

C. P[r2][c2] − P[r1][c2] − P[r2][c1]

D. P[r2][c2] + P[r1 − 1][c1 − 1]

11. Updates and queries are mixed — each update must be visible to the next query. Prefix sums alone cost…

A. O(1) per operation

B. O(n) per update, because every later prefix changes

C. O(log n) per operation

D. Nothing extra

ANSWER KEY · CHECKPOINT — PREFIX SUMS AND DIFFERENCE ARRAYS

1. C p[4] = 3 + 1 + 4 + 1 = 9. p[5] would be 14.

2. B p[4] − p[1] = 9 − 3 = 6 = 1 + 4 + 1. The lower index is l − 1 = 1.

3. B p[0] is the sum of no elements. Without it, every query with l = 1 needs an if.

4. B d[r + 1] −= v with r = n touches index n + 1. That slot is never read when rebuilding a[1..n] — it only gives the update a place to stop.

5. A Running sums of d[1..4]: 2, 2 + 3 = 5, 5 + 0 = 5, 5 − 2 = 3.

6. C Each query can cover all n elements: $n \cdot q = 10^{10}$, about a hundred seconds at 10^8 a second. With a prefix array it is $n + q = 2 \cdot 10^5$.

7. B Two writes per update, then one running sum: O(n + m). The prefix array is the read-side twin — many queries, no updates.

8. B A prefix can reach $2 \cdot 10^{14}$, far past int's $2.1 \cdot 10^9$ (and unsigned int's $4.3 \cdot 10^9$).

9. B a[l..r] = k means p[r] − p[l − 1] = k, so p[l − 1] = p[r] − k: look it up among the prefixes seen so far. A sliding window needs all values non-negative.

10. B Take the big block, subtract the strip above and the strip to the left, then add back the corner both strips removed — inclusion–exclusion.

11. B One changed element shifts every prefix after it. Interleaved updates and queries are the job of a Fenwick or segment tree, O(log n) each.