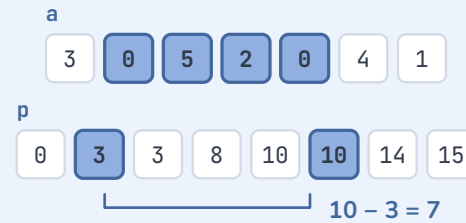


# Prefix sums

Chapter 2.1 · Overall level: Core



Many problems ask the same kind of question over and over: *what is the total of this stretch of the list?* A prefix sum array answers every one of those questions with a single subtraction, after one quick pass over the data.

The same running-total idea also handles many range additions at once and rectangles in a grid, so it is worth learning carefully.

## 2.1.1 The repeated-sum problem

A weather station records rainfall, in millimetres, once a day. A researcher wants many totals: days 2 to 5, days 1 to 7, days 4 to 6, and so on. The direct method adds the days in each range one by one.

That works, but count the work. With  $n$  days and  $q$  questions, a single question can touch up to  $n$  values, so the total is about  $n \times q$  additions. If  $n$  and  $q$  are both 200,000, that is 40,000,000,000 additions, far more than a program can afford to repeat.

The waste is that neighbouring questions add the same days again and again. Prefix sums do that shared work once.

## 2.1.2 Building the prefix array

Number the values  $a[1]$ ,  $a[2]$ , ...,  $a[n]$ . The prefix array  $p$  stores running totals:

$$\begin{aligned} p[0] &= 0 \\ p[i] &= p[i - 1] + a[i] \quad \text{for } i = 1, 2, \dots, n \end{aligned}$$

So  $p[i]$  is the total of the first  $i$  values. The extra entry  $p[0] = 0$  stands for “the total of nothing”, and it is what makes ranges starting at position 1 work without a special case.

|          |   |   |   |   |    |    |    |    |
|----------|---|---|---|---|----|----|----|----|
| index    | 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7  |
| <b>a</b> |   | 3 | 0 | 5 | 2  | 0  | 4  | 1  |
| <b>p</b> | 0 | 3 | 3 | 8 | 10 | 10 | 14 | 15 |

Figure 2.1.1 — Seven days of rainfall (a) and the running totals (p). Each p value is the one to its left plus the a value above it. Building p takes a single pass:  $O(n)$  time.

### 2.1.3 Answering a range in $O(1)$

To find the total of positions  $l$  through  $r$ , take everything up to  $r$  and remove everything before  $l$ :

$$\text{sum}(l, r) = p[r] - p[l - 1]$$

Why it works:  $p[r] = a[1] + \dots + a[l - 1] + a[l] + \dots + a[r]$ , and  $p[l - 1] = a[1] + \dots + a[l - 1]$ .

Subtracting cancels the shared beginning and leaves exactly  $a[l] + \dots + a[r]$ . Negative values do not change this: the shared part cancels whatever its sign.

|          |   |   |   |   |    |    |    |    |
|----------|---|---|---|---|----|----|----|----|
| index    | 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7  |
| <b>a</b> |   | 3 | 0 | 5 | 2  | 0  | 4  | 1  |
| <b>p</b> | 0 | 3 | 3 | 8 | 10 | 10 | 14 | 15 |

Figure 2.1.2 — The highlighted days are the range. Only the two highlighted p values are needed, whatever the length of the range.

### 2.1.4 Worked example

Using the rainfall data above, answer three questions. Each one costs one subtraction.

| Question | Days   | Formula                | Result | Check by hand   |
|----------|--------|------------------------|--------|-----------------|
| 1        | 2 to 5 | $p[5] - p[1] = 10 - 3$ | 7      | $0 + 5 + 2 + 0$ |
| 2        | 1 to 7 | $p[7] - p[0] = 15 - 0$ | 15     | all seven days  |
| 3        | 4 to 6 | $p[6] - p[3] = 14 - 8$ | 6      | $2 + 0 + 4$     |

Question 2 shows why  $p[0]$  matters: without it, a range that starts on day 1 would need its own branch in the code.

## Pseudocode

Prefix sums:  $O(n)$  to build,  $O(1)$  per question

```

read n, q and the values a[1..n]
p[0] ← 0
for i from 1 to n:
    p[i] ← p[i - 1] + a[i]
for each question (l, r):
    output p[r] - p[l - 1]
```

### 2.1.5 Positions in the input, indices in the code

Statements usually number positions from 1 and include both ends, as in “days 2 to 5”. Most programming languages number list elements from 0. The prefix array bridges the two:  $p[k]$  is the total of the first  $k$  values, so  $k$  counts values rather than naming one of them. Think of  $k$  as a boundary between values.

With 0-based code, the inclusive input range  $l..r$  is the half-open stretch of indices  $[l - 1, r)$ : it starts at index  $l - 1$  and stops just before index  $r$ . A half-open stretch  $[x, y)$  always has total  $p[y] - p[x]$ , which is the same formula seen from the code side.

input range  $l..r \rightarrow$  indices  $[l - 1, r) \rightarrow p[r] - p[l - 1]$

Days 2 to 5 become indices  $[1, 5)$ , and the answer is  $p[5] - p[1]$ , exactly as in Figure 2.1.2.

### 2.1.6 Code in four languages

Each program solves the first practice problem, **Range totals**. The first line holds  $n$  and  $q$ , the second line the  $n$  values, then  $q$  lines “ $l\ r$ ” with positions starting at 1. It prints one total per line.

| Sample input | Sample output |
|--------------|---------------|
| 5 3          | 7             |
| 3 1 4 2 5    | 15            |
| 2 4          | 4             |
| 1 5          |               |
| 3 3          |               |

The four programs follow the pseudocode line for line. Only the input reading differs: each language has its own fast way to read many numbers.

## Python 3

```
import sys

def main():
    data = sys.stdin.buffer.read().split()
    n, q = int(data[0]), int(data[1])
    p = [0] * (n + 1)
    for i in range(n):
        p[i + 1] = p[i] + int(data[2 + i])
    out = []
    idx = 2 + n
    for _ in range(q):
        l, r = int(data[idx]), int(data[idx + 1])
        idx += 2
        out.append(p[r] - p[l - 1])
    print("\n".join(map(str, out)))

main()
```

## C++17

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, q;
    cin >> n >> q;
    vector<long long> p(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        long long 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;
}
```

## Java 17

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) throws IOException {
        StreamTokenizer in = new StreamTokenizer(
            new BufferedReader(new InputStreamReader(System.in)));
        in.nextToken();
        int n = (int) in.nval;
        in.nextToken();
        int q = (int) in.nval;
        long[] p = new long[n + 1];
        for (int i = 1; i <= n; i++) {
            in.nextToken();
            p[i] = p[i - 1] + (long) in.nval;
        }
        StringBuilder out = new StringBuilder();
        for (int k = 0; k < q; k++) {
            in.nextToken(); int l = (int) in.nval;
            in.nextToken(); int r = (int) in.nval;
            out.append(p[r] - p[l - 1]).append('\n');
        }
        System.out.print(out);
    }
}
```

## JavaScript (Node)

```
const data = require("fs").readFileSync(0, "utf8")
    .trim().split(/\s+/).map(Number);
let pos = 0;
const n = data[pos++], q = data[pos++];
const p = new Array(n + 1).fill(0);
for (let i = 1; i <= n; i++) p[i] = p[i - 1] + data[pos++];
const out = [];
for (let k = 0; k < q; k++) {
    const l = data[pos++], r = data[pos++];
    out.push(p[r] - p[l - 1]);
}
console.log(out.join("\n"));
```

In JavaScript, ordinary numbers are exact up to about  $9 \times 10^{15}$ . Range totals here stay below  $2 \times 10^{11}$ ; for larger totals, use `BigInt`.

## 2.1.7 Common mistakes

- **Off by one.** The answer is  $p[r] - p[l - 1]$ , not  $p[r] - p[l]$ . The second version silently drops  $a[l]$ . Test a range of length 1:  $\text{sum}(4, 4)$  must equal  $a[4]$ .
- **Forgetting  $p[0]$ .** If the array starts at  $p[1] = a[1]$ , ranges that begin at position 1 read outside the array or need a special case. Allocate  $n + 1$  entries and set  $p[0] = 0$ .
- **Overflow.** Totals grow much larger than the inputs. 200,000 values of size  $10^6$  add up to  $2 \times 10^{11}$ , which does not fit in a 32-bit `int` (limit about  $2.1 \times 10^9$ ). Use `long` `long` in C++ and `long` in Java.
- **Mixing updates with a fixed table.** If a value changes after  $p$  is built, every later prefix is now wrong. A static table suits questions on unchanging data; Chapter 4.2 handles values that change between questions.

## 2.1.8 Many range additions: the difference array

Now turn the problem around. Start with  $n$  zeros and apply many instructions of the form “add  $v$  to every position from  $l$  to  $r$ ”, and only look at the final array at the end. Updating every position for every instruction repeats work again.

Instead, record only where each addition starts and where it stops, in a difference array  $D$  with one spare slot after position  $n$ :

```
D[l] += v
D[r + 1] -= v
```

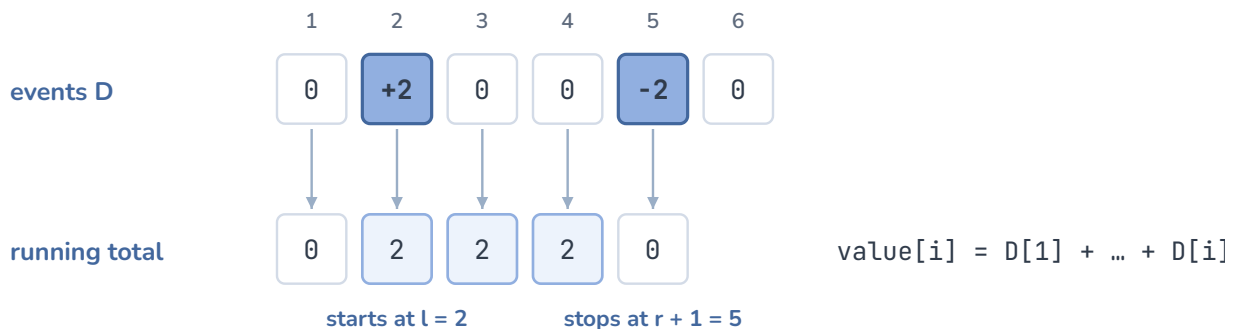


Figure 2.1.3 — A range update becomes two boundary events, followed by one reconstruction pass. The running total of  $D$  is the final array.

Why it works: the running total at position  $i$  adds every  $D[j]$  with  $j \leq i$ . An instruction contributes  $+v$  once the running total passes  $l$ , and the  $-v$  at  $r + 1$  cancels it from then on, so exactly positions  $l$  to  $r$  receive  $v$ . Different instructions simply add, even when they overlap or  $v$  is negative.

For example, “add 2 to positions 2–4” and then “add 1 to positions 1–3” on five zeros give  $D[1..6] = [1, 2, 0, -1, -2, 0]$ , whose running totals are 1, 3, 3, 1, 0. Positions 2 and 3 received both additions.

```

D[1..n + 1] ← 0
for each instruction (l, r, v):
    D[l] ← D[l] + v
    D[r + 1] ← D[r + 1] - v
running ← 0
for i from 1 to n:
    running ← running + D[i]
    output running

```

Each instruction costs  $O(1)$  and the final pass costs  $O(n)$ , so  $q$  instructions take  $O(n + q)$  time instead of up to  $n \times q$ . The difference array and the prefix array undo each other: prefix sums of  $D$  rebuild the values, and differences of neighbouring prefix values give back  $a$ .

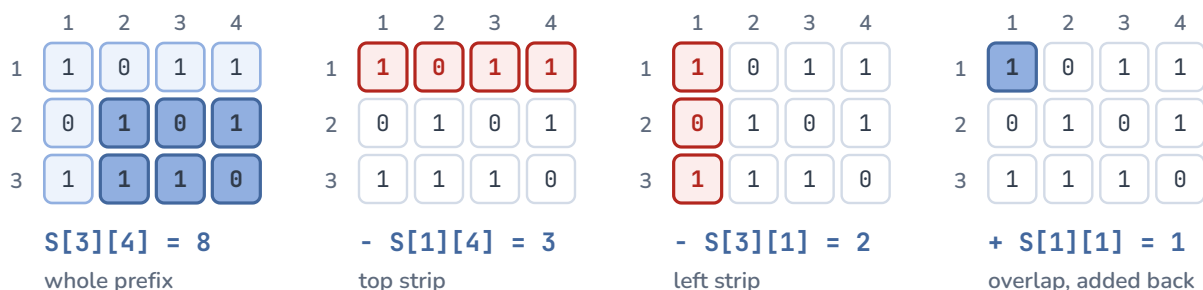
## 2.1.9 Two-dimensional prefix sums

The same idea works on a grid. Let  $S[i][j]$  be the total of the rectangle from the top-left corner (1, 1) to  $(i, j)$ . Row 0 and column 0 are zeros. Each entry is built from three already-known neighbours:

$$S[i][j] = a[i][j] + S[i-1][j] + S[i][j-1] - S[i-1][j-1]$$

The last term is subtracted because the top-left block was counted twice. A rectangle with corners  $(r1, c1)$  and  $(r2, c2)$  then costs four lookups:

$$S[r2][c2] - S[r1-1][c2] - S[r2][c1-1] + S[r1-1][c1-1]$$



**Answer for rows 2–3, columns 2–4:  $8 - 3 - 2 + 1 = 4$**

Figure 2.1.4 — Subtract the unwanted strips, then add back their overlap because it was removed twice. Row and column numbers are 1-based, as in the input.

|      | c = 0 | 1 | 2 | 3 | 4 |
|------|-------|---|---|---|---|
| S[0] | 0     | 0 | 0 | 0 | 0 |
| S[1] | 0     | 1 | 1 | 2 | 3 |
| S[2] | 0     | 1 | 2 | 3 | 5 |
| S[3] | 0     | 2 | 4 | 6 | 8 |

The full table S for the grid in Figure 2.1.4.

Building it row by row takes  $O(nm)$  time and memory; each rectangle question then takes  $O(1)$ , so  $q$  questions cost  $O(nm + q)$  in total.

For a  $1000 \times 1000$  grid the table holds about a million numbers. That is usually fine, but it is the memory, not the arithmetic, that grows fastest.

## 2.1.10 Choosing the right tool

| Situation                                     | Good starting point                                  |
|-----------------------------------------------|------------------------------------------------------|
| Many range sums, values never change          | Prefix sums                                          |
| Every block has the same length $k$           | Prefix sums: block $l..l+k-1$ is $p[l+k-1] - p[l-1]$ |
| Many range additions, answers only at the end | Difference array                                     |
| Many rectangle counts on a fixed grid         | Two-dimensional prefix sums                          |
| Questions mixed with changing values          | Fenwick or segment tree (Chapter 4.2)                |

## Exercises

Difficulty: ● quick check, ●● needs an extra idea. Try each one before reading the answers.

- For  $a = [4, -1, 2, 6, -3]$ , write out  $p[0]$  to  $p[5]$ . Then find the total of positions 2 to 4 using  $p$ .
- Why does the prefix array need  $n + 1$  entries rather than  $n$ ?
- For  $a = [5, 8, 2, 7, 4, 1]$ , explain how to answer “how many even numbers are in positions  $l$  to  $r$ ?” in  $O(1)$  per question, and give the answer for positions 2 to 5.
- Start with five zeros. Add 2 to positions 1–3, add 1 to positions 2–5, and add 5 to position 4. Find the final array with a difference array and a single pass at the end.
- Using the table S above, find the number of ones in rows 1–3, columns 3–4. Which four entries do you need?

## Answers and explanations

Read these after attempting the exercises.

1.  $p = [0, 4, 3, 5, 11, 8]$ . Total of positions 2–4 is  $p[4] - p[1] = 11 - 4 = 7$  (check:  $-1 + 2 + 6$ ).
2. The extra entry  $p[0] = 0$  is the empty prefix. It lets a range that starts at position 1 use the same formula,  $p[r] - p[0]$ .
3. Build a prefix array over “is even” markers (1 if even, 0 if odd):  $e = [0, 0, 1, 2, 2, 3, 3]$ . The count for  $l$  to  $r$  is  $e[r] - e[l - 1]$ . For 2 to 5:  $e[5] - e[1] = 3 - 0 = 3$  (8, 2 and 4). Prefix sums work for any quantity you can add up, not only the values themselves.
4.  $D[1..6] = [2, 1, 0, 3, -5, -1]$ : “add 2 to 1–3” gives  $D[1] += 2$  and  $D[4] -= 2$ , “add 1 to 2–5” gives  $D[2] += 1$  and  $D[6] -= 1$ , and “add 5 to 4–6” gives  $D[4] += 5$  and  $D[5] -= 5$ . Running totals give the final array **[2, 3, 3, 6, 1]**.
5.  $S[3][4] - S[0][4] - S[3][2] + S[0][2] = 8 - 0 - 4 + 0 = 4$ . Check: column 3 holds 1, 0, 1 and column 4 holds 1, 1, 0.

## Practice this topic

Four coding problems for this chapter are in the **Practice** tab. Open any of them in any order; the label shows difficulty within this topic.

### Range totals

Apply

Answer  $q$  range-sum queries on an unchanged array. Each query gives inclusive, 1-based positions  $l$  and  $r$ .

### Best fixed block

Connect

Among all consecutive blocks of exactly  $k$  elements, print the largest sum. Values may be negative.

### Range paint counts

Reason

An initially zero array has  $n$  positions. Each of  $q$  instructions adds integer  $v$  to every position in inclusive range  $l..r$ . After all instructions, print the final array.

### Rectangle sensors

Stretch

A grid contains 0 or 1 in each cell. Answer queries counting 1s in an inclusive rectangle with top-left( $r1, c1$ ) and bottom-right( $r2, c2$ ).