Module 1 of 12

Fast I/O and complexity budgets

The contest template, fast input and output, long long by default, and reading the constraints to know which complexity will pass in one second.

Every contest problem is two problems: find an idea, and find one that is fast enough. This module settles the second one before you write a line. A judge runs roughly 10^8 simple operations a second, so the largest n in the constraints already names the slowest algorithm that can pass — $n \le 10^5$ rules out $O(n^2)$, $n \le 20$ invites $O(2^n)$.

It also gives you the template the rest of the track starts from: `bits/stdc++.h`, `using namespace std`, the two fast-I/O lines and `long long` for anything that sums or multiplies. The exercises are judged at the limits: a sum that overflows an `int`, and a pair count where n reaches 10^9 and only the counting idea fits in time.

CONTENTS

1	The contest template and complexity budgets	15 min
✓	Checkpoint — Fast I/O and complexity budgets	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

The contest template and complexity budgets

A contest problem is judged twice: is the answer right, and did it arrive within the time limit? This lesson settles the second question before you write a line.

The template

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<ll> a(n);
    for (auto& x : a) cin >> x;
    // solve
    cout << answer << '\n';
}
```

- `bits/stdc++.h` includes the whole standard library at once (GCC's libstdc++, which judges use). Fine for a contest, never in a project.
- `using namespace std` saves typing in a one-file solution.
- The two fast-I/O lines make `cin` / `cout` fast enough for 10^6 numbers. After them, never mix in `scanf` / `printf`.
- `ll` — **long long by default**. An `int` stops at about 2.1×10^9; a sum of 10^5 values of 10^9 is 10^{14}.
- Print `'\n'`, not `endl`: `endl` flushes on every line.

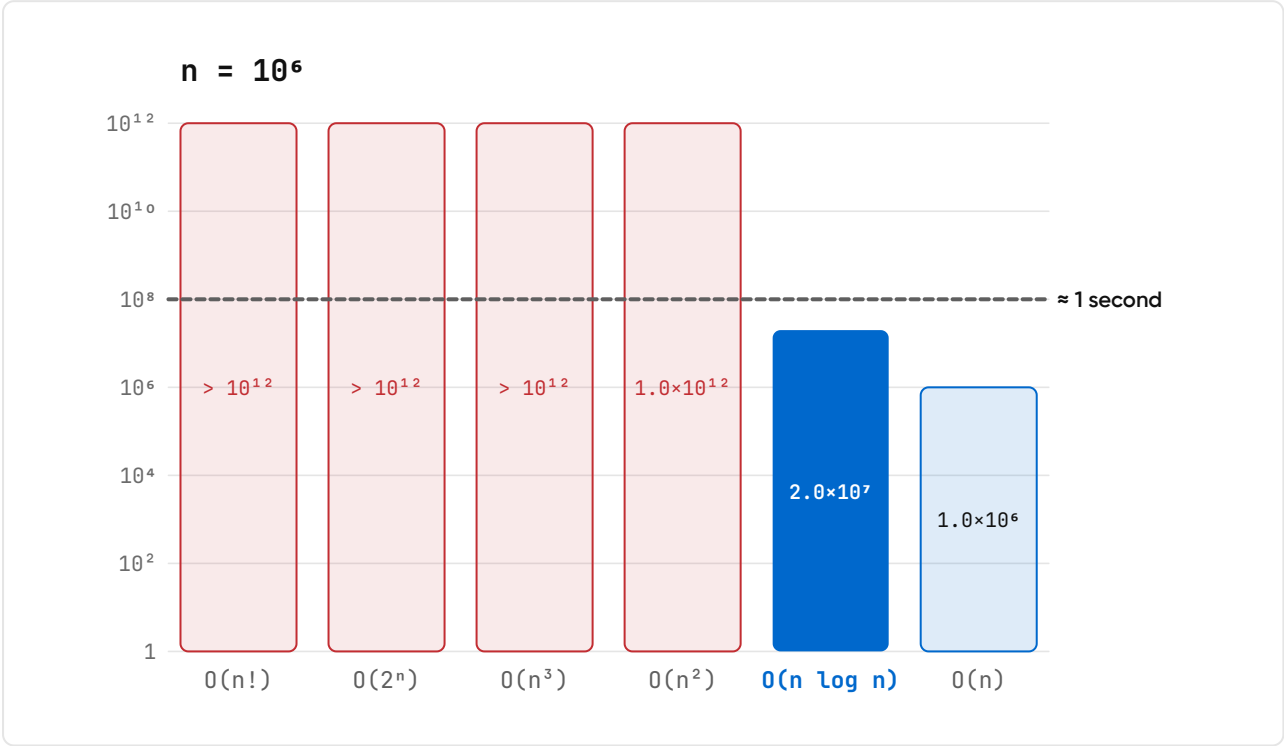
Read the constraints first

A judge runs roughly **10^8 simple operations per second**. Put the largest n into your idea's complexity:

- $n \le 10^5$ and an $O(n^2)$ idea → 10^{10} operations, a hundred times over. Too slow.
- $n \le 10^5$ and $O(n \log n)$ → about 1.7×10^6. Passes easily.

So the constraint on n names the slowest algorithm the setter allows. Watch the bars move as n grows:

Which complexity fits a 1-second limit?



1. n = 11: even O(n!) is only 4.0×10^7 operations, so anything up to O(n!) fits.
2. n = 25: $O(2^n)$ is 3.4×10^7 operations, under the 10^8 line, while O(n!) would be $> 10^{12}$. The slowest class that fits is $O(2^n)$, so that is the budget.
3. n = 400: $O(n^3)$ is 6.4×10^7 operations, under the 10^8 line, while $O(2^n)$ would be $> 10^{12}$. The slowest class that fits is $O(n^3)$, so that is the budget.
4. $n = 10^4$: $O(n^2)$ is 1.0×10^8 operations, under the 10^8 line, while $O(n^3)$ would be 1.0×10^{12}. The slowest class that fits is $O(n^2)$, so that is the budget.
5. $n = 10^6$: O(n log n) is 2.0×10^7 operations, under the 10^8 line, while $O(n^2)$ would be 1.0×10^{12}. The slowest class that fits is O(n log n), so that is the budget.
6. Read it the other way in a contest: the constraint on n names the slowest algorithm allowed. $n \le 10^6$ means O(n log n) or better; an $O(n^2)$ idea is over by a factor of ten thousand.

The table to keep in your head, every row computed at 10^8:

- $n \le 11$ → O(n!) · $n \le 25$ → $O(2^n)$ · $n \le 400$ → $O(n^3)$
- $n \le 10^4$ → $O(n^2)$ · $n \le 10^6$ → O(n log n) · $n \le 10^8$ → O(n)
- n up to 10^{18} → O(log n) or a formula

Habits that save a verdict

- Read the constraints **before** the idea. They are a hint.
- Use the worst case, not the example: the hidden tests are at the limit.
- Two seconds doubles the budget; it does not turn $O(n^2)$ at $n = 10^5$ into a pass.
- `1LL * a * b`, not `a * b`, when `a` and `b` are `int`.

More practice

- Two Sum — $O(n^2)$ is the trap, a hash map is $O(n)$.
- Contains Duplicate — sort or hash.
- Maximum Subarray — $O(n)$ after spotting $O(n^2)$.
- Sort an Array — $O(n \log n)$ or bust.

EXERCISE 1

Sum and maximum, fast

Read `n` and then `n` integers. Print their sum and their maximum, separated by a space.

Constraints: $1 \le n \le 2 \cdot 10^5$, $-10^9 \le a_i \le 10^9$.

Input: `n` on the first line, the values on the second. **Output:** `<sum> <max>`.

```
5
3 -1 4 1 5
```

prints

```
12 5
```

INPUT 1

```
5
3 -1 4 1 5
```

OUTPUT 1

```
12 5
```

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;
    cin >> n;
    ll sum = 0, best = LLONG_MIN;
    for (int i = 0; i < n; ++i) {
        ll x;
        cin >> x;
        sum += x;
        best = max(best, x);
    }
    cout << sum << ' ' << best << '\n';
    return 0;
}
```

EXERCISE 2

Pairs that add up to a multiple of k

Count the pairs `(i, j)` with `1 ≤ i < j ≤ n` whose sum `i + j` is divisible by `k`.

Constraints: `1 ≤ n ≤ 10⁹`, `1 ≤ k ≤ 10⁶`. Time limit about one second.

Input: `n k`. **Output:** the number of pairs.

```
5 3
```

prints

```
4
```

(the pairs are 1+2, 1+5, 2+4 and 4+5.)

Read the constraints before you code: trying every pair is O(n²), and n reaches 10⁹.

INPUT 1

```
5 3
```

OUTPUT 1

```
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);
    ll n, k;
    cin >> n >> k;
    // How many of 1..n leave remainder r when divided by k.
    auto cnt = [&](ll r) -> ll { return n / k + ((r >= 1 && r <= n % k) ? 1 : 0); };
    auto pairsWithin = [](ll c) -> ll { return c * (c - 1) / 2; };
    ll total = pairsWithin(cnt(0));
    for (ll r = 1; r < k - r; ++r) total += cnt(r) * cnt(k - r);
    if (k % 2 == 0) total += pairsWithin(cnt(k / 2));
    cout << total << '\n';
    return 0;
}
```

QUIZ

1. $n \le 10^5$ and your idea compares every pair of elements. Will it pass a 1-second limit?

A. Yes — 10^5 is small

B. No — about $n^2/2 = 5 \cdot 10^9$ comparisons, roughly 50 times the ~10^8 a judge does in a second

C. Only with fast I/O

D. Only in C++

2. What do `ios::sync_with_stdio(false);` and `cin.tie(nullptr);` buy you?

A. The first stops cin/cout synchronising with C's stdio, the second stops cout being flushed before every read — together they make large input and output several times faster

B. They make the algorithm run in O(n)

C. They enable 64-bit integers

D. Nothing on a modern compiler

3. `int s = 0;` then $2 \cdot 10^5$ values, each up to 10^9, are added to `s`. What goes wrong?

A. Nothing

B. The sum reaches $2 \cdot 10^{14}$, past an int's $2.1 \cdot 10^9$: signed overflow, which is undefined behaviour and in practice a wrong answer

C. It is too slow

D. The compiler refuses it

4. Why print with `'\n'` rather than `endl` when the output has 10^5 lines?

A. `endl` is not standard

B. `endl` also flushes the stream every time — 10^5 system calls instead of a few large writes

C. `'\n'` is shorter to type

D. There is no difference

ANSWER KEY · THE CONTEST TEMPLATE AND COMPLEXITY BUDGETS

1. B Put the largest n into the complexity: $(10^5)^2 / 2 = 5 \cdot 10^9$. At about 10^8 simple operations per second that is close to a minute. Fast I/O speeds up reading the input, not the algorithm — $n \le 10^5$ is asking for O(n log n) or O(n).

2. A Synchronisation with stdio and the flush before each read are both per-operation costs; with 10^5–10^6 numbers in the input they add up to a real share of the time limit. After the first line, do not mix `scanf` / `printf` with `cin` / `cout`.

3. B An `int` holds about $\pm 2.1 \cdot 10^9$. Anything that sums or multiplies large values goes in `long long` ($\pm 9.2 \cdot 10^{18}$), which is why the template has `using ll = long long;`.

4. B A newline character goes into the buffer, which is written out in large chunks. `endl` writes the newline and then forces a flush, so a loop printing with it pays a write per line.

CHECKPOINT · 45 MIN

Checkpoint — Fast I/O and complexity budgets

The first timed checkpoint of the track. Like a contest round, it has a clock: **45 minutes** from when you start it.

What it covers. Turning a constraint into a complexity budget at about 10^8 operations per second, the contest template's fast-I/O lines, and `long long` against overflow.

How it works. Eleven questions and one program. Clear 70% of the questions and get the program accepted — its hidden cases are at the limits, so an O(n · q) scan runs out of time where an O((n + q) log n) one passes.

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

- the slowest complexity that fits n ≤ 20, 400, 5000, 10^6 and 10^{18};
- what `1LL * a * b` fixes;
- why `endl` is slow and what `cin.tie(nullptr)` removes.

EXERCISE

How many are at most x

Read `n` values and then answer `q` queries: for each query value `x`, print how many of the values are `≤ x`.

Constraints: `1 ≤ n, q ≤ 2·10^5`, `0 ≤ a_i, x ≤ 10^9`.

Input: `n q`, then the `n` values, then `q` lines with one `x` each. **Output:** `q` lines, one answer each.

```
5 3
4 1 7 4 2
4
0
10
```

prints

```
4
0
5
```

Scanning all n values per query is $O(n \cdot q) = 4 \cdot 10^{10}$ at the limits — budget first.

INPUT 1

```
5 3
4 1 7 4 2
4
0
10
```

OUTPUT 1

```
4
0
5
```

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> a(n);
    for (auto& x : a) cin >> x;
    sort(a.begin(), a.end());
    while (q--) {
        ll x;
        cin >> x;
        cout << (upper_bound(a.begin(), a.end(), x) - a.begin()) << '\n';
    }
    return 0;
}
```

CHECKPOINT QUESTIONS

1. $n \le 20$. Which is the slowest complexity that still fits in about 10^8 operations?

A. $O(n!)$

B. $O(2^n \cdot n)$

C. $O(n^4 \cdot 2^n)$

2. $n \le 5000$. Which fits?

A. $O(n^3)$

B. $O(n^2)$

C. Neither

3. $n \le 10^6$. Which fits?

A. $O(n^2)$

B. $O(n\sqrt{n})$

C. $O(n \log n)$

4. $n \le 10^{18}$. What can the solution afford?

A. $O(n)$

B. $O(\sqrt{n})$

C. $O(\log n)$

5. $n \le 400$. Which is the slowest complexity that fits?

A. $O(n^3)$

B. $O(n^4)$

C. $O(2^n)$

6. Which line belongs in a contest template but never in a header file of a real project?

A. `using namespace std;`

B. `int main()`

C. `return 0;`

D. `#include <vector>`

7. After `ios::sync_with_stdio(false);`, which is safe?

A. Mixing `printf` and `cout`

B. Using only `cin` / `cout`

C. Mixing `scanf` and `cin`

D. All of them

8. `int a = 100000, b = 100000; long long c = a * b;` — what is `c`?

A. 10^{10}

B. Undefined behaviour: `a * b` is computed in int and overflows before it is converted

C. 0

D. A compile error

9. Which of these is about 10^8?

A. The simple operations a judge runs in roughly one second

B. The largest int

C. The number of bytes in a 256 MB memory limit

D. The largest n any problem allows

10. The time limit is 2 seconds and $n \le 10^5$. Your idea is $O(n \log^2 n)$. Is it worth coding?

A. No, $\log^2$ is too slow

B. Yes — $10^5 \cdot 17^2 \approx 2.9 \cdot 10^7$, well inside the budget

C. Only if $n \le 10^4$

D. Only with a faster language

11. Why is `long long` the default in the template rather than `int`?

A. It is faster

B. Sums, products and counts in contest problems routinely pass $2.1 \cdot 10^9$; a wrong answer from overflow costs far more than the few bytes

C. `int` is not allowed on judges

D. Because `cin` cannot read ints

ANSWER KEY · CHECKPOINT — FAST I/O AND COMPLEXITY BUDGETS

1. B $2^{20} \cdot 20 \approx 2.1 \cdot 10^7$ fits. $20! \approx 2.4 \cdot 10^{18}$ and $20^4 \cdot 2^{20} \approx 1.7 \cdot 10^{11}$ do not. n around 20 is the classic sign of a bitmask or subset search.

2. B $5000^2 = 2.5 \cdot 10^7$ fits; $5000^3 = 1.25 \cdot 10^{11}$ is about a thousand times over.

3. C $10^6 \cdot \log_2 10^6 \approx 2 \cdot 10^7$ fits. $n\sqrt{n} = 10^9$ and $n^2 = 10^{12}$ do not.

4. C $\sqrt{(10^{18})} = 10^9$ is already ten times over the budget. Only O(log n) or a formula works — binary search, fast exponentiation, digit-by-digit counting.

5. A $400^3 = 6.4 \cdot 10^7$ fits; $400^4 = 2.56 \cdot 10^{10}$ does not; 2^{400} is astronomically large. A few hundred usually means $O(n^3)$ — Floyd–Warshall or an interval DP.

6. A In a one-file solution it saves typing; in a header it pulls every `std` name into every file that includes it, where it can collide with your own names.

7. B With synchronisation off, the C++ streams and C's stdio keep separate buffers, so interleaving them can print or read out of order. Pick one family.

8. B The type of the destination does not change the type of the multiplication. Write `1LL * a * b` or make `a` and `b` `long long`.

9. A It is a rule of thumb, not a constant — a tight loop of additions does more, cache misses or divisions fewer — but it is the right order of magnitude to budget with. The largest int is about $2.1 \cdot 10^9$, and 256 MB is about $2.7 \cdot 10^8$ bytes.

10. B $\log_2 10^5 \approx 17$, so the extra log factor costs about 17×. $2.9 \cdot 10^7$ operations is comfortably within two seconds.

11. B Counting pairs of 10^5 elements already gives $5 \cdot 10^9$. Use `long long` for values and totals, and keep `int` for indexes and loop counters.