Module 4 of 12

Binary search on the answer

Minimise the maximum, maximise the minimum: write a yes/no check for one guess, prove it monotone, and let about sixty halvings find the best value — with long long ranges and checks that cannot overflow.

Some contest problems ask for the best value directly — the smallest possible largest part, the largest possible smallest gap, the earliest time something can be done — and computing it head-on is hard. Checking one guess is easy. When the answers to "does X work?" are monotone (no, no, …, yes, yes), binary search over the *range of answers* finds the boundary in $\log_2(\text{range})$ checks: thirty for 10^9, sixty for 10^{18}.

The lesson shows the range halving on a real split, gives the two templates (first yes and last yes, and why `mid` rounds up in the second), and names the traps that fail solutions at the limits: sums in `int`, unprovable bounds and a check that overflows before it answers. The exercises are the two classics — split an array into k parts and aggressive cows — with hidden cases at 4 000 values up to 10^9. The timed checkpoint is a factory of machines where the answer reaches 10^{18} and the check must stop counting before `long long` runs out.

CONTENTS

1	Binary search on the answer — guess, check, halve	14 min
✓	Checkpoint — Binary search on the answer	45 min

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

LESSON 1 · 14 MIN

Binary search on the answer — guess, check, halve

Some problems ask for the best value directly — "the smallest largest part", "the largest smallest gap" — and computing it is hard. Checking a *guess* is easy. Binary search on the answer turns the hard question into about sixty easy ones.

The shape to look for

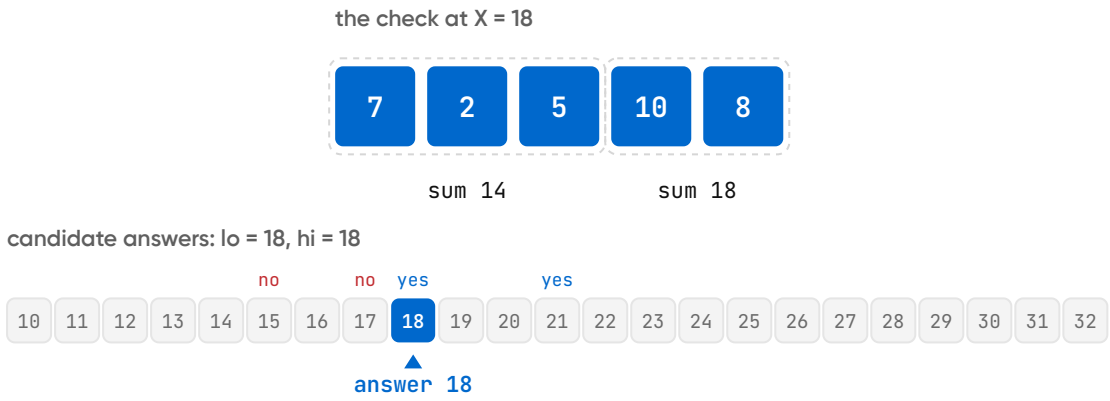
- The statement says **minimise the maximum** or **maximise the minimum**.
- For a fixed guess `X` you can answer **"is X achievable?"** quickly, usually greedily in O(n).
- The answers are **monotone**: once `X` works, every bigger `X` works too (or every smaller one, for a maximum). The row of answers reads no, no, …, no, yes, yes, …, yes.

That row is a sorted array of booleans, and binary search finds its first yes in $\log_2$(range) steps.

Watch it halve

Split `[7, 2, 5, 10, 8]` into `k = 2` contiguous parts so the largest part sum is as small as possible. The check for a limit `X`: walk left to right, start a new part whenever the next number would push the current one past `X`, and count the parts.

Binary search on the answer: the range of candidates halves



1. No split can beat the largest element, 10, and one part holding everything sums to 32. So the answer lies in [10, 32]: 23 candidates, and the search will check only a handful.
2. mid = 21: the sweep makes 2 parts (sums 14 and 18), within k = 2. 21 works, so the answer is 21 or smaller: hi = 21.
3. mid = 15: the sweep needs 3 parts (sums 14 and 10 and 8), more than k = 2. 15 fails, and so does everything below it: lo = 16.
4. mid = 18: the sweep makes 2 parts (sums 14 and 18), within k = 2. 18 works, so the answer is 18 or smaller: hi = 18.
5. mid = 17: the sweep needs 3 parts (sums 14 and 10 and 8), more than k = 2. 17 fails, and so does everything below it: lo = 18.
6. lo and hi meet at 18: 4 checks instead of 23. The split [7, 2, 5] + [10, 8] has largest sum 18, and no split does better.

The answer can never be below the largest element (10) or above the total (32), so those are the bounds. Every check throws away half of what is left.

The template

```
ll lo = *max_element(a.begin(), a.end());      // certainly the floor
ll hi = accumulate(a.begin(), a.end(), 0LL);   // certainly feasible
while (lo < hi) {
    ll mid = lo + (hi - lo) / 2;               // rounds down
    if (ok(mid)) hi = mid;                     // mid works: the answer is mi
    else lo = mid + 1;                         // mid fails: the answer is ri
}
// lo == hi == the first X with ok(X)
```

- **Minimise** (first yes): `hi = mid` / `lo = mid + 1`, `mid` rounds down.
- **Maximise** (last yes): `lo = mid` / `hi = mid - 1`, `mid = lo + (hi - lo + 1) / 2` rounds *up* — or the loop sticks on a two-value range.
- `lo + (hi - lo) / 2` never overflows; `(lo + hi) / 2` can when both are near 10^18.

The traps

- **Sums in `int`.** A part sum of 10^5 values up to 10^9 is 10^14. Use `long long` for the range and the check.
- **A check that overflows.** Counting "products made by time T" adds up to n values of `T / t`, each up to 10^18. Stop as soon as the count reaches the target.
- **Wrong bounds.** If `hi` might be infeasible, the search returns `hi` anyway. Prove both bounds from the constraints.

GAME · QUICK FIRE

Which way does the range move?

For each check result, pick the update that keeps the answer inside [lo, hi].

1/5

Minimise the largest part. ok(mid) is true.

A hi = mid

B lo = mid + 1

C hi = mid - 1

More practice

- Koko Eating Bananas — minimise a speed.
- Capacity To Ship Packages Within D Days — the lesson's check, by days.
- Split Array Largest Sum — the figure's problem.
- Magnetic Force Between Two Balls — maximise the minimum.
- Minimum Number of Days to Make m Bouquets — search over days.

EXERCISE 1

Split the array into k parts

Split an array of `n` positive integers into `k` non-empty **contiguous** parts so that the largest part sum is as small as possible. Print that smallest possible largest sum.

Input: `n k` (`1 ≤ k ≤ n ≤ 10^5`), then `n` integers `a_i` (`1 ≤ a_i ≤ 10^9`). **Output:** one integer.

```
5 2
7 2 5 10 8
```

prints

```
18
```

`[7, 2, 5]` and `[10, 8]` have sums 14 and 18; no split has a smaller maximum. Sums reach 10^{14}, and the hidden cases are at the limits.

INPUT 1

```
5 2
7 2 5 10 8
```

OUTPUT 1

```
18
```

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, k;
    cin >> n >> k;
    vector<ll> a(n);
    for (auto& x : a) cin >> x;

    // Parts needed when no part may exceed limit (limit ⩾ every element).
    auto parts = [&](ll limit) {
        int count = 1;
        ll sum = 0;
        for (ll x : a) {
            if (sum + x > limit) {
                ++count;
                sum = 0;
            }
            sum += x;
        }
        return count;
    };

    ll lo = *max_element(a.begin(), a.end());
    ll hi = accumulate(a.begin(), a.end(), 0LL);
    while (lo < hi) {
        ll mid = lo + (hi - lo) / 2;
        if (parts(mid) ⩽ k) hi = mid;
        else lo = mid + 1;
    }
    cout << lo << '\n';
    return 0;
}
```

EXERCISE 2

Aggressive cows

There are `n` stalls at distinct positions on a line and `c` cows. Put the cows in `c` different stalls so that the **smallest distance between any two cows** is as large as possible. Print that distance.

Input: `n c` (`2 ≤ c ≤ n ≤ 10^5`), then `n` distinct positions `x_i` (`0 ≤ x_i ≤ 10^9`), in any order.
Output: one integer.

```
5 3
1 2 8 4 9
```

prints

```
3
```

Cows at 1, 4 and 8 (or 9) are at least 3 apart; no placement does better.

INPUT 1

```
5 3
1 2 8 4 9
```

OUTPUT 1

```
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);
    int n, c;
    cin >> n >> c;
    vector<ll> x(n);
    for (auto& v : x) cin >> v;
    sort(x.begin(), x.end());

    // Can c cows stand at least d apart? Greedy: first stall, then the next one far
    auto fits = [&](ll d) {
        int placed = 1;
        ll last = x[0];
        for (int i = 1; i < n && placed < c; ++i) {
            if (x[i] - last >= d) {
                ++placed;
                last = x[i];
            }
        }
        return placed >= c;
    };

    ll lo = 0, hi = x.back() - x.front();
    while (lo < hi) {
        ll mid = lo + (hi - lo + 1) / 2;   // round up: the update is lo = mid
        if (fits(mid)) lo = mid;
        else hi = mid - 1;
    }
    cout << lo << '\n';
    return 0;
}
```

QUIZ

1. Which property must the check `ok(X)` have for binary search on the answer to be correct when minimising X?

A. ok(X) must run in O(1)

B. ok(X) must be true for exactly one X

C. The input array must be sorted

D. If ok(X) is true then ok(Y) is true for every Y > X

2. Searching for the **last** X with `ok(X)` true, the loop is `while (lo < hi) { mid = lo + (hi - lo) / 2; if (ok(mid)) lo = mid; else hi = mid - 1; }`. With `lo = 4, hi = 5` and `ok(5)` true, what happens?

A. It loops forever

B. It returns 5

C. It returns 4

D. It reads out of bounds

3. Splitting `n = 10^5` values, each up to `10^9`, into k parts: why must `lo`, `hi`, `mid` and the running sum be `long long`?

A. Because `accumulate` only accepts `long long`

B. The total can reach 10^14, far past int's 2.1 × 10^9

C. Binary search needs 64-bit indexes

D. `long long` arithmetic is faster than `int`

4. For a minimise-the-maximum search over answers up to 10^18 with an O(n) check and n = 2 × 10^5, roughly how many operations does the whole search cost?

A. About 10^18

B. About 4 × 10^10

C. About 1.2 × 10^7

D. About 2 × 10^5

ANSWER KEY · BINARY SEARCH ON THE ANSWER — GUESS, CHECK, HALVE

1. D Monotonicity is what turns the answers into a sorted row of booleans (no … no yes … yes), which is the shape binary search needs. The check may take O(n) — that is the usual case — and the input itself need not be sorted.

2. A mid rounds down to 4; if ok(4) is true, lo = mid = 4 and nothing changed, so the loop never ends. With `lo = mid` the midpoint must round up: `lo + (hi - lo + 1) / 2` gives 5.

3. B The upper bound is the total sum, up to 10^5 × 10^9 = 10^14. In int it overflows (undefined behaviour). Note that `accumulate(a.begin(), a.end(), 0)` would also sum in int — the initial value decides the type, hence `0LL`.

4. C log2(10^18) ≈ 60 checks, each O(n) = 2 × 10^5, so about 1.2 × 10^7 — comfortably inside a one-second limit.

CHECKPOINT · 45 MIN

Checkpoint — Binary search on the answer

A timed checkpoint on binary search on the answer: recognising the shape, proving the bounds, the two loop templates and the overflow traps.

How it works. Ten questions and one program, in **45 minutes** from when you start the timer. Clear 70% of the questions and get the program accepted on every hidden case — including answers near 10^{18} — to complete the module.

Before you start, make sure you can answer these from memory:

- What makes a yes/no check monotone, and why that is what binary search needs.
- The first-yes template (`hi = mid`, `lo = mid + 1`) and the last-yes template (`lo = mid`, `hi = mid - 1`, `mid` rounded up).
- How to prove `lo` and `hi` from the constraints.
- Why a sum inside the check can overflow `long long`, and how to stop it.
- How many checks a range of 10^9 or 10^{18} takes.

EXERCISE

Factory machines

A factory has `n` machines. Machine `i` makes one product every `k_i` seconds, and all machines work at the same time. What is the shortest time in which they can make `t` products together?

Input: `n t` (`1 ≤ n ≤ 2·10^5`, `1 ≤ t ≤ 10^9`), then `n` integers `k_i` (`1 ≤ k_i ≤ 10^9`). **Output:** one integer, the minimum time in seconds.

```
3 7
3 2 5
```

prints

```
8
```

In 8 seconds the machines make 2 + 4 + 1 = 7 products; in 7 seconds only 2 + 3 + 1 = 6. The answer can reach 10^{18} — and adding up `T / k_i` over every machine can overflow even `long long`.

INPUT 1

```
3 7
3 2 5
```

OUTPUT 1

```
8
```

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;
    ll t;
    cin >> n >> t;
    vector<ll> k(n);
    for (auto& x : k) cin >> x;

    // Products made in T seconds reach t? Stop as soon as they do: the sum of
    // T / k_i over 2·10^5 machines could pass the long long range otherwise.
    auto enough = [&](ll T) {
        ll made = 0;
        for (ll x : k) {
            made += T / x;
            if (made >= t) return true;
        }
        return false;
    };

    ll lo = 1;
    ll hi = *min_element(k.begin(), k.end()) * t;   // the fastest machine alone: at
    while (lo < hi) {
        ll mid = lo + (hi - lo) / 2;
        if (enough(mid)) hi = mid;
        else lo = mid + 1;
    }
    cout << lo << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. Which statement wording most strongly suggests binary search on the answer?

A. Count the subarrays with sum exactly k

B. Find the shortest path between two cities

C. Print the array in sorted order

D. Minimise the maximum load over all workers

2. In the factory problem, why is `hi = min(k) * t` a correct upper bound?

A. The fastest machine alone makes t products in that time, so the answer is no larger

B. It is the sum of all k_i

C. It is the largest value long long can hold

D. Because every machine must make t products

3. Why does the factory check return as soon as the count reaches `t`?

A. Because the judge counts loop iterations

B. Adding T / k_i for every machine can overflow long long when T is near 10^18

C. It is required for correctness of binary search in general

D. To make the predicate monotone

4. For aggressive cows (maximise the smallest gap), which update pair is right?

A. fits(mid) → lo = mid + 1; otherwise hi = mid

B. fits(mid) → hi = mid - 1; otherwise lo = mid

C. fits(mid) → lo = mid; otherwise hi = mid - 1, with mid rounded up

D. fits(mid) → hi = mid; otherwise lo = mid + 1, with mid rounded down

5. What does `lo + (hi - lo) / 2` avoid that `(lo + hi) / 2` does not?

A. Rounding towards zero

B. An infinite loop when hi = lo + 1

C. Negative midpoints

D. Overflow when lo and hi are both close to the type's maximum

6. A range of candidate answers is 1 … 10^9. How many checks does the search make, at most?

A. About 30

B. About 10

C. About 1000

D. About 10^9

7. Splitting `[7, 2, 5, 10, 8]` into 2 parts, the search checks `X = 15`. How many parts does the greedy sweep make, and what does the search do?

A. 4 parts, so lo = 16

B. 3 parts, so lo = 16

C. 2 parts, so hi = 15

D. 3 parts, so hi = 15

8. Why may the split check count "at most k parts" rather than "exactly k"?

A. Because binary search ignores the exact count

B. It may not — it must be exactly k

C. A split into fewer parts can always be cut further without raising the maximum, as long as k ≤ n

D. Because the judge accepts fewer parts

9. A check `ok(X)` takes O(n log n) because it sorts. With n = 10^5 and answers up to 10^9, about how much work is the search?

A. About 10^5

B. About 10^14

C. About 1.7 × 10^6

D. About 30 × 1.7 × 10^6 ≈ 5 × 10^7

10. Which problem is **not** solved by binary search on the answer?

A. Count how many pairs in an array sum to k

B. Koko's minimum eating speed to finish in h hours

C. Ship packages within D days with the least capacity

D. Place balls in baskets to maximise the minimum magnetic force

ANSWER KEY · CHECKPOINT — BINARY SEARCH ON THE ANSWER

1. D "Minimise the maximum" (and "maximise the minimum") is the signature: a guessed bound is easy to check and the answers are monotone in it.

2. A The answer is the earliest time all machines together reach t. The fastest one by itself reaches t at min(k) · t, and the others can only help, so that time is feasible — exactly what hi must be.

3. B With T up to 10^18 and k_i = 1, a single term is 10^18 and two of them already pass 9.2 × 10^18. Stopping once the target is met keeps the running total below 2 × 10^18. Monotonicity holds either way.

4. C A gap that fits means the answer is at least mid, so keep mid (lo = mid); a gap that does not fit rules mid out (hi = mid - 1). Keeping mid on the lo side needs mid rounded up.

5. D `lo + hi` can exceed the type's range when both are near 10^18; `hi - lo` never does. It still rounds down, so it does not fix an infinite loop caused by `lo = mid`.

6. A Each check halves the range: log2(10^9) ≈ 29.9, so 30 checks.

7. B 7 + 2 + 5 = 14, then 10 would make 24 > 15, so a new part [10]; 10 + 8 = 18 > 15, a third part [8]. 3 > 2, so 15 fails and lo becomes 16.

8. C Cutting a part into two never increases any part's sum, so any split into fewer than k parts can be refined to exactly k with the same or smaller maximum.

9. D 30 checks × n log n ≈ 30 × 10^5 × 17. If the sort does not depend on X, do it once outside the search and the check becomes O(n).

10. A Counting pairs has no answer to guess and no monotone yes/no check; it is a hashing or two-pointer problem. The other three are classic minimise-the-max or maximise-the-min searches.