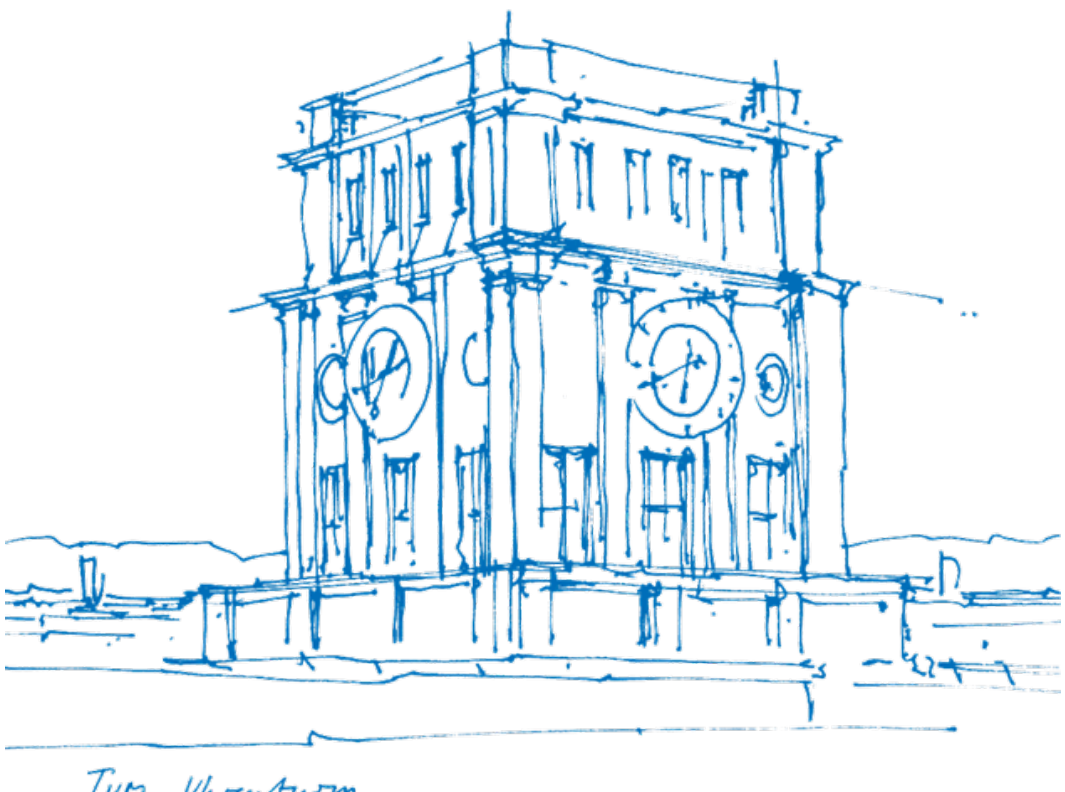
School of Computation, Information and Technology
Technical University of Munich

AACPP 2025

Week 2: Tackling a Problem



Mateusz Gienieczko, **Mykola Morozov**

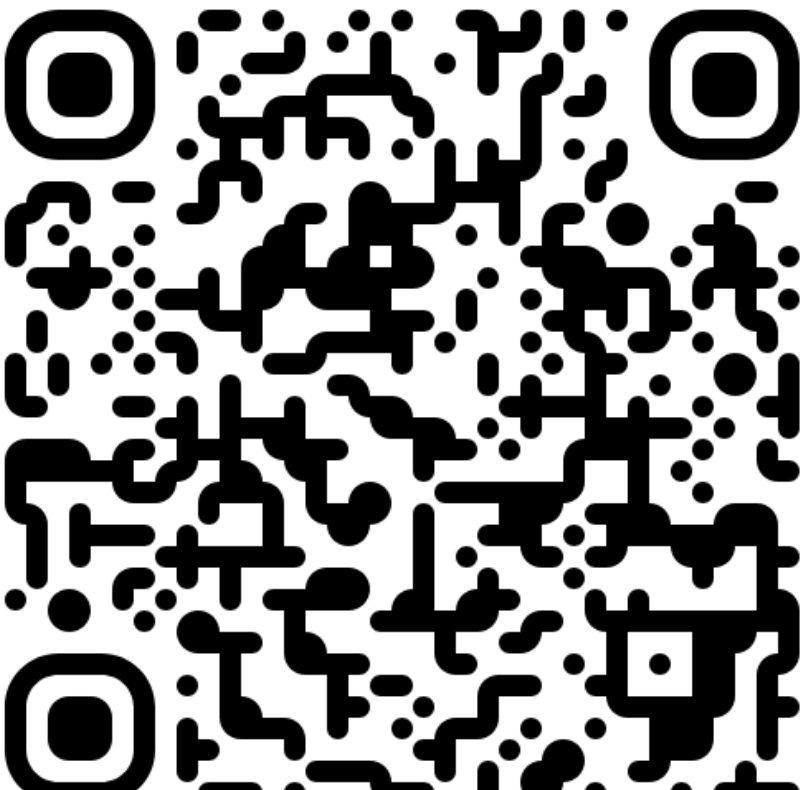
School of Computation, Information and Technology
Technical University of Munich

2025.05.06

First two problems

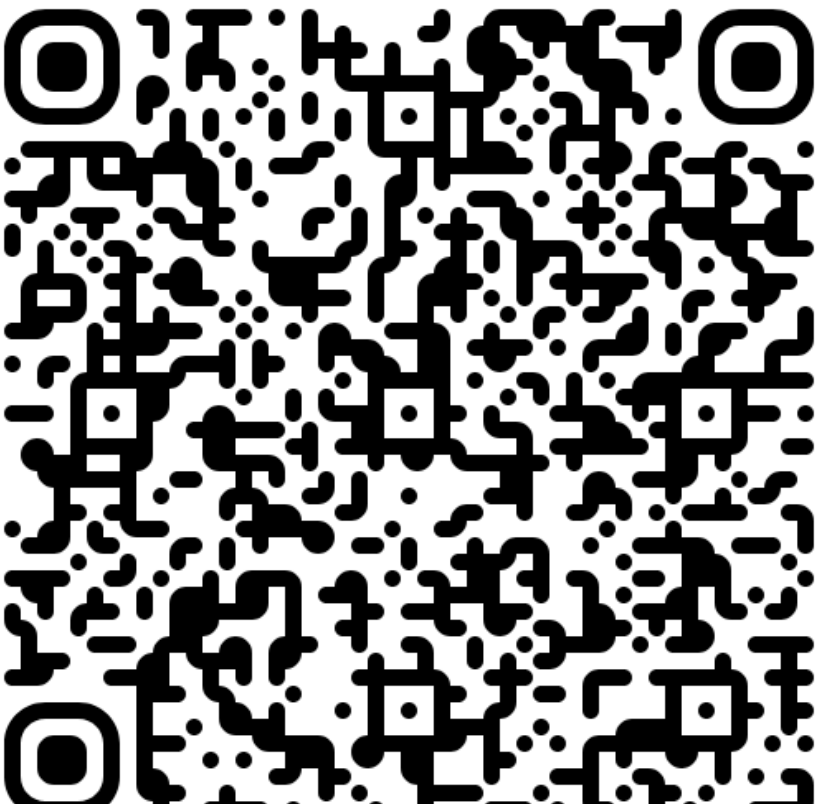
First deadline – 13.05.2025, 10:00 AM.

Try to get non-zero number of points :)



Mattermost reminder

Official Mattermost



Theses! B.Sc.! M.Sc.!

We do databases and data processing.

Fast.

C++ or Rust.

https://db.in.tum.de/research

https://db.in.tum.de/people/sites/gienieczko/

https://db.in.tum.de/people/sites/morozov/

The Hard Part

- Greedy and dynamic programming (DP)
- Trees
- Graphs
- Ways to turn graphs into trees (DFS, BFS, Dijkstra, MST)
- Ways to run DP on graphs (Toposort)
- Advanced graph algorithms (Matchings, flows)
- Binary Search Trees
- Number theory
- String algorithms (KMP, tries, suffix tables)

Some problems can’t even be solved efficiently (NP-completeness)

The Hard Part

How to decide what tools to use?

Example: XAP

Task: XAP
Experimental Assault Pigeons

AACPP SuSe 2024 Round 2 Memory: 32MiB (Java: 128MiB) 2024.05.28 – 2024.06.11

After helping with the Universal Signalling System, Byteman has been promoted higher in the ranks of Byteland's IT Task Force and is now overseeing the computerised mission control system for the Bytelandian Air Force. The hottest innovation in aerial operations? *Pigeons*.

BAF wants to use specially trained pigeons for its experimental Rapid Aerial Payload Delivery programme. In theory they could be used to deliver messages, electronic interference devices, small explosives, or smaller military animals[1], while remaining undetected in enemy territory. This "undetected" part is Byteman's assignment right now – he needs to help XAP units evade radar.

Byteman has access to a training area simulating the territory to inflitrate. From a bird's eye view, the area is a rectangular 2D grid divided into three areas along the X axis. The top and bottom area spanning y values $[0, 100]$ and $[w + 100, w + 200]$ are radar arrays – there are

Example: XAP

n radars in total, and each has a circular detection radius. The edge of the detection range is still dangerous and can detect a pigeon. The flight path is restricted to the middle area of width w. The pigeons start at $x = 0$ and any $y \in (100, w + 100)$, while its goal is to reach l on the X axis. The third dimension does not matter – the radars' operating range is much higher than the maximum altitude of any pigeon.

Fortunately, unlike regular pigeons, the XAP units are smart and agile fliers. They can turn sharp angles in an instant to evade radar detection – their trajectory forms an arbitrary polygonal chain. The catch is that a well-designed radar network might make navigating through the entire territory *impossible*. XAP units might require preliminary strikes – a number of radars to be taken down before the operation begins. To make operations cost-effective, though, the BAF wants to *minimise* the number of destroyed radars.

Example: XAP

Input

In the first line of standard input there are three integers n, w, l, in order: the number of radars, the width of the middle area, and the target x coordinate.

The next n lines contain the description of the radar system. In the i-th line there are three integers, x_i, y_i, r_i, describing the coordinates of the i-th radar and its detection radius, respectively.

There is always at least two radars, one in each array. The x coordinates are always between 0 and l, whereas the y coordinates are always in $[0, 100] \cup [w + 100, w + 200]$.

Output

Your program should output two lines. The first line should contain one integer $0 \leq k \leq n$ – the minimal number of radars that have to be removed to clear a valid flight path.

In the next line there should be k unique integers between 1 and n, denoting which of the radars need to be destroyed, in ascending order. If $k = 0$ then the line must be left blank.

While the minimal k is well defined, there might be more than one correct set of radars of size k that can be destroyed. Your program may output any of them.

Example: XAP

Example

For the input:

```
7 200 700
175 88 100
362 44 200
548 88 125
100 312 62
294 312 106
456 308 127
553 326 88
```

one of the correct outputs is:

```
2
2 6
```

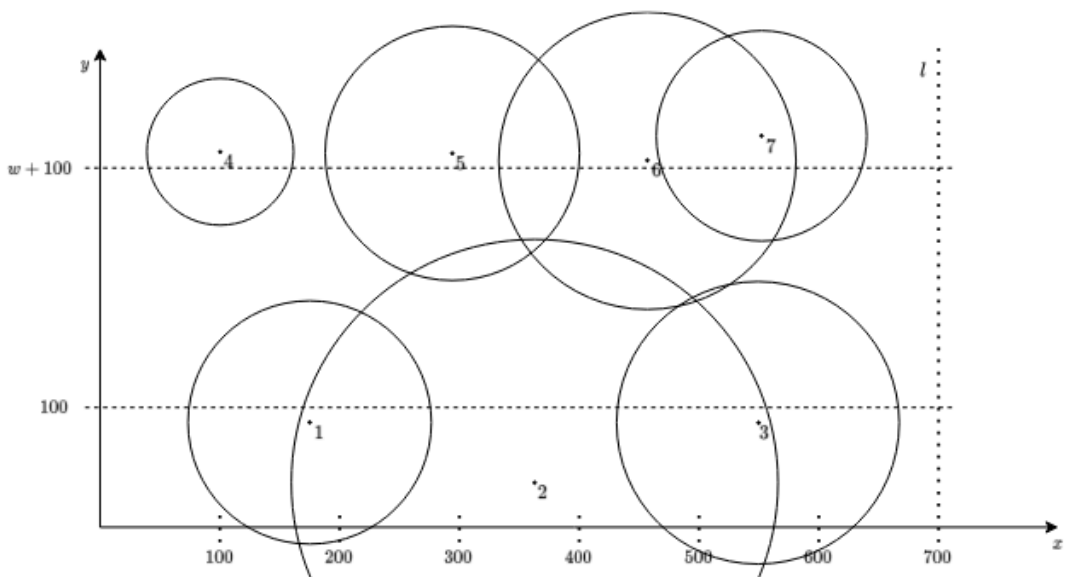


Figure 1: Test setup with 3 radars in the lower and 4 in the upper area. There is no clear path through the radars.

Example: XAP

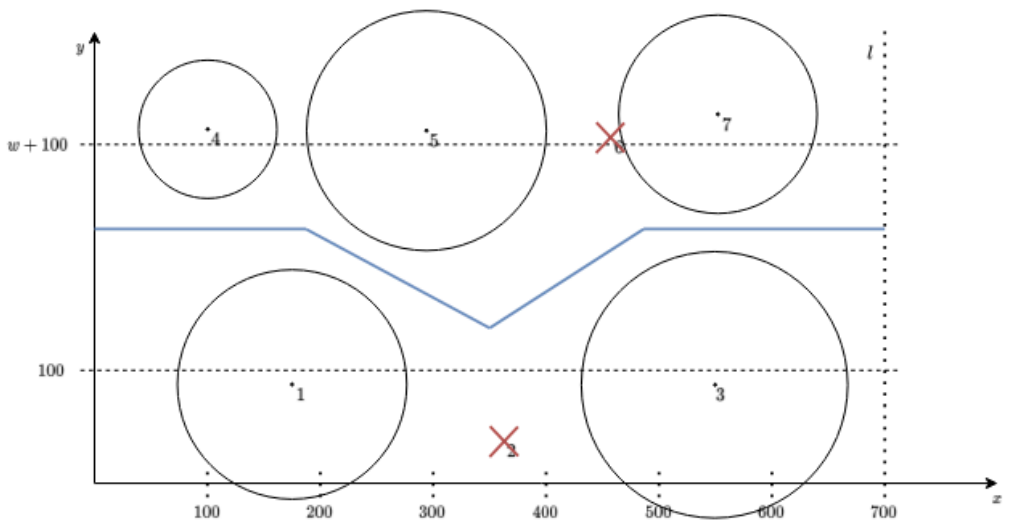


Figure 2: Two destroyed radars (2 and 6) clear the flight path (in blue).

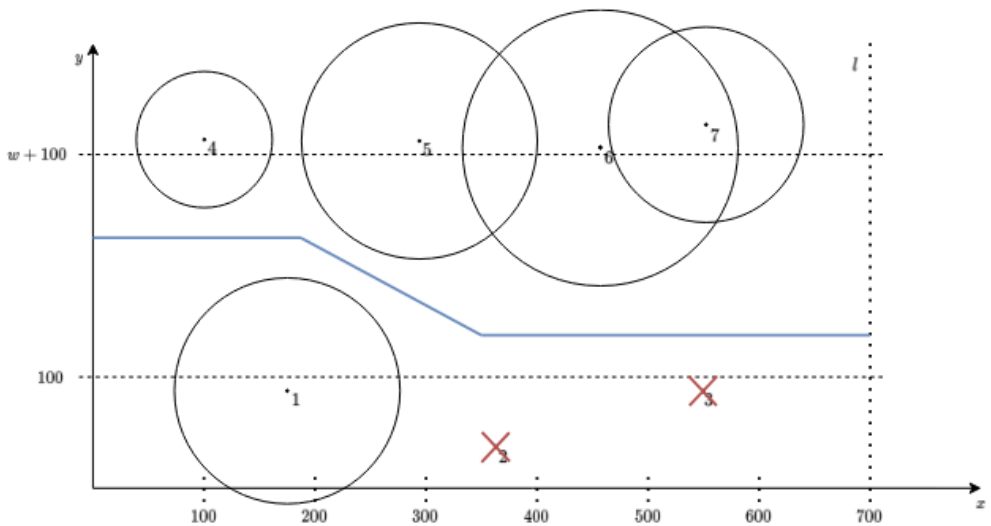


Figure 3: Alternative correct solution where radars 2 and 3 are destroyed.

Example: XAP

Limits

Your solution will be evaluated on a number of hidden test cases divided into groups. Points for a group are awarded if and only if the submission returns the correct answer for each of the tests in the group within the allotted time limit. These groups are organised into subtasks with the following limits and points awarded.

In all tests each radar radius is limited by $10{,}000$.

Partial points

If your solution outputs the correct number of radars (first line of output), and the second line is left blank or not correct, it will receive 50% of the points for a given test group.

Subtask	Limits	Points
1.	$2 \leq n \leq 20$, $1 \leq w \leq 800$, $1 \leq l \leq 1{,}000$	2
2.	$2 \leq n \leq 40$, $1 \leq w \leq 800$, $1 \leq l \leq 4{,}000$	2
3.	$2 \leq n \leq 1{,}000$, $1 \leq w \leq 800$, $1 \leq l \leq 25{,}000$	4
4.	$2 \leq n \leq 5{,}000$, $1 \leq w \leq 2{,}500$, $1 \leq l \leq 40{,}000$	2

Simplify

What if radars are only on one side?

Simplify

What if radars are only on one side?

When are they on both sides but answer is 0?

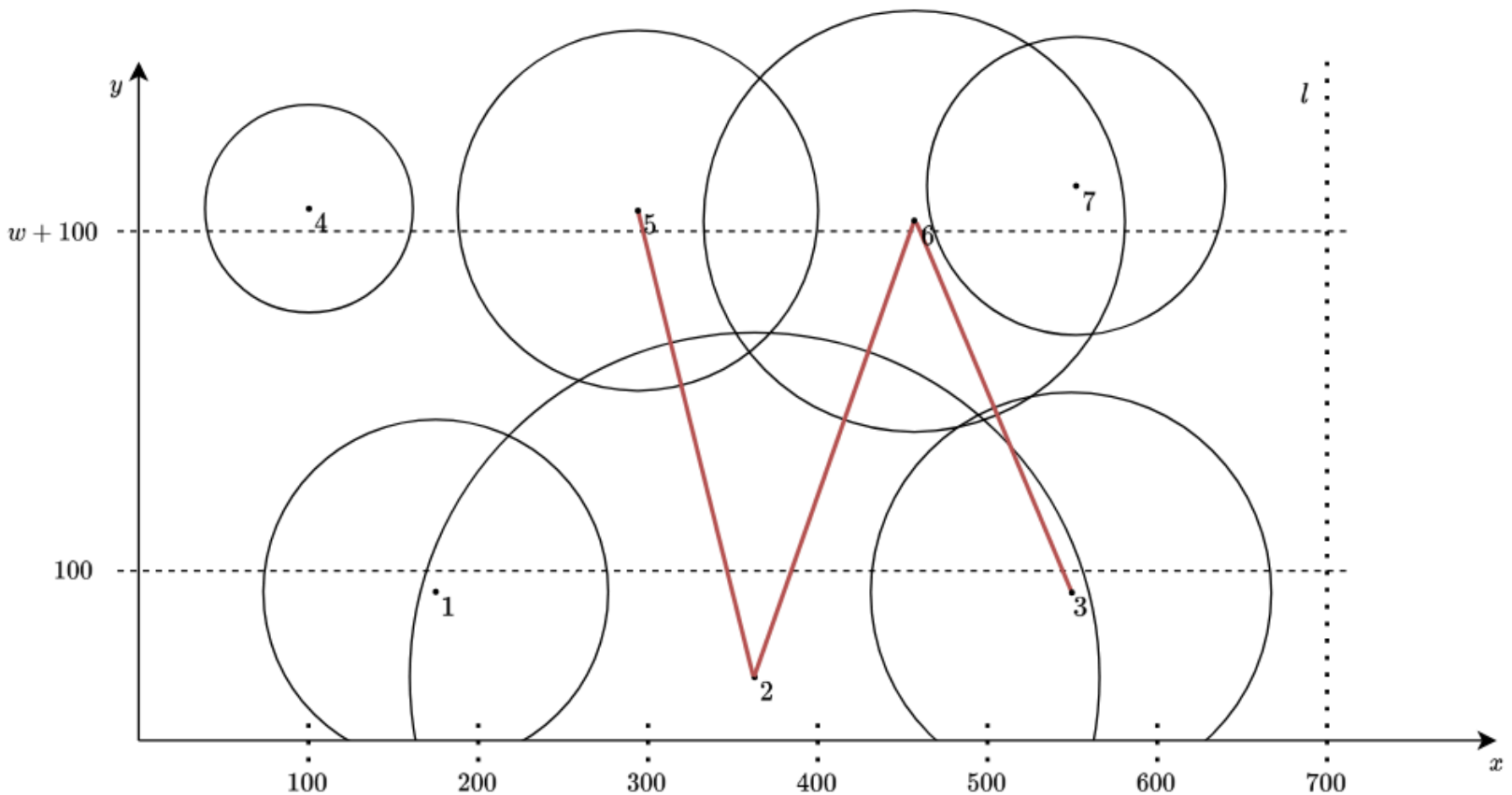
Simplify

What if radars are only on one side?

When are they on both sides but answer is 0?

Simplest case where answer is 1?

Formal characterisation



Formal characterisation

Remove minimal number of vertices to remove all edges.

Minimal Vertex Cover in a bipartite graph.

Oh no it doesn’t work

If it doesn’t work on example or your manual tests – good!

What if we don’t have a counterexample?

Proving correctness

Sometimes solution doesn’t work fundamentally.

Sometimes it’s a coding bug.

Hard to find a bug if you’re not sure that your solution is supposed to work.

Usual approach – vibes well means it works

Quite useful for competitive programming where time is limited.

Dangerous, greedy solutions often vibe well but might not work.

At least try to prove it to yourself

Often you'll talk yourself into a counterexample.

... or maybe it actually works conceptually and then you need to debug.

Generating tests

Random tests are usually poor quality.

Generating tests

Random tests are usually poor quality.

BUT

We can generate *a lot* of them.

Output?

Sometimes it’s possible to generate a test for a given output, but not always.

Brute-force solutions to compare against.

XAP brute force

Just select a subset to circles to remove and check if it works.

Go over all subsets.

$\mathcal{O}\left(2^n n^2\right)$, but it works for subtask 1.

Generate tests with $n \leq 20$.

Debugging

USE A DEBUGGER.

If you’re already using one, good, keep doing so.

Debugging

USE A DEBUGGER.

If you’re already using one, good, keep doing so.

If not and you like vim you will also like gdb.

Debugging

USE A DEBUGGER.

If you’re already using one, good, keep doing so.

If not and you like vim you will also like gdb.

Otherwise, use an IDE. CLion is really good.

Printf debugging

Slapping some print statements to print variables and overall flow of the program.

Actually pretty effective in competitive programming.

Real Life

Coding solutions for competitive programming is different than actual coding.

Real Life

Coding solutions for competitive programming is different than actual coding.

BUT

it **is** very educational!

Testing

Random tests are generally bad, but **fuzzing** is an important technique.

Widely used in production software.

Debugging

Duh.

Debugging

Duh.

Print debugging as well, we just call it “structured logging” to sound professional.

Complexity

No one will ask you to prove complexity at a Real Job™...

Complexity

No one will ask you to prove complexity at a Real Job™...

BUT

they will ask “will this work if we have a million entries”.

a treat tastier than the previous one

Problem solving

We have a giant toolbox at our disposal and a problem to solve.

Usually not as well defined, but translating requirements to something sensible is similar.

The main difference is there is no judging system.

Problem solving

We have a giant toolbox at our disposal and a problem to solve.

Usually not as well defined, but translating requirements to something sensible is similar.

The main difference is there is no judging system.

Try not to solve tasks by a thousand resubmissions, for your own sake.

Various tips and tricks

C++: Don't use `<iostream>`, or at least `sync_with_stdio(false)`.

Various tips and tricks

When reading to a `Vec` or `std::vector`, reserve the capacity first.

Rust:

```rust
let v = Vec::with_capacity(n);
```

C++:

```cpp
std::vector<SomeType> v{};
v.reserve(n);
```

Various tips and tricks

Remember the relative performance of operations.

Various tips and tricks

Remember the relative performance of operations.

On our 32-bit judging platforms `int32` ops are faster than `int64`.

Floating-point operations are slow, usually can be avoided.

Ops on `double` are **much** slower than on `float`.

Division and modulus is slower than multiplication, which is slower than addition.

Various tips and tricks

Standard library functions are usually much faster than your own implementations.

As true here as it is in Real Life™.

Also less buggy :)

Basic algorithms and data structures

Sorting.

Stacks, queues, deques.

Heaps (priority queues).

Divide and conquer.

“Two pointers” or “the caterpillar”.

Divide and conquer general scheme

Divide a big instance into smaller subproblems.

Solve subproblems independently.

Combine the results.

E.g. merge sort.

Divide and conquer example

Task

Dexter received a new treat in form of a stick divided into n segments. Some parts are tastier than others and some are not tasty at all. Dexter wants to eat the tastiest bits first, but he can only eat segments adjacent to each other. Help him decide which segments to eat to maximise tastiness.

Example

```
7
5 -7 2 4 -1 6 -3

11
```

Divide and conquer example

Divide the array in “half”.

The optimal result is either

- fully in the left part,
- fully in the right part,
- between the parts.

We obtain the first two via recursion, and the other can be computed in $\mathcal{O}(n)$.

Base case of $n = 1$ is trivial.

Divide and conquer example

`left = [5, -7, 2], right = [4, -1, 6, -3]`

Optimal in `left` is 5, optimal in right is 9.

Compute prefix sums in `right` and suffix sums in `left`:

`lefts = [0, -5, 2], rights = [4, 3, 9, 6]`

Best choice for in-between solution is 11.

Time complexity

At every level we do $\mathcal{O}(n)$ work.

Every recursion divides the size in half.

$$n + \frac{n}{2} + \frac{n}{2} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{8} \ldots = \Theta(n \log n)$$

Common D&C: binary search

Basic case: we have a sorted array and want to find an element x.

`[3, 7, 7, 9, 11, 23, 100]`

Check the middle element – if greater than x then x can only be in the left half. Otherwise, in the right.

Standard libraries have implementations already, `slice::binary_search` in Rust and `std::lower_bound`/`std::upper_bound` in C++.

Generic binary-search

Binary search finds the critical point of any *binary monotonic function*.

The general property of: *If element at index i satisfies the condition, then the one at $i + 1$ also does.*

Binary-search example

Dexter found a cat lottery! There are n prizes, where i-th prize is a batch of i treats. To enrol Dexter needs to buy (with his own treats) 1 *lot for the first prize,* 2 *lots for the second price, etc., up to n lots for the n-th prize. Each lot has equal probability of winning, but to get treat i you need* all *i lots to win. Given value a_i of each of the treats and the number b of treats Dexter has, help him decide what is the minimum probability (with 10^{-6} accuracy) of a lot winning that makes entering worth it.*

```
3 3
1 5 10

0.5
```

Binary-search example

For a given probability p the expected payoff $E(p)$ is:

$$\sum_{1 \le i \le n} a_i x^i$$

This has to be at most b.

This condition is monotonic: $E(p) > b \Rightarrow E(p + \varepsilon) > b$.

Binary-search for the probability after adjusting to integers.

Two pointers / Caterpillar

Go through a sequence S keeping two pointers i, j, $j \geq i$.

The currently considered subsequence is $S[i..j]$.

Both pointers advance forward.

Common pattern in solutions.

Two pointers example

Dexter the Cat wants to open an account at Fressnapf to order his own treats on Mat's credit card he found on the table. While he is very smart, his little paws are made to maximise cuteness, not to use a keyboard. To create a password he stomped around and got a long and strong password but, of course, the site has some silly requirements on "complexity". They require that each prefix of the password has at most k more capital letters than small letters. *Dexter can now remove some letters from the beginning and the end, but not the middle. Help Dexter obtain the longest valid password and get his treats!*

```
2
BBcDEXlolDEX

9
```

Two pointers example

We maintain a subsequence that is valid, starting with $i = j = 0$.

Try to extend by moving j forward.

If condition is violated, move i forward until it's not.

Maintaining the count difference is easy.

Since both pointers move at most n times we have $\mathcal{O}(n)$.

See you next week

ZOO and TOY: 13.05.2025,
10:00 AM

Good luck!