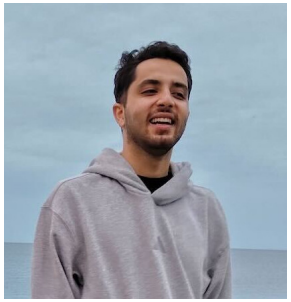
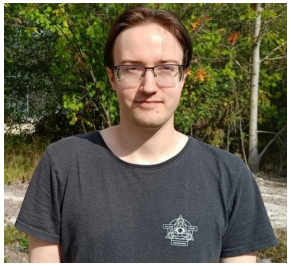
# HIIT Open 2024

November 16, 2024

# Organizers



Amirreza Akbari



Juha Harviainen

Tuukka Korhonen

Antti Laaksonen

Henrik Lievonen

Jukka Suomela

# Statistics

- 19 teams, 47 contestants
- 327 submissions, 92 accepted
- Each team solved at least one problem
- First accepted solution at 11 last sometime after the freeze
- Shortest accepted solution 77 characters, longest 2225

# C – Conspiracies Everywhere (17/19, first at 11 min)

## Problem

Pick qualities for the bottom layer of a pyramid such that the top stone has the given quality when the quality of each stone is the sum of the qualities below it.

- Each quality has to be positive—at least one
- With $n$ layers, the minimum quality of the top stone is $2^{n-1}$
- Increasing the quality of the leftmost stone by $x$ increases the quality of the top stone by $x$
- Let other qualities be 1 and the leftmost stone $q - 2^{n-1} - 1$

# K – Key Cutting (16/19, first at 11 min)

## Problem

Pick intervals $[\ell_i, r_i]$ and depths $d_i$ such that $v_j == \max_{i \text{ s.t. } j \in [\ell_i, r_i]} d_i$ for the given array $v_j$.

- Solve recursively for intervals
- When cutting interval $[\ell, r]$, find smallest $v_j$ there and make cut that interval at depth $v_j$
- Now, some intervals within $[\ell, r]$ remain that need further cutting
- With e.g. minimum segment tree, $O(n \log n)$
- A linear time algorithm exists with an approach similar to finding largest rectangle

# F – Forgotten Measurements (16/19, first at 79 min)

## Problem

Find the minimum number of fence segments whose lengths are enough for determining the total circumference of the fence.

- Knowing lengths of segments going to the left determines the total length of horizontal segments
- Similarly for segments going right, up, and down
- Is $\min(\#L, \#R) + \min(\#U, \#D)$ optimal?
- Suppose we have one segment going to left and one going to right whose lengths we do not know
- Removing those segments splits the fence into two parts
- Since the segments don't overlap, the parts are some distance $\epsilon$ apart
- Shift one part to left by distance $\epsilon$
- Total length would increase by $2\epsilon$, but the description of the fence would remain the same

# J – Just Do It (9/19, first at 22 min)


```
// Sorts (a portion of) an array, divides it into partitions, then sorts those
algorithm quicksort(A, lo, hi) is
  if lo >= 0 && hi >= 0 && lo < hi then
    p := partition(A, lo, hi)
    quicksort(A, lo, p) // Note: the pivot is now included
    quicksort(A, p + 1, hi)

// Divides array into two partitions
algorithm partition(A, lo, hi) is
  // Pivot value
  pivot := A[lo] // Choose the first element as the pivot

  // Left index
  i := lo - 1

  // Right index
  j := hi + 1

  loop forever
    // Move the left index to the right at least once and while the element at
    // the left index is less than the pivot
    do i := i + 1 while A[i] < pivot

    // Move the right index to the left at least once and while the element at
    // the right index is greater than the pivot
    do j := j - 1 while A[j] > pivot

    // If the indices crossed, return
    if i >= j then return j

    // Swap the elements at the left and right indices
    swap A[i] with A[j]
```

# J – Just Do It (9/19, first at 22 min)

**Problem**

Create a pathological instance for quicksort.

- Make sure that each step the split is as uneven as possible
- Choose pivot among the two largest elements of the array

# J – Just Do It (9/19, first at 22 min)

**Problem**

Create a pathological instance for quicksort.

- Make sure that each step the split is as uneven as possible
- Choose pivot among the two largest elements of the array
- Keep an array of original indices
- Run algorithm, always ensuring uneven split
- Construct a DAG with edges "$x$ is larger than $y$"
- Topological order of the DAG gives the relative order of elements

# J – Just Do It (9/19, first at 22 min)

## Problem

Create a pathological instance for quicksort.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
        int n;
        cin >> n;

        for (int i = 0; i < n; ++i) {
                cout << (i + 1) % n + 1 << ' ';
        }
        cout << endl;
}
```

# E – Equilateral Numbers (8/19, first at 78 min)

**Problem**

Find the smallest number of triangle numbers with the desired sum.

- Studying the pattern suggests that three triangle numbers always suffice
- Enumerate triangle numbers up to target sum
- Test in $O(\sqrt{n})$ time if the target is a triangle number or a sum of two triangle numbers
- If not, then output 3

# H – Hiitism (7/19, first at 74 min)

**Problem**

Find a sequence of brushstrokes resulting in the given painting.

- Construct the sequence backwards
- Find a column or a row that has only a single color
- "Undo" that brushstroke
- Find the next column or a row with only a single color, ignoring the colors under the undone brushstrokes
- If colored cell remain in the end, the painting is fake

# G – Gerbil's Run (4/19, first at 123 min)

## Problem

Find a set of intervals on a circle such that

- for any point in an interval, both points in distance 1 to clockwise or ccw are not within any interval; and
- for any point outside an interval, either the point in distance 1 to clockwise or ccw is within an interval

- Pick intervals $[0, 1)$, $[2, 3)$, ... until the last upper bound $u$ is at least $2\pi r - 3$
- If $u$ is at least $2\pi r - 2$, we are done
- Otherwise, shift all intervals but the first one to the right by one

# L – Light Rail Connections (1/19, first at 119 min)

## Problem

Keep $3n$ edges such that

- connected nodes remain connected; and
- if two nodes would remain connected despite removing any two edges, that should be true after removing the edges

- Remove a spanning forest thrice from the graph: at most $3n - 3$ edges
- Connected nodes clearly remain connected
- If $\mathrm{mincut}(\mathrm{G}, \mathrm{v}, \mathrm{u}) \geq 3$, we want to have $\mathrm{mincut}(\mathrm{G}', \mathrm{v}, \mathrm{u}) \geq 3$
- Assume this does not hold and look at disjoint paths $P_1$, $P_2$, and $P_3$ between $v$ and $u$ in $G$
- There were at most 2 spanning forests with a path between $v$ and $u$; in one of them, they were in separate components
- At least one of the edges in $P_1$, $P_2$, or $P_3$ could have been added

# B – Budgetting Bridges (1/19, first at 181 min)

## Problem

For each query, decide if there is a minimum spanning tree containing the given edges.

- Consider edges of different weights separately in each query
- Consider Kruskal's MST algorithm
- After attempting to add all edges of weight $< w$, the connected components are uniquely determined
- Now, attempt to add edges of weight $w$ in the queries
- Each of them has to merge two disjoint components; if not, the query is invalid
- Implement functionality to undo edge additions to efficiently process the queries

# D – Dominoes (0/19)

## Problem

Put domino tiles in a sequence such that consequtive numbers are distinct.

- No solution exists iff tiles are of the form $(x, y)$ and $(y, x)$
- Otherwise, always solvable
- Merge tiles as long as you can, e.g., $(x, y) + (z, a)$ becomes $(x, a)$
- We are left with a tile $(x, y)$, and possibly several tiles $(y, x)$ if $x \neq y$
- Since $x \neq y$, we can move the rightmost tile in the sequence creating $(x, y)$ to be its leftmost tile instead
- We have at least three different types of tiles, so rotating the sequence creating $(x, y)$ sufficiently many times results in a merged tile distinct from $(x, y)$

# A – Adventure (0/19)

## Problem

Find a multicolored cycle that uses at most twice the minimum possible number of colors.

- Guess one of the nodes of the cycle
- Compute shortest paths to other nodes using the number of colors as the distance
- When BFS/Dijkstra tries to arrive to a node the second time, you have found two distinct short paths
- Their union has a cycle
- If starting node was part of the cycle, it used at most twice the optimal number of colors
- Repeat for all starting nodes

# I – Inheritance (0/19)

## Problem

Find two disjoint subsets with the same sum.

- A solution always exists by the pigeonhole principle: sums at most $2^n - 2$ but $2^n - 1$ nonempty subsets
- How many subsets are there whose sum is between $[\ell, r]$?
- Solvable in $O(2^{n/2} n)$ by splitting the items in two halves
- For both halves, compute sums of all subsets and sort them
- Count with e.g. two pointers
- If $[\ell, (\ell + r)/2]$ has more than $(\ell + r)/2 - \ell$ subsets, recurse there
- Otherwise recurse to $[(\ell + r)/2 + 1, r]$

# Results

# Results

- Third place:

# Results

- Third place: [table flip meme] (7/12, time 845)

# Results

- Third place: [table flip meme] (7/12, time 845)
- Second place:

# Results

- Third place: [table flip meme] (7/12, time 845)
- Second place: Aalto CS-A1140 Team 2 (8/12, time 1231)

# Results

- Third place: [table flip meme] (7/12, time 845)
- Second place: Aalto CS-A1140 Team 2 (8/12, time 1231)
- Winner:

# Results

- Third place: [table flip meme] (7/12, time 845)
- Second place: Aalto CS-A1140 Team 2 (8/12, time 1231)
- Winner: Barren plateau (8/12, time 808)