# UKIEPC 2023

Summary and solution outlines

# Problem Solutions

# Assessment

**25** correct • solved at: **??:??** by

**??**

**??**

## Overview

- We know that our grader will be using a tedious sorting algorithm
  - The algorithm sorts pairs (x,y)
  - (x,y) ⊂ (u,v) if all hold:
    - $x \le u$
    - $y \le v$
    - $x+y < u+v$
- What is an example of a worst-case input (cubic complexity) you can pass in to this algorithm?

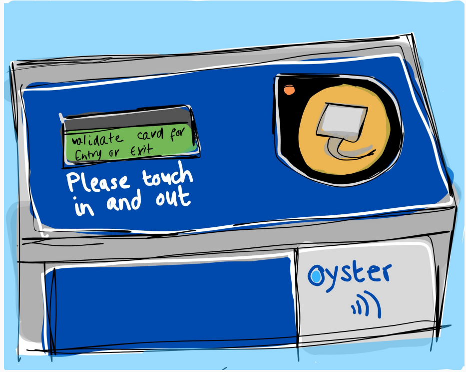
# Assessment

## Techniques

- Complexity analysis
- Construction

## Algorithm

- We need to construct a case with O(N^3) comparisons.
- We'll aim to guarantee two things in every rounds:
  - Commit at most **two** items and leave everything else **pending** so that we perform O(N) rounds.
  - For a significant fraction of the items eventually marked as **pending**, perform O(N) comparisons for $O(N^2)$ in total.
- How will we do this?
  - The first **K** elements should be incomparable, for example for (x, y) and (u, v) perhaps $x < u$, $y > v$
  - The remaining **N-K** elements can be arranged such that they are dominated by the last **N-K** elements of the first set.
    - So, in the first **N-K** rounds we will commit one element from the **K** set and one element from the **N-K** set.




# Boat Commuter

**124** correct • solved at: **??:??** by
**??**
**??**

## Overview

- People tap in and out of the ticket machines for ferries
  - We will fine people £100 if they fail to tap out or tap back in the same place
  - How much should we charge them?

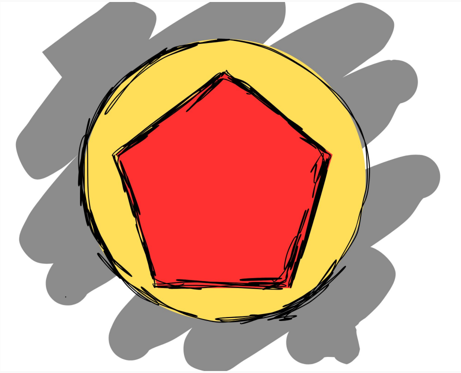
# Boat Commuter

## Techniques

- Maps
- Bookkeeping

## Algorithm

- Keep an array of the last place somebody tapped in.
  - Use -1 if they have not tapped in at all.
- Scan through the events in the order they are given.
  - If the last place somebody tapped in is -1,
    - Record where they tapped in
    - And immediately fine them £100
  - Otherwise,
    - Calculate the cost of their trip
    - And if their trip started and ended at different places, refund their £100
- Keeping track of fines separately also works. Just look out for open journeys at the end of the input.



# Clearing Space

**27** correct • solved at: **??:??** by
**??**
**??**

## Overview

- Choose N out of M points on a circle
  - Such that the polygon they define has maximum possible area

# Clearing Space

## Techniques

- Geometry
- Dynamic programming

## Algorithm

- Assume we start at point **S** in the input.
  - Rotate the input array so that point **S** is at the start.
  - We have to go through a subset of **K** points (including S itself at the end) to create the maximum-area polygon.
- The area of a polygon can be calculated using triangles from the centre of the circle.
  - For every pair of adjacent points **A** and **B**, their area contribution is (Ay - By) * (Ax - Bx) / 2
- The solution is a recursion which can be memoised with dynamic programming:
  - **answer**(p, points_used) = max(
    answer(p’ < p, points_used-1)+ area(p’, p)
- Try every possible start point **S**.


# Delivery Forces

**151** correct • solved at: **??:??** by
**??**
**??**

## Overview

- We have 3N people and we must group them into teams of 3
  - Such that their "strength" is the median of the group
  - Find the way to group people together such that the sum of strength is largest

# Delivery Forces

## Techniques

- Greedy algorithms
- Sorting

## Algorithm

- The biggest element certainly can't be a median
  - The only way the second-biggest element can be a median is if it is in a set with the biggest,
  - So, the optimal thing for the two biggest elements is to put them together.
  - Combine the two biggest elements with the smallest element (since this is no worse than any other)
- Now we have **N-3** items and can repeat
- If we simply reverse-sort the list, we can take every second element and sum up until we reach **N/3** elements.


# Enchanted

**1** correct • solved at: **??:??** by
**??**
**??**

## Overview

- When we choose a subset of spells including (i,j) we look up in a matrix cell (i,j) to find how much they contribute to our strength
  - Find which subset we should choose so that sum(strength) is largest

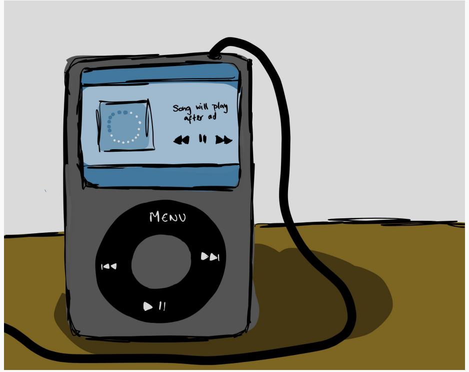
# Enchanted Fortress

## Techniques

- Bitmasks
- Meet-in-the-middle
- (Simulated annealing)

## Algorithm

- A simple brute-force over all combinations is almost effective, but too slow because we have to do a heavy loop at the end to confirm the solution is correct.
  - So, instead, we'll do 2 smaller brute-forces on bitmasks and work on a fast way of combining them.
  - For each half of the array, run a brute-force version of the algorithm and record the "partial" strength of all the pairwise combinations in that half.
- To combine the two halves of our answer, we have to brute-force everything.
  - We still need to collect some information to "join" the two halves. We can use join_result[mask] to quickly construct join_result[mask | x]




# Fast Forward

22 correct • solved at: **??:??** by
**??**
**??**

## Overview

- We are going to play a playlist starting from song X and ending on song (X+N-1) MOD N
- After K seconds since the last advertisement **and** after the last song finishes playing, we play an advertisement
    - For each X, how many ads are we going to hear?

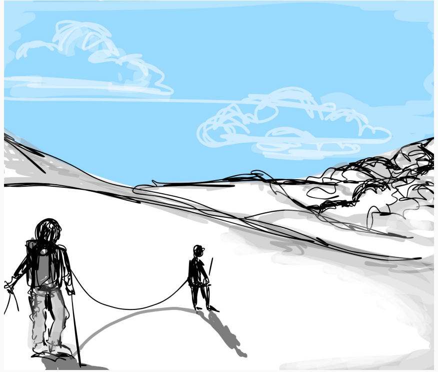
# Fast Forward

## Techniques

- Jump tables
- Two pointers

## Algorithm

- Assuming we start at song X, we can easily work out next(X): the next time at which we'll hear an advertisement
  - What about the second-next time? This is simply next(next(X)) if we already calculated all the values of next(). Let's record it as next[1](X).
  - Similarly we can record the position after 4 ad breaks next[2](X) = next[1](next[1](X))
- We need to know how many times T we can iterate next(X) before arriving back at X (modulo N)
  - Generate T bit-by-bit, starting from a large number B (eg. next[B=20] for 1048576 ads) and checking if this goes over. As long as it does, keep reducing B until we can repeat.
- Complexity: O(NlogN)



# Glacier Travel

**1** correct • solved at: **??:??** by
**??**
**??**

## Overview

- You are walking along the same trail as someone else, X metres behind them on the trail
    - However, the trail turns left and right and doubles back, so you may be closer at some times
    - What is the closest you will come during the walk?

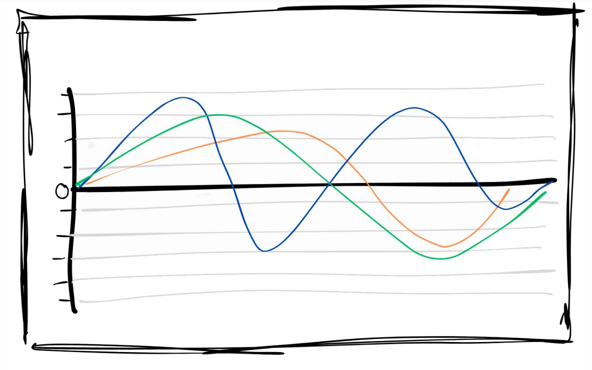
# Glacier Travel

## Techniques

- Two-pointers
- Ternary search
- 3:1 Hauling system

## Algorithm

- If persons A and B were simply travelling in the same straight lines forever, this would be an easy problem.
    - Either analytically, use calculus to find the time at which the square of their distance is as small as possible
    - Or numerically, use a ternary search to find the same time.
- We can simplify the problem by cutting it up
    - Persons A and B change directions every time they come to a vertex. Person A comes to the vertex at time L (where L = sum of all the lengths up to the vertex), and person B comes to the vertex at time L+S.
- Put all of the key times into a set, and for each adjacent pair of times compute the directions persons A and B are walking, and solve this smaller problem, then take the minimum of all answers.




# History

**3** correct • solved at: **??:??** by

**??**

**??**

**Overview**

- You have a sequence of data with condition for the “validity” of a subsequences:
    - The local minimums of the array are strictly increasing
    - Local minimums are calculated ignoring adjacent identical values
- We will repeatedly modify segments of this array by a constant amount
    - Given some subsequences on-demand, calculate if they are “valid”

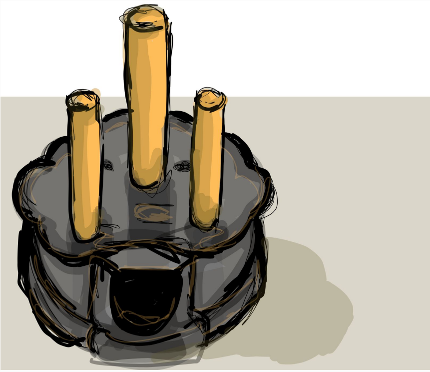
# History in Numbers

## Techniques

- Segment trees
- Lazy updates

## Algorithm

- We will keep track of ranges of “increasing” and “non-increasing” sequences and their boundaries in a segment tree.
  - When it comes to time to update a range, in theory we must run some extra logic on each subtree to get them into a correct state for the next query and deal with boundary conditions.
  - However, we may have more updates than queries, or queries may not touch the changed nodes, so instead we can mark the node as “pending change by X” if we don’t need the result right now.
- When recursing through the segment tree, we need to process pending updates and make the query afterwards.



# International

**0** correct • **not** solved

## Overview

- Fit 3 power plug pins to 3 power socket cylinders
  - The sizes of the pins are different from the sizes of the pins, so fitting is non-trivial
  - You may need to rotate/move the pins to get them to fit.

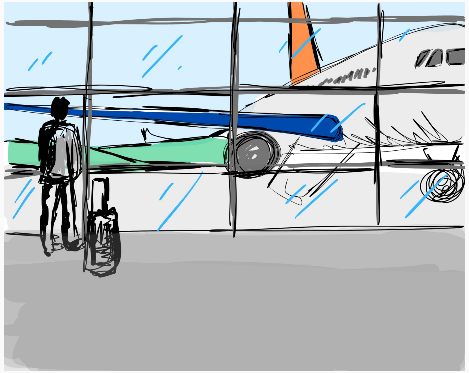
# International Travels

## Techniques

- Graph generation
- Depth-first-search

## Algorithm

- First, make a brute-force loop over the (few) possible ways of matching pins to socket cylinders.
    - Now that the assignments are fixed, we can “shrink” the pins to points and “grow” the cylinders by the same amount.
    - The problem is now to fit the 3 vertices of a triangle into their 3 assigned circles.
- The next useful principle is that if a solution is possible, there is a solution where two of the vertices are on the border of their circles.
    - You will need case analysis on the different ways the points align with the circles, plus 2-dimensional geometry (and/or more ternary search), and a strong drink.



# Journey

**0** correct • solved at: **02:53** by

**??**

**??**

## Overview

- You expect a flight on your travel itinerary to get cancelled
  - When this happens, you will take a direct shortest path to your final destination airport
  - What is the latest that this can end up making you, if you behave optimally?

# Journey of Recovery

## Techniques

- Shortest paths
- Graph reversal

## Algorithm

- Use the flight list to find all of the “interesting” times for an airport.
    - Make a graph which has one vertex for every (airport x time) combination, eg. (Sydney @ 0d:23:30)
    - Edges consist of flights from the input, as well as “default” edges to the next interesting time in the same airport.
- For any vertex, we need to know the earliest time to arrive at our destination **T**. Reverse all the edges of the graph and run depth-first search from all the (**T**, time) combinations starting from the latest one
    - When visiting a node for the first time, mark it as “visited” and annotate it with “time”.
- For all flights on the itinerary, check the **T**-time in O(1) and print the maximum. Tiebreak by moving itinerary flights 0.5 minutes earlier.

# Kernel Scheduler

**30** correct • solved at: **??:??** by
??
??

## Overview

- Remove less than half of a set of dependencies
  - Such that there are no cycles in the dependencies any more
  - Print the remaining dependencies

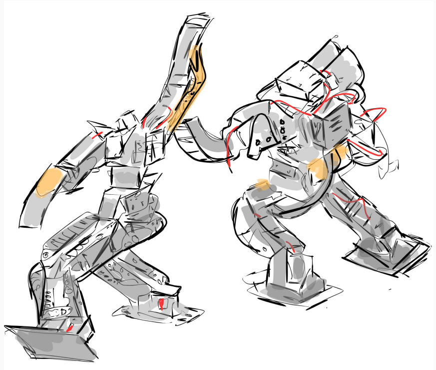
# Kernel Scheduler

## Techniques

- Acyclic graphs

## Algorithm

- The dependencies form a graph. We are looking to make an **acyclic subgraph** (no loops) with at least M/2 edges.
- One easy way of fulfilling the brief is to look at the ordering of the tasks and split the dependencies into two classes:
  - **increasing**: the edge $(a \to b)$ connects $(a < b)$
  - **decreasing**: the edge $(a \to b)$ connects $(a > b)$
- There can never be a loop in a graph made only of **increasing** dependencies, or only of **decreasing** dependencies, and at least one set contains half or more of the edges.
  - So, make the two sets, and then print the bigger one.



# Last One Standing

**161** correct • solved at: **??:??** by
**??**
**??**

**1x unsuccessful attempt by the GenAI team**

## Overview

- Every T seconds, one robot does D damage to the other robot with health H
  - The other robot will be doing exactly the same back with its own values of T, D, and H
  - Which robot will be victorious? If both robots fire at the same time both are hit later

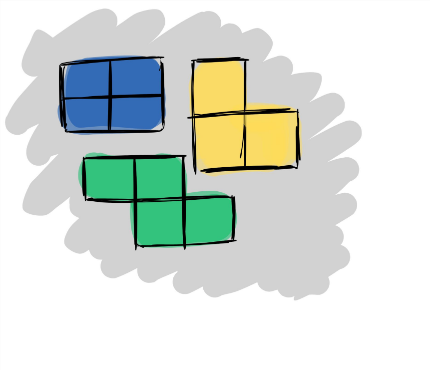
# Last One Standing

## Techniques

- Integer arithmetic
- Endurance

## Algorithm

- We must calculate the time at which each robot dies and compare them. For this we need:
  - **H** = Health
  - **D** = Damage (other player)
  - **T** = Time (other player)
- After **X** seconds, we'll have taken $\lfloor (X+T)/T \rfloor$ hits
  - Thus, solve $H \le D*\lfloor (X+T)/T \rfloor$ for X
  - This gives $\lceil H/D-1 \rceil * T = X$
- Compare the times, and print the appropriate answer.



# Mini-Tetris 3023

**169** correct • solved at: **??:??** by
**??**
**??**

**Also solved by the GenAI team!**

## Overview

- You have 3 kinds of Tetris pieces and want to use them to build a really long 2xN rectangle
  - Without any gaps
  - How big can you make this rectangle?

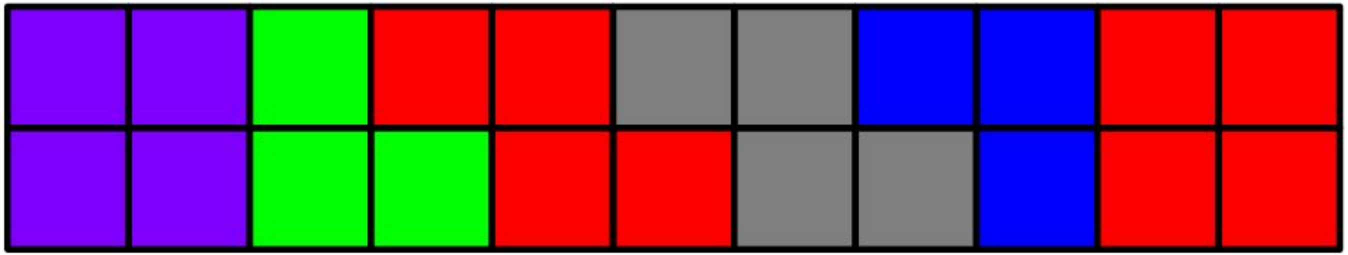
# Mini-Tetris 3023

## Techniques

- Logic
- Construction

## Algorithm

- 2x2 squares just add 2 points each to the answer
- “S” pieces are useless on their own
  - Whenever we have at least 2 “L” pieces, we can sandwich all the “S” pieces between them in one conga line:



- The remaining “L” pieces must be paired up into 3-square-wide blocks, so round down to an even number and multiply by 1.5.

# Naming Wine

**155** correct • solved at: **??:??** by
**??**
**??**

**Also solved by the GenAI team!**

**Overview**

- You have a collection of sizes of wine bottles.
  - Create names for them.
  - The names must be unique and consistent for the same-sized bottle.

# Naming Wine Bottles

## Techniques

- Hash maps

## Algorithm

- We're going to invent creative names for the bottles based on a "canonical" version of the number so that it's consistent.
- One algorithm:
  - Convert the input to a string. Remove trailing "0" and "L" characters, and the "." if it is the only thing remaining.
  - Map all the digits "0-9" to "a-j" and the decimal point "." to "z". Print this out.
- Another option:
  - Create a (hash)map of sizes to names. Floats (narrowly) work as keys.
  - For each element, check the map. If it's not present generate a random name.

# Final Standings

http://ukiepc2023.cloudcontest.org/