Bergen Open 2021
Solution slides

November 6, 2021

The jury

- Petter Daae
- Simen Hornnes
- Brigt Arve Toppe Håvardstun
- Torstein Strømme
- Kristoffer Æsøy

Special thanks:

- Greg Hamerly (Kattis)
- Olav Røthe Bakken

Junior price robot

- Input: A list of numbers
- Question: What is the distance between the first element and the next element which is less than or equal to the first element?
- Algorithm:
  - Let a0, a1, ..., a(n-1) denote the numbers in the list
  - for i in 1, 2, ..., n-1:
    - if ai <= a0: return i
  - else return “infinity”

- Runtime: *O(n)*

Author: Torstein Strømme **First solved**: 00:04 **Solved by**: 35 teams

Archipelago

- Problem summary: sort the islands by their “airport utility.”
  - Airport utility is defined as how many islands one can reach by travelling at most *d* kilometers before refuelling
- Observation: all islands that can reach each other have the same utility
- Algorithm A:
  - Make a graph: compare all islands, make an edge between them if they are within reach of each other
  - Do dfs or bfs to explore the graph. Count how many vertices are discovered for each root
  - Set utility of the discovered vertices found before moving on to the next root
  - Sort the islands by their utility
- Runtime: $O(n^2)$

Author: Kristoffer Æsøy **First solved**: 00:06 **Solved by**: 14 teams

Archipelago

- Problem summary: sort the islands by their “airport utility.”
  - Airport utility is defined as how many islands one can reach by travelling at most *d* kilometers before refuelling

- Observation: all islands that can reach each other have the same utility
- Algorithm B:
  - Use union-find. Store size of each component (like a size-balanced union-find would do).
  - For each pair of islands: call union on them if their distance is less than or equal to *d*
  - Utility of an island is the size of its component
  - Sort the islands by their utility

- Runtime: $O(n^2)$ (almost regardless of which union-find structure is used)

Author: Kristoffer Æsøy **First solved**: 00:06 **Solved by**: 14 teams

Coins

- Problem summary: Pick 1, 2 or 3 coins from the pile; avoid to pick the last coin.
- Clearly, you're in a losing position if there's only 1 coin left
- Clearly, you're in a winning position if there's 2, 3 or 4 coins left – respectively pick 1, 2 or 3 coins such that your opponent go to the losing position
- If there's 5 coins left, your opponent ends up in a winning position no matter what you do – hence, you're in a losing position
- If there's 6, 7 or 8 coins left, you're in a winning position – respectively pick 1, 2 or 3 coins to leave your opponent with 5 coins left.
- ...and so forth.

Author: Kristoffer Æsøy **First solved**: 00:10 **Solved by**: 22 teams

Coins

- Problem summary: Pick 1, 2 or 3 coins from the pile; avoid to pick the last coin.
- Observation: you're in a losing position if there are $4k + 1$ coins left
- Strategy: pick the number of coins such that your opponent will have $4k + 1$ coins left.
  - If there are $4k$ coins left (i.e. number of coins % 4 is 0), pick 3
  - If there are $4k+3$ coins left (i.e. number of coins % 4 is 3), pick 2
  - If there are $4k+2$ coins left (i.e. number of coins % 4 is 2), pick 1

- TLE should not be an issue (unless you recursively try every possible game or something)

Author: Kristoffer Æsøy **First solved**: 00:10 **Solved by**: 22 teams

Glitching screen

- Problem summary: Can you uniquely identify which picture it is, even when some pixels are incorrectly set to 0?
- Algorithm: just do it
  - For each picture:
    - for each row:
      - for each column:
        - if there is an active pixel on the screen, but not in the picture, then it can't be this picture
  - Output 'yes' if the number of qualified pictures is 1

- Runtime: $O(n)$

Author: Petter Daae **First solved**: 00:13 **Solved by**: 21 teams

Irritating accountants

- Problem summary: Sort items according to order of categories the account operates with.

- Algorithm:
  - Use a dictionary/hashmap/treemap to map categories to their sorting index
  - Use a dictionary/hashmap/treemap to map items to their category
  - Use a list of lists: append each bought item to the list at their category's index
  - Print the items in the lists in correct order

- Runtime: $O(n+m)$

Author: S. Hornnes, T. Strømme, K. Æsøy, and P. Daae **First solved**: 00:30 **Solved by**: 15 teams

King of Cans

- Input: The number of bottles worth 2 and 3 kroners, respectively
- Question: How many piles of bottles worth exactly 100 kroners can we create?

- Observation: You must always use an even number of 3's in every pile
  - You can divide the number of 3's by two (round down) and think of them as 6's instead
- Observation: 2's are strictly more flexible than 6's
  - Everything you can do with 6's you can also do with the same worth of 2's
- Conclusion: Greedily use as many 6's as possible in each pile.
  - Using 16 of them yields 96 kroners – then use two 2's to get up to 100

Author: Brigt Arve Toppe Håvardstun **First solved**: 00:38 **Solved by**: 14 teams

King of Cans

- Input: The number of bottles worth 2 and 3 kroners, respectively
- Question: How many piles of bottles worth exactly 100 kroners can we create?

- Greedily use as many 6's as possible in each pile
  - repeat:
    - pick 6's: min(16, number of remaining 6's)
    - pick 2's: as many as necessary to make 100
    - if there were not enough resources, break. Otherwise, increase counter.

- Runtime: $O(a + b)$

Author: Brigt Arve Toppe Håvardstun **First solved**: 00:38 **Solved by**: 14 teams

King of Cans

- Input: The number of bottles worth 2 and 3 kroners, respectively
- Question: How many piles of bottles worth exactly 100 kroners can we create?

- Observation: the only way bottles go to waste, is if there are not enough 2's
  - need at least two 2's for each pile
  - print(min((6 * sixes + 2 * twos) / 100, twos / 2))

- Runtime: *O(1)*

Author: Brigt Arve Toppe Håvardstun **First solved**: 00:38 **Solved by**: 14 teams

Doomsday

- Problem summary: Walk from base and fetch water and food before returning to base.

- Algorithm:
  - Run Dijkstra from base at location 0.
  - Add two new vertices to the graph:
    - connect the water depots to the first new vertex. Use the distance found in step 1 as weights.
    - connect the food depots to the second new vertex. Use the distance found in step 1 as weights.
  - Run Dijkstra to find distance between the two new nodes.

- $O(m \log n)$

Author: Simen Hornnes **First solved**: 00:40 **Solved by**: 3 teams

Elder price robot

- Problem summary: For each day, calculate how far back you need to go to to find a day which had a lower price.

- Naive algorithm: repeat the algorithm for the junior price robot
  - for each day:
    - step back in time until you find a day with a lower or equal price
    - report number of steps required

- $O(n^2)$ 😭

Author: Torstein Strømme **First solved**: 00:46 **Solved by**: 10 teams

Elder price robot

- Problem summary: For each day, calculate how far back you need to go to to find a day with has a lower price.

- Better algorithm
  - maintain a list *B* which holds the latest date the given price occurred. Initially all infinity long ago.
  - in backwards order of the input list:
    - check the list *B* for all possible prices <= to today's price – remember the latest date found
    - Compute difference of dates
    - Update the date of the current price in *B*

- $O(n^2)$ 😒

Author: Torstein Strømme **First solved**: 00:46 **Solved by**: 10 teams

Elder price robot

- Problem summary: For each day, calculate how far back you need to go to to find a day with has a lower price.

Using a segment tree

- Better algorithm
  - maintain a ~~list~~ B which holds the latest date the given price occurred. Initially all infinity long ago.
  - in backwards order of the input list:
    - check the list B for all possible prices <= to today's price – remember the latest date found
    - Compute difference of dates
    - Update the date of the current price in B

- ~~$O(n^2)$~~ $O(n \log n)$ 😄

Author: Torstein Strømme **First solved**: 00:46 **Solved by**: 10 teams

Elder price robot

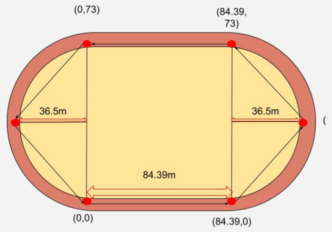
- Problem summary: For each day, calculate how far back you need to go to to find a day with has a lower price.

- Even better algorithm
  - maintain a stack with pairs (price, date) – the invariant is that both price and date is sorted
  - go through the list backwards:
    - pop all larger prices from the stack
    - the top of the stack now holds the next occurrence of a number smaller or equal
      - if empty, then "infinity"
    - put yourself on the stack

- $O(n)$ 🤩

Author: Torstein Strømme **First solved**: 00:46 **Solved by**: 10 teams

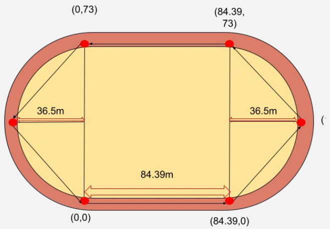
100 meter dash



- Problem summary: Given GPS locations with timestamps, what is the fastest 100m?
- Naive algorithm:
  - Guess every each location L_{start}. Then find the time used to run *100*m starting starting from L_{start}, and search forwards to find the nearest location L_{end} where the distance ran between L_{start} and $L_{end} \geq 100$.
  - Add up the time needed at each full segment. Compute the fractional time required for the last segment.
  - Observation: it might be better to let the first segment be fractional; deal with this case by also running algorithm backwards.
  - Observation: not necessary to account for the case where both starting and ending segments are fractional.
- $O(n^2)$ 😭

Author: Brigt Arve Toppe Håvardstun **First solved**: 01:20 **Solved by**: 4 teams

100 meter dash



- Problem summary: Given GPS locations with timestamps, what is the fastest 100m?
- Smarter algorithm:
  - Build up a distance array D, D[i] holding total distance from start to L_i.
    - Using this we can find the distance (time) between two locations in *O(1)* time.
  - Use a “sliding window” to move over the list of points::
    - Keep two pointers *start* and *end*; when distance L_{start} to L_{end} is smaller than 100, increment *end*.
    - Otherwise, compute the the time starting at *start* as before, and then increment *start*.
    - Remember fastest time as you go.
  - Slide over the points in both directions.
- *O(n)* 😁

Author: Brigt Arve Toppe Håvardstun **First solved**: 01:20 **Solved by**: 4 teams

Live aid

- Problem summary: Pick a non-overlapping set of intervals for the concert such that the attention is maximized. Output the total attention.
- Algorithm
  - (Weighted Interval Scheduling)
  - Sort intervals by end time
  - *p(i)* is the latest interval (by end time) that does not overlap with interval *i*. Find it by a binary search.
  - *DP[i]* is the total attention of the optimal scheduling of intervals from *0* to *i*
  - *DP[i+1]=max(DP[i-1], DP[p(i)] + a_i)*

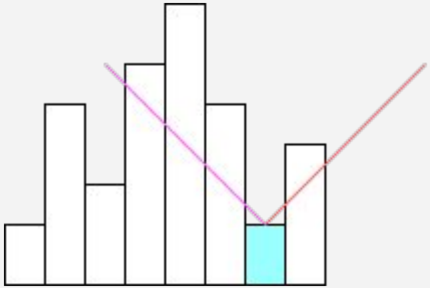
- *O(n log n)*

Author: Petter Daae **First solved**: 01:27 **Solved by**: 5 teams

Meticulous smoothing

- Problem summary: Difference in thickness between consecutive sections of wood can be no more than 1. What are the fewest strokes of sandpaper needed to obtain this?

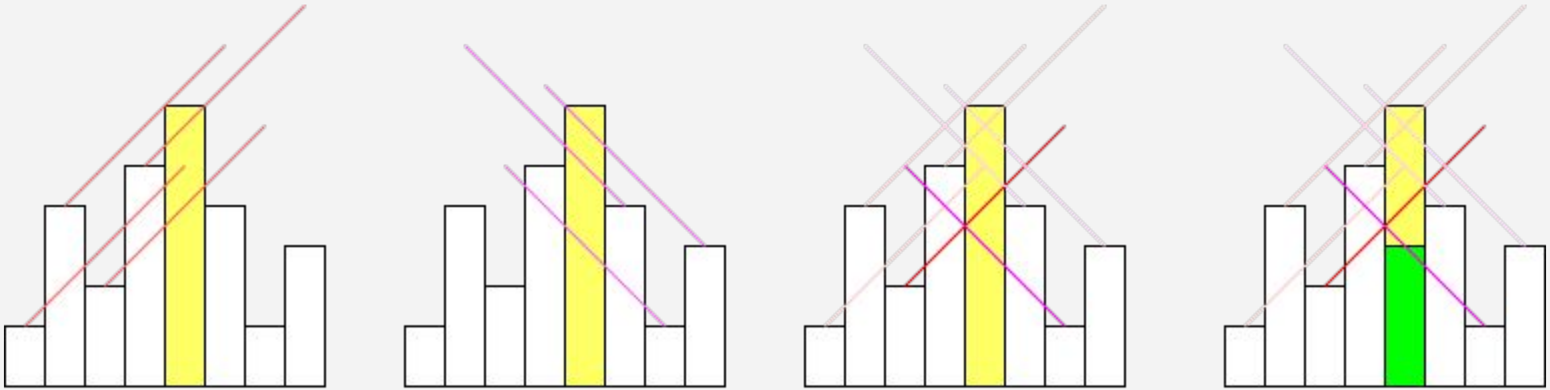
- Each point provides some upper limit for all other points



Author: Torstein Strømme **First solved**: 02:53 **Solved by**: 2 teams

Meticulous smoothing

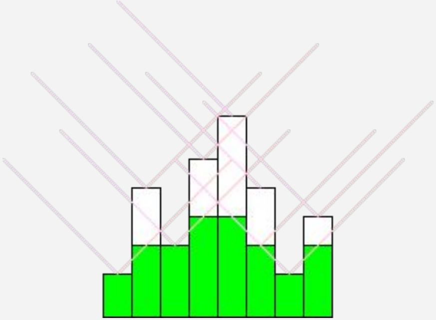
- Each point must respect limits set by all other points on both sides.
  - Requirement depends on height and distance
  - Must respect the strictest requirement



Author: Torstein Strømme **First solved**: 02:53 **Solved by**: 2 teams

Meticulous smoothing

- ➢ Observation: we only need to know the strictest limit from each side.
- ➢ Algorithm:
  - Walk along the list from left to right, and remember the strictest limit as we go.
  - At each step, the limit imposed by previous items is relaxed/heightened by 1.
  - Compare limit set by previous items with limit given by this item (i.e. the thickness at this point); keep the strictest limit. Mark the position with the limit.
  - Do the same backwards.
  - Final thickness is minimum of forward and backward limit.
  - Compute the differences for each point, and return their sum.



- ➢ Runtime: *O(n)*

Author: Torstein Strømme **First solved**: 02:53 **Solved by**: 2 teams

F1 racing

- Problem summary: A car uses $r+b*x$ seconds to complete one lap on x laps old tires. Given r, b, the time a pit stop takes, and the number of laps: what time is needed to finish a race?
- Observations:
    - Given a fixed number of pit stops, it is always best to distribute them as evenly as possible through the race.
        - The problem boils down to finding the optimal number of pit stops
        - Time required as a function of pit stops is either
            - non-decreasing (pit stop time is very large)
            - non-increasing (pit stop time is 0), or
            - follows a U-curve
        - Hence, we can ternary search the number of pit stops.

Author: B. A. T. Håvardstun, T. Strømme, P. Daae, and S. Hornnes **First solved**: N/A **Solved by**: 0 teams

F1 racing

- Problem summary: A car uses $r+b*x$ seconds to complete one lap on x laps old tires. Given r, b, the time a pit stop takes, and the number of laps: what time is needed to finish a race?

- How to find racetime using A pit stops?
  - segments = $A+1$
  - long_segments = n % segments
  - short_segments = segments - long_segments
  - The rest can be done in $O(1)$ time using math.
    - Sum $1..n \rightarrow n(n+1)/2$

laps_per_long_segment = ⌈total_laps / segments⌉

laps_per_short_segment = ⌊total_laps / segments⌋

Author: B. A. T. Håvardstun, T. Strømme, P. Daae, and S. Hornnes **First solved**: N/A **Solved by**: 0 teams

F1 racing

- Problem summary: A car uses $r+b*x$ seconds to complete one lap on x laps old tires. Given r, b, the time a pit stop takes, and the number of laps: what time is needed to finish a race?

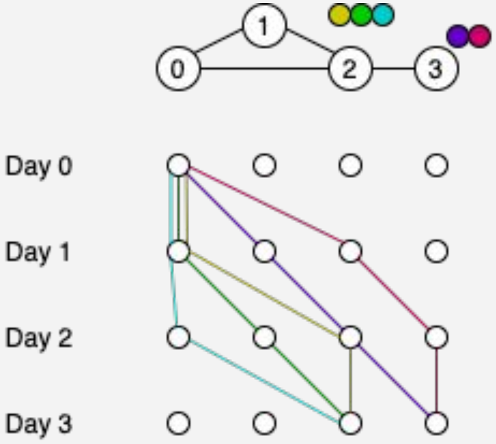
- Runtime w/ternary search + constant time calculation: $O(\log n)$ 😁
- Also accepted:
  - Try every number of pit stops up to square root of number of laps + try every number of laps per segment up to square root of number of laps, using constant time calculations → $O(\sqrt{n})$ 🙂
  - Ternary search + linear calculation of sum 1...n accepted in some languages (e.g. C++) → $O(n \log n)$ 😮‍💨

Setting bounds that killed this would have required the use of 128-bit integers or more to avoid overflow issues. So we didn't.

Author: B. A. T. Håvardstun, T. Strømme, P. Daae, and S. Hornnes **First solved**: N/A **Solved by**: 0 teams

Bombs

- Problem summary: Move bombs to their specified locations; at most one movement through each edge per day, at most one movement for each bomb per day.
- Visualize the sample test case:



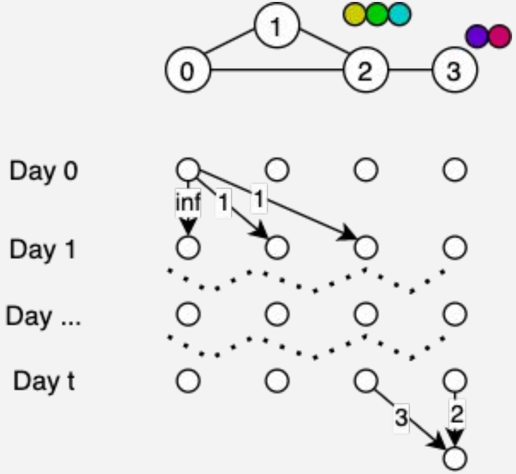
Author: Torstein Strømme **First solved**: N/A **Solved by**: 0 teams

Bombs

- Problem summary: Move bombs to their specified locations; at most one movement through each edge per day, at most one movement for each bomb per day.

- Guess (binary search) how many days are needed
- Create the "grid graph" of the guessed height
- If max flow = # of bombs, try fewer days
- Otherwise, try more days

- $O(n(n+t)(m(n+t))^2 \log(n+t))$ (w/ Edmonds-Karp)



Author: Torstein Strømme **First solved**: N/A **Solved by**: 0 teams

Statistics

- Number of teams: 37
- Number of participants: 83
- Number of submissions: 973
  - of these 8 were submitted by a team for a problem that they had already solved.
- Number of accepted submissions: 145
- First accepted submission: 00:04:47 (Junior price robot - solved by *Game Hoppers*)
- Last accepted submission: 04:58:49 (Glitching screen - solved by *Digitøs*)
- Number of commits to problem repository: 585

Copyright notes