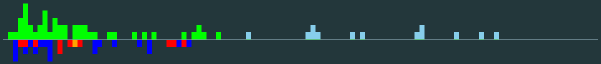
DAPC 2022

Solutions presentation

September 30, 2022

L: Lots of Liquid

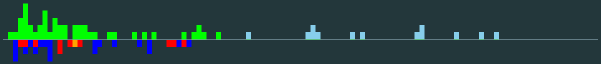
Problem Author: Maarten Sijm



- **Problem:** Find the length of the side of a cube that contains all liquid.

L: Lots of Liquid

Problem Author: Maarten Sijm

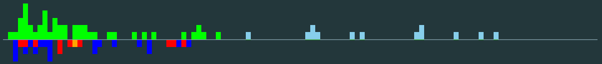


- **Problem:** Find the length of the side of a cube that contains all liquid.
- **Solution:** Calculate the value of following expression:

$$\sqrt[3]{\sum_{c} c^3}$$

L: Lots of Liquid

Problem Author: Maarten Sijm



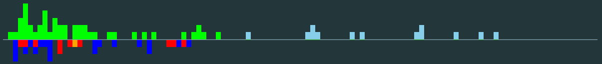
- **Problem:** Find the length of the side of a cube that contains all liquid.
- **Solution:** Calculate the value of following expression:

$$\sqrt[3]{\sum_{c} c^3}$$

- **Pitfall:** Make sure to use `double`, not `float`

L: Lots of Liquid

Problem Author: Maarten Sijm



- **Problem:** Find the length of the side of a cube that contains all liquid.
- **Solution:** Calculate the value of following expression:

$$\sqrt[3]{\sum_{c} c^3}$$

- **Pitfall:** Make sure to use `double`, not `float`

Statistics: 97 submissions, 46 accepted, 13 unknown

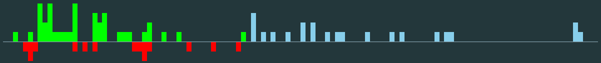
F: Fastestest Function

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given that `foo` took x% of the total run time before optimizing and y% after, what is the factor of how much faster `foo` got?

F: Fastestest Function

Problem Author: Ragnar Groot Koerkamp



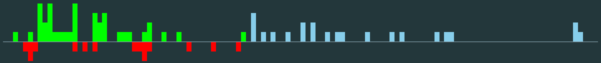
- **Problem:** Given that `foo` took $x\%$ of the total run time before optimizing and $y\%$ after, what is the factor of how much faster `foo` got?
- **Observation:** We can express this problem as the following equations:

$$\frac{\text{old time }\texttt{foo}}{\text{old time }\texttt{foo} + \text{other time}} = x\% \qquad \frac{\text{new time }\texttt{foo}}{\text{new time }\texttt{foo} + \text{other time}} = y\%$$

Goal: Rewrite these equations to find $\frac{\text{old time }\texttt{foo}}{\text{new time }\texttt{foo}}$

F: Fastestest Function

Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given that `foo` took $x\%$ of the total run time before optimizing and $y\%$ after, what is the factor of how much faster `foo` got?
- **Observation:** We can express this problem as the following equations:

$$\frac{\text{old time } \texttt{foo}}{\text{old time } \texttt{foo} + \text{other time}} = x\% \qquad \frac{\text{new time } \texttt{foo}}{\text{new time } \texttt{foo} + \text{other time}} = y\%$$

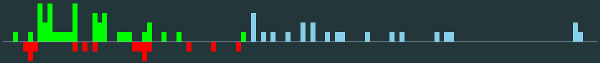
  Goal: Rewrite these equations to find $\frac{\text{old time } \texttt{foo}}{\text{new time } \texttt{foo}}$

- **Solution:**

$$\text{factor} = \frac{\text{old time } \texttt{foo}}{\text{new time } \texttt{foo}} = \frac{x \cdot (1 - y)}{y \cdot (1 - x)}$$

F: Fastestest Function

Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given that `foo` took $x\%$ of the total run time before optimizing and $y\%$ after, what is the factor of how much faster `foo` got?
- **Observation:** We can express this problem as the following equations:

$$\frac{\text{old time } \texttt{foo}}{\text{old time } \texttt{foo} + \text{other time}} = x\% \qquad \frac{\text{new time } \texttt{foo}}{\text{new time } \texttt{foo} + \text{other time}} = y\%$$

  Goal: Rewrite these equations to find $\frac{\text{old time } \texttt{foo}}{\text{new time } \texttt{foo}}$

- **Solution:**

$$\text{factor} = \frac{\text{old time } \texttt{foo}}{\text{new time } \texttt{foo}} = \frac{x \cdot (1 - y)}{y \cdot (1 - x)}$$

Statistics: 74 submissions, 37 accepted, 22 unknown

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations. Runtime: $\mathcal{O}(n^2)$.

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations.
  Runtime: $\mathcal{O}(n^2)$.
- **Optimized:**
  - Observe that high numbers move to the right immediately, and low numbers move $k-1$ to the left per iteration.
  - Solution: find the maximum distance (to the left) of any value to their sorted position (D), and output $\left\lceil \frac{D}{k-1} \right\rceil$.
  - Runtime: $\mathcal{O}(n \log n)$ for sorting, $\mathcal{O}(n)$ for finding the maximum distance.

B: Bubble-bubble Sort

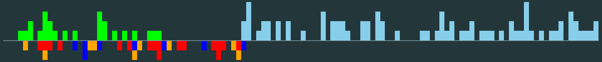
Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations. Runtime: $\mathcal{O}(n^2)$.
- **Optimized:**
  - Observe that high numbers move to the right immediately, and low numbers move $k-1$ to the left per iteration.
  - Solution: find the maximum distance (to the left) of any value to their sorted position (D), and output $\left\lceil \frac{D}{k-1} \right\rceil$.
  - Runtime: $\mathcal{O}(n \log n)$ for sorting, $\mathcal{O}(n)$ for finding the maximum distance.

Statistics: 73 submissions, 29 accepted, 27 unknown

A: Abbreviated Aliases

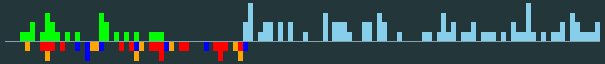
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given a list of n aliases, calculate the total length of the shortest unique prefixes of these aliases.

A: Abbreviated Aliases

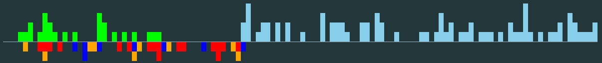
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given a list of n aliases, calculate the total length of the shortest unique prefixes of these aliases.
- **Naive solution:** For each alias, compare it to every other alias to find the shortest unique prefix of that alias.
  - Complexity: $\mathcal{O}(n^2)$ (too slow)

A: Abbreviated Aliases

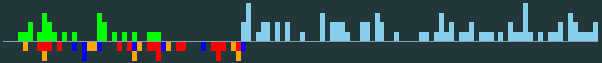
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given a list of n aliases, calculate the total length of the shortest unique prefixes of these aliases.
- **Naive solution:** For each alias, compare it to every other alias to find the shortest unique prefix of that alias.
  - Complexity: $\mathcal{O}(n^2)$ (too slow)
- **Solution:** First sort the list of aliases lexicographically. Then for each alias you only need to compare against the previous and the next alias in the sorted list to compute its shortest unique prefix.
  - Complexity: $\mathcal{O}(n \log n)$

A: Abbreviated Aliases

Problem Author: Ragnar Groot Koerkamp

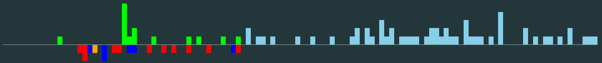


- **Problem:** Given a list of n aliases, calculate the total length of the shortest unique prefixes of these aliases.
- **Naive solution:** For each alias, compare it to every other alias to find the shortest unique prefix of that alias.
  - Complexity: $\mathcal{O}(n^2)$ (too slow)
- **Solution:** First sort the list of aliases lexicographically. Then for each alias you only need to compare against the previous and the next alias in the sorted list to compute its shortest unique prefix.
  - Complexity: $\mathcal{O}(n \log n)$

Statistics: 154 submissions, 25 accepted, 77 unknown

E: Extended Braille

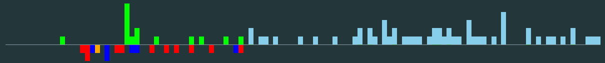
Problem Author: Wessel van Woerden



- **Problem:** Given *n* braille characters by their points, determine how many of them are distinct up to translation.

E: Extended Braille

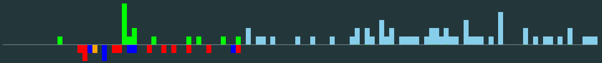
Problem Author: Wessel van Woerden



- **Problem:** Given n braille characters by their points, determine how many of them are distinct up to translation.
- **To compare two characters:** For each character $P = \{p_1, \ldots, p_m\} \subset \mathbb{Z}^2$ sort its points lexicographically such that $p_1 < \ldots < p_m$. For two sorted characters P, Q check if all points differ by the same translation: $p_1 - q_1 = \ldots = p_m - q_m$.
  - Complexity: $\mathcal{O}(m \log m)$.

E: Extended Braille

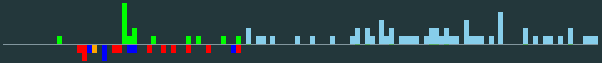
Problem Author: Wessel van Woerden



- **Problem:** Given n braille characters by their points, determine how many of them are distinct up to translation.
- **To compare two characters:** For each character $P = \{p_1, \ldots, p_m\} \subset \mathbb{Z}^2$ sort its points lexicographically such that $p_1 < \ldots < p_m$. For two sorted characters P, Q check if all points differ by the same translation: $p_1 - q_1 = \ldots = p_m - q_m$.
  - Complexity: $\mathcal{O}(m \log m)$.
- **Naive solution:** For each character, compare it to every other character.
  - Complexity: $\mathcal{O}(n^2)$ compares (too slow)

E: Extended Braille

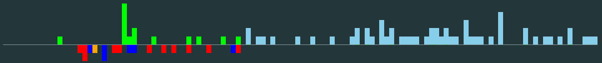
Problem Author: Wessel van Woerden



- **Problem:** Given n braille characters by their points, determine how many of them are distinct up to translation.
- **To compare two characters:** For each character $P = \{p_1, \ldots, p_m\} \subset \mathbb{Z}^2$ sort its points lexicographically such that $p_1 < \ldots < p_m$. For two sorted characters P, Q check if all points differ by the same translation: $p_1 - q_1 = \ldots = p_m - q_m$.
  - Complexity: $\mathcal{O}(m \log m)$.
- **Naive solution:** For each character, compare it to every other character.
  - Complexity: $\mathcal{O}(n^2)$ compares (too slow)
- **Solution:** For each character, sort its points, and translate such that the first point is $(0, 0)$. Then sort or use a hash set to count the number of unique characters.
  - Complexity: $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ compares

E: Extended Braille

Problem Author: Wessel van Woerden



- **Problem:** Given n braille characters by their points, determine how many of them are distinct up to translation.
- **To compare two characters:** For each character $P = \{p_1, \ldots, p_m\} \subset \mathbb{Z}^2$ sort its points lexicographically such that $p_1 < \ldots < p_m$. For two sorted characters P, Q check if all points differ by the same translation: $p_1 - q_1 = \ldots = p_m - q_m$.
  - Complexity: $\mathcal{O}(m \log m)$.
- **Naive solution:** For each character, compare it to every other character.
  - Complexity: $\mathcal{O}(n^2)$ compares (too slow)
- **Solution:** For each character, sort its points, and translate such that the first point is $(0, 0)$. Then sort or use a hash set to count the number of unique characters.
  - Complexity: $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ compares

Statistics: 91 submissions, 14 accepted, 56 unknown

C: Cookbook Composition

Problem Author: Timon Knigge

- **Problem:** Order the recipes by accessibility (lowest $\frac{\text{beginner time}}{\text{expert time}}$ first).

C: Cookbook Composition

Problem Author: Timon Knigge

- **Problem:** Order the recipes by accessibility (lowest $\frac{\text{beginner time}}{\text{expert time}}$ first).
- **Solution:** For every recipe, calculate:
  - beginner time by summing the durations of all steps.
  - expert time by calculating the maximal finish time of a step.

C: Cookbook Composition

Problem Author: Timon Knigge

- **Problem:** Order the recipes by accessibility (lowest $\frac{\text{beginner time}}{\text{expert time}}$ first).
- **Solution:** For every recipe, calculate:
  - beginner time by summing the durations of all steps.
  - expert time by calculating the maximal finish time of a step.
- The finish time of a step is the step's duration plus the maximal finish time of its dependencies.
  - Calculate using dynamic programming by running over all steps in order.

C: Cookbook Composition

Problem Author: Timon Knigge

- **Problem:** Order the recipes by accessibility (lowest $\frac{\text{beginner time}}{\text{expert time}}$ first).
- **Solution:** For every recipe, calculate:
  - beginner time by summing the durations of all steps.
  - expert time by calculating the maximal finish time of a step.
- The finish time of a step is the step's duration plus the maximal finish time of its dependencies.
  - Calculate using dynamic programming by running over all steps in order.
- Finally, order the recipes by the given ratio.

C: Cookbook Composition

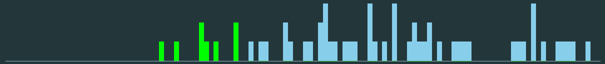
Problem Author: Timon Knigge

- **Problem:** Order the recipes by accessibility (lowest $\frac{\text{beginner time}}{\text{expert time}}$ first).
- **Solution:** For every recipe, calculate:
  - beginner time by summing the durations of all steps.
  - expert time by calculating the maximal finish time of a step.
- The finish time of a step is the step's duration plus the maximal finish time of its dependencies.
  - Calculate using dynamic programming by running over all steps in order.
- Finally, order the recipes by the given ratio.

Statistics: 111 submissions, 10 accepted, 87 unknown

K: Knitting Patterns

Problem Author: Maarten Sijm



- **Problem:** Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.

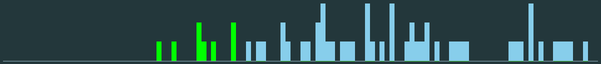
K: Knitting Patterns

Problem Author: Maarten Sijm

- **Problem:** Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.
- **Observation:** Between two times a colour of wool is used, you either leave the strand through the back unused for the entire gap, or you immediately end the use at the beginning of the gap and start using it at the end.

K: Knitting Patterns

Problem Author: Maarten Sijm



- **Problem:** Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.
- **Observation:** Between two times a colour of wool is used, you either leave the strand through the back unused for the entire gap, or you immediately end the use at the beginning of the gap and start using it at the end.
- **Solution:** For every colour, iterate through the knitting pattern and remember the index of the last time the colour occurred. If you encounter the colour again, the marginal cost is the minimum between leaving the strand unused the whole time since the last time, and the sum of the costs for ending and starting. Runs in $\mathcal{O}(|w| \cdot n)$.

K: Knitting Patterns

Problem Author: Maarten Sijm

- **Problem:** Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.
- **Observation:** Between two times a colour of wool is used, you either leave the strand through the back unused for the entire gap, or you immediately end the use at the beginning of the gap and start using it at the end.
- **Solution:** For every colour, iterate through the knitting pattern and remember the index of the last time the colour occurred. If you encounter the colour again, the marginal cost is the minimum between leaving the strand unused the whole time since the last time, and the sum of the costs for ending and starting. Runs in $\mathcal{O}(|w| \cdot n)$.
- **Remark:** Can be done in $\mathcal{O}(n)$ by doing some bookkeeping and storing for every colour the last time it occurred.

K: Knitting Patterns

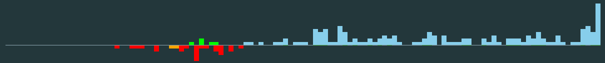
Problem Author: Maarten Sijm

- **Problem:** Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.
- **Observation:** Between two times a colour of wool is used, you either leave the strand through the back unused for the entire gap, or you immediately end the use at the beginning of the gap and start using it at the end.
- **Solution:** For every colour, iterate through the knitting pattern and remember the index of the last time the colour occurred. If you encounter the colour again, the marginal cost is the minimum between leaving the strand unused the whole time since the last time, and the sum of the costs for ending and starting. Runs in $\mathcal{O}(|w| \cdot n)$.
- **Remark:** Can be done in $\mathcal{O}(n)$ by doing some bookkeeping and storing for every colour the last time it occurred.

Statistics: 58 submissions, 8 accepted, 50 unknown

J: Jabbing Jets

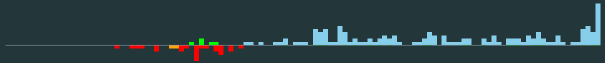
Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.

J: Jabbing Jets

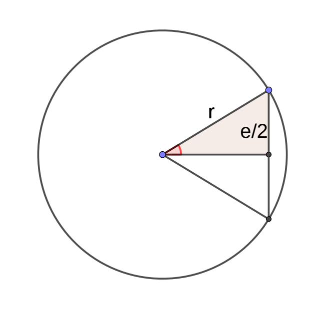
Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.

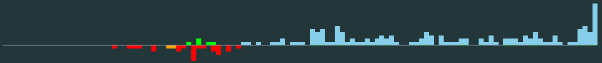
- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.
- The number of points on circle i is

$$\left\lfloor \frac{2\pi}{2\arcsin(\frac{e}{2r_i})} \right\rfloor.$$



J: Jabbing Jets

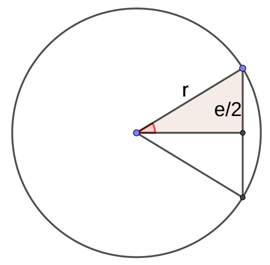
Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.
- The number of points on circle i is

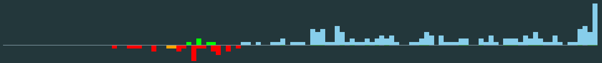
$$\left\lfloor \frac{2\pi}{2\arcsin\left(\frac{e}{2r_i}\right)} \right\rfloor .$$

- Edge case: if $2r_i < e$, the number of points is 1.



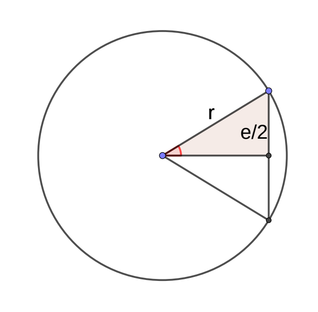
J: Jabbing Jets

Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.
- The number of points on circle i is

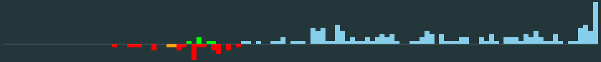
$$\left\lfloor \frac{2\pi}{2\arcsin\left(\frac{e}{2r_i}\right)} \right\rfloor.$$



- Edge case: if $2r_i < e$, the number of points is 1.
- Beware floating point issues!

J: Jabbing Jets

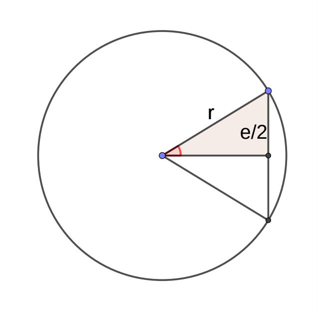
Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.
- The number of points on circle i is

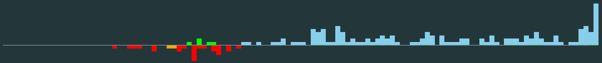
$$\left\lfloor \frac{2\pi}{2\arcsin\left(\frac{e}{2r_i}\right)} \right\rfloor .$$

- Edge case: if $2r_i < e$, the number of points is 1.
- Beware floating point issues!
  - Add $0.5 \cdot 10^{-6}$ to every radius.



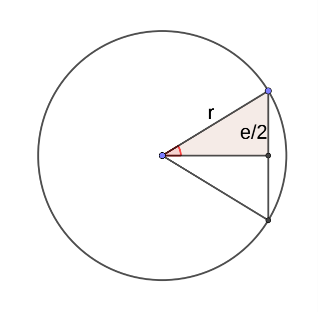
J: Jabbing Jets

Problem Author: Abe Wits



- **Problem:** Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.
- **Observation:** Because $r_{i+1} - r_i \geq e$, each circle can be considered separately.
- The number of points on circle i is

$$\left\lfloor \frac{2\pi}{2\arcsin(\frac{e}{2r_i})} \right\rfloor .$$

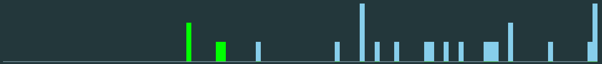


- Edge case: if $2r_i < e$, the number of points is 1.
- Beware floating point issues!
  - Add $0.5 \cdot 10^{-6}$ to every radius.

Statistics: 168 submissions, 5 accepted, 137 unknown

D: Dimensional Debugging

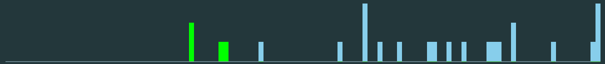
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?

D: Dimensional Debugging

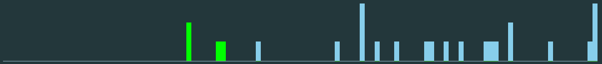
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?
- Since you know the answer in $\vec{0}$, you can verify the correctness of all algorithms with $\vec{L} = \vec{0}$.

D: Dimensional Debugging

Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?
- Since you know the answer in $\vec{0}$, you can verify the correctness of all algorithms with $\vec{L} = \vec{0}$.
- Once algorithm i has been verified, you can verify other algorithms j for which $\vec{L}_j \leq \vec{H}_i$.

D: Dimensional Debugging

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?
- Since you know the answer in $\vec{0}$, you can verify the correctness of all algorithms with $\vec{L} = \vec{0}$.
- Once algorithm i has been verified, you can verify other algorithms j for which $\vec{L}_j \leq \vec{H}_i$.
- More generally, the number of algorithms reachable in this way can be counted using BFS or DFS (floodfill).

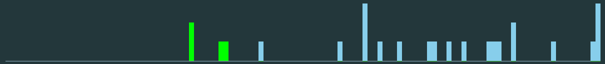
D: Dimensional Debugging

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?
- Since you know the answer in $\vec{0}$, you can verify the correctness of all algorithms with $\vec{L} = \vec{0}$.
- Once algorithm i has been verified, you can verify other algorithms j for which $\vec{L}_j \leq \vec{H}_i$.
- More generally, the number of algorithms reachable in this way can be counted using BFS or DFS (floodfill).
- Complexity: $\mathcal{O}(n^2)$

D: Dimensional Debugging

Problem Author: Ragnar Groot Koerkamp

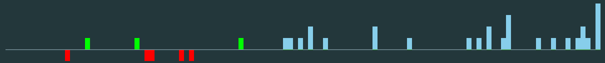


- **Problem:** Given n algorithms that only work when their input $\vec{x}$ is small enough ($\vec{x} \leq \vec{H}$), can you verify the correctness of all of them on sufficiently large inputs ($\vec{x} \geq \vec{L}$)?
- Since you know the answer in $\vec{0}$, you can verify the correctness of all algorithms with $\vec{L} = \vec{0}$.
- Once algorithm i has been verified, you can verify other algorithms j for which $\vec{L}_j \leq \vec{H}_i$.
- More generally, the number of algorithms reachable in this way can be counted using BFS or DFS (floodfill).
- Complexity: $\mathcal{O}(n^2)$

Statistics: 25 submissions, 4 accepted, 21 unknown

I: Inked Inscriptions

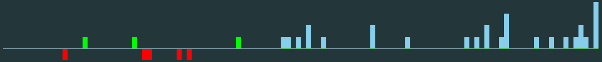
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.

I: Inked Inscriptions

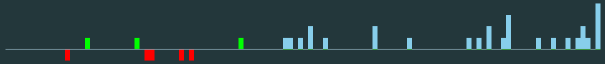
Problem Author: Ragnar Groot Koerkamp



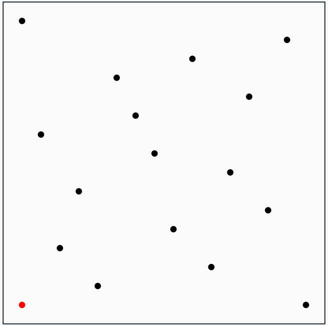
- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Most naive solutions use $\mathcal{O}(n^2)$ pageflips, so you need to be smarter.

I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp

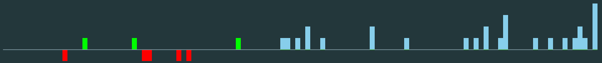


- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Most naive solutions use $\mathcal{O}(n^2)$ pageflips, so you need to be smarter.
- Idea: each psalm corresponds to a point in 2D space, and the pageflips needed to copy one psalm after another corresponds to their Manhattan distance.

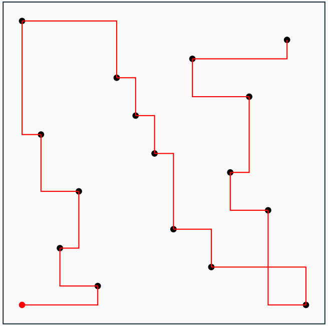


I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp



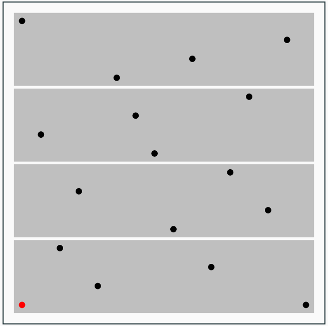
- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Most naive solutions use $\mathcal{O}(n^2)$ pageflips, so you need to be smarter.
- Idea: each psalm corresponds to a point in 2D space, and the pageflips needed to copy one psalm after another corresponds to their Manhattan distance.
- So, you need to find a path of bounded length that visits all points.



I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp

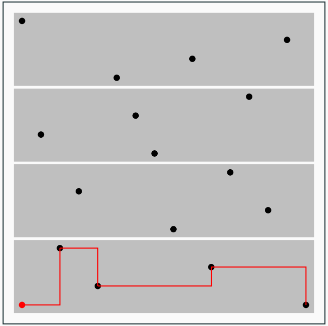
- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Solution idea: divide the plane into $\sqrt{n}$ bands of height $\sqrt{n}$.



I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp

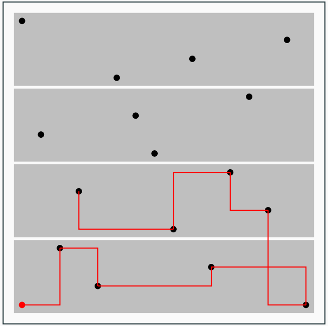
- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Solution idea: divide the plane into $\sqrt{n}$ bands of height $\sqrt{n}$.
- For the first band, iterate over all points in it from left to right.



I: Inked Inscriptions

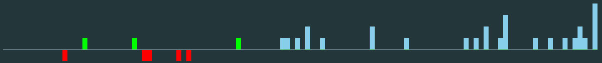
Problem Author: Ragnar Groot Koerkamp

- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Solution idea: divide the plane into $\sqrt{n}$ bands of height $\sqrt{n}$.
- For the first band, iterate over all points in it from left to right.
- For the second band, iterate over all points from right to left.

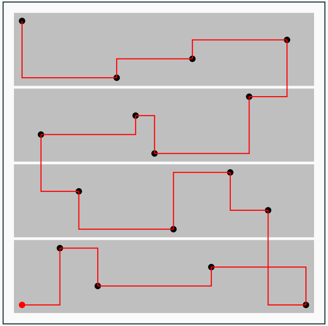


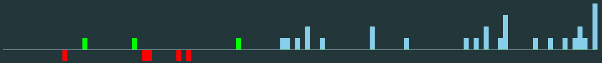
I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp

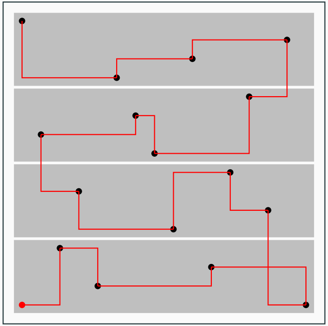


- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Solution idea: divide the plane into $\sqrt{n}$ bands of height $\sqrt{n}$.
- For the first band, iterate over all points in it from left to right.
- For the second band, iterate over all points from right to left.
- And so on...



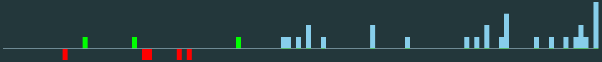


- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Each band uses n page flips for the horizontal segments, and at most $1 + 2 + \ldots + \sqrt{n} \approx n/2$ page flips for the vertical segments.

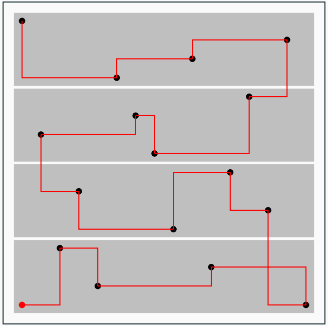


I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp

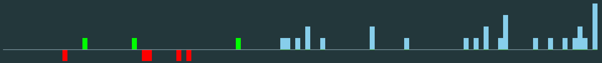


- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Each band uses n page flips for the horizontal segments, and at most $1 + 2 + \ldots + \sqrt{n} \approx n/2$ page flips for the vertical segments.
- The transitions between bands use at most $2n$ page flips total.

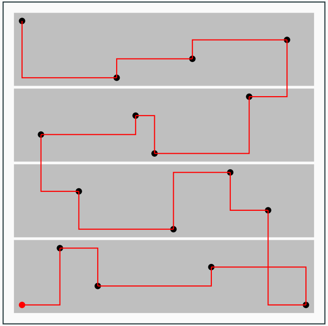


I: Inked Inscriptions

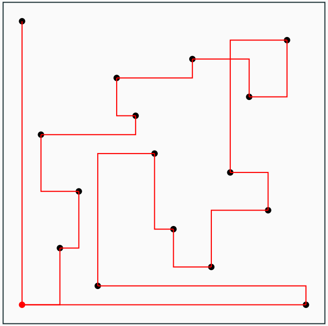
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Each band uses n page flips for the horizontal segments, and at most $1 + 2 + \ldots + \sqrt{n} \approx n/2$ page flips for the vertical segments.
- The transitions between bands use at most $2n$ page flips total.
- Overall: this uses at most $\sqrt{n}(n + n/2) + 2n = 1.5n\sqrt{n} + 2n$ page flips.

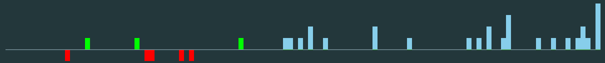


- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Alternative solution: just greedily go to the nearest unvisited point.

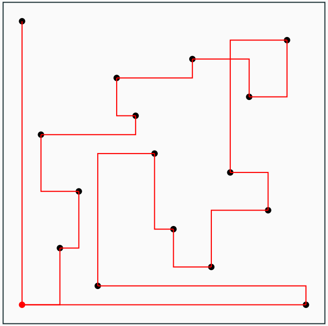


I: Inked Inscriptions

Problem Author: Ragnar Groot Koerkamp



- **Problem:** Copy n psalms in at most $2n\sqrt{n}$ pageflips.
- Alternative solution: just greedily go to the nearest unvisited point.



Statistics: 36 submissions, 3 accepted, 28 unknown

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!
- **Observation:** Many guesses may be needed when only a single digit is not known yet.

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!
- **Observation:** Many guesses may be needed when only a single digit is not known yet.
- **Solution:** Ensure that the first two 5-digit primes contain all 10 digits (e.g. 24683 and 10597) so that the set of digits is known. Then make up to 4 random consistent guesses.

G: Guessing Primes

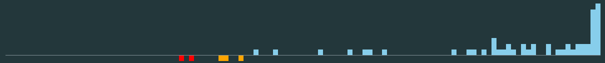
Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!
- **Observation:** Many guesses may be needed when only a single digit is not known yet.
- **Solution:** Ensure that the first two 5-digit primes contain all 10 digits (e.g. 24683 and 10597) so that the set of digits is known. Then make up to 4 random consistent guesses.
- This is guaranteed to succeed in 6 guesses:
  - When all digits are distinct, each digit can only be guessed in the wrong location 4 times, once in the first 2 guesses, and in 3 of the 4 remaining guesses.
  - When there are at most 4 distinct digits, each position can only be guessed wrongly at most 3 times.

  In both cases, the final guess must be correct.

G: Guessing Primes

Problem Author: Ludo Pulles



- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!
- **Observation:** Many guesses may be needed when only a single digit is not known yet.
- **Solution:** Ensure that the first two 5-digit primes contain all 10 digits (e.g. 24683 and 10597) so that the set of digits is known. Then make up to 4 random consistent guesses.
- This is guaranteed to succeed in 6 guesses:
  - When all digits are distinct, each digit can only be guessed in the wrong location 4 times, once in the first 2 guesses, and in 3 of the 4 remaining guesses.
  - When there are at most 4 distinct digits, each position can only be guessed wrongly at most 3 times.

  In both cases, the final guess must be correct.
- Note: The hidden test data simply consists of 13 test cases covering all 5-digit primes.

G: Guessing Primes

Problem Author: Ludo Pulles

- **Problem:** Guess the hidden 5-digit prime in at most 6 guesses, i.e., play *Primel*.
- First approach: repeatedly guess a random prime satisfying all constraints so far.
  - Wrong answer: Occasionally uses 7 guesses!
- **Observation:** Many guesses may be needed when only a single digit is not known yet.
- **Solution:** Ensure that the first two 5-digit primes contain all 10 digits (e.g. 24683 and 10597) so that the set of digits is known. Then make up to 4 random consistent guesses.
- This is guaranteed to succeed in 6 guesses:
  - When all digits are distinct, each digit can only be guessed in the wrong location 4 times, once in the first 2 guesses, and in 3 of the 4 remaining guesses.
  - When there are at most 4 distinct digits, each position can only be guessed wrongly at most 3 times.

  In both cases, the final guess must be correct.
- Note: The hidden test data simply consists of 13 test cases covering all 5-digit primes.

Statistics: 59 submissions, 0 accepted, 54 unknown

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2. What is the minimal cost to make all box positions distinct?

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Observation:** The boxes will remain in their original order (they will never overtake each other).

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Observation:** The boxes will remain in their original order (they will never overtake each other).
- **Observation:** Groups of consecutive boxes map to an interval.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Observation:** The boxes will remain in their original order (they will never overtake each other).
- **Observation:** Groups of consecutive boxes map to an interval.
- The cost of moving a box from position p to a position x, can be modelled with a quadratic function $C_p(x) = (x - p)^2$.
  - Example: For one box with original position 3 moved to position x, $C_3(x) = (x - 3)^2 = x^2 - 6x + 9$.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

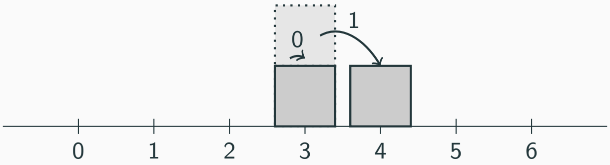
- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Observation:** The boxes will remain in their original order (they will never overtake each other).
- **Observation:** Groups of consecutive boxes map to an interval.
- The cost of moving a box from position p to a position x, can be modelled with a quadratic function $C_p(x) = (x - p)^2$.
  - Example: For one box with original position 3 moved to position x, $C_3(x) = (x - 3)^2 = x^2 - 6x + 9$.
- When adding the costs of two groups of boxes that overlap together, *translate* the cost function of the right group of boxes by the size of the left group.
  - Example: For two boxes with original position 3, moved such that the left-most box is at position x, the summed cost is $C_{3,3}(x) = C_3(x) + C_3(x + 1) = (x - 3)^2 + (x - 2)^2 = 2x^2 - 10x + 13$.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- The cost of a box at a position x, starting at position p, can be modelled with a quadratic function $C_p(x) = (x - p)^2$.
  - For two boxes that start at position 3, the summed cost is $C_{3,3}(x) = C_3(x) + C_3(x+1) = (x-3)^2 + (x-2)^2 = 2x^2 - 10x + 13$.

Proof by example:



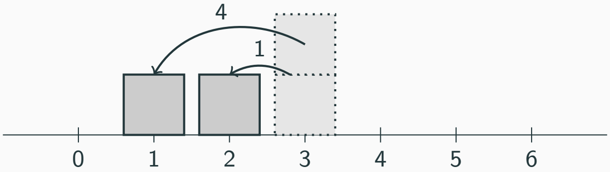
$$C_{3,3}(3) = 2 \cdot 3^2 - 10 \cdot 3 + 13 = 1$$

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- The cost of a box at a position x, starting at position p, can be modelled with a quadratic function $C_p(x) = (x-p)^2$.
  - For two boxes that start at position 3, the summed cost is
    $C_{3,3}(x) = C_3(x) + C_3(x+1) = (x-3)^2 + (x-2)^2 = 2x^2 - 10x + 13$.

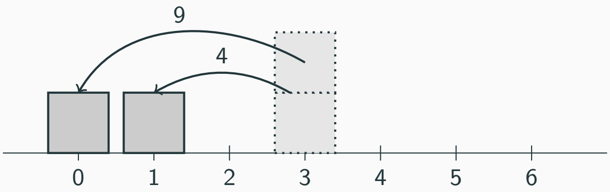
Proof by example:



$$C_{3,3}(1) = 2 \cdot 1^2 - 10 \cdot 1 + 13 = 5$$

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- The cost of a box at a position x, starting at position p, can be modelled with a quadratic function $C_p(x) = (x - p)^2$.
  - For two boxes that start at position 3, the summed cost is
    $C_{3,3}(x) = C_3(x) + C_3(x+1) = (x-3)^2 + (x-2)^2 = 2x^2 - 10x + 13$.

Proof by example:



$$C_{3,3}(0) = 2 \cdot 0^2 - 10 \cdot 0 + 13 = 13$$

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Solution:** Add every box from left to right, maintaining the optimal placement by maintaining the cost function for every group of boxes.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Solution:** Add every box from left to right, maintaining the optimal placement by maintaining the cost function for every group of boxes.
- If two groups of boxes touch or overlap, merge them into one group by summing their (possibly translated) costs.
  - This new group may overlap with its preceding group after the merge, so merge recursively.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2. What is the minimal cost to make all box positions distinct?
- **Solution:** Add every box from left to right, maintaining the optimal placement by maintaining the cost function for every group of boxes.
- If two groups of boxes touch or overlap, merge them into one group by summing their (possibly translated) costs.
  - This new group may overlap with its preceding group after the merge, so merge recursively.
- For every group with cost $C(x) = ax^2 + bx + c$, the minimal cost is:

$$C\left(\left\lfloor \frac{-b}{2a} + \frac{1}{2} \right\rfloor\right)$$

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Solution:** Add every box from left to right, maintaining the optimal placement by maintaining the cost function for every group of boxes.
- If two groups of boxes touch or overlap, merge them into one group by summing their (possibly translated) costs.
  - This new group may overlap with its preceding group after the merge, so merge recursively.
- For every group with cost $C(x) = ax^2 + bx + c$, the minimal cost is:

$$C\left(\left\lfloor \frac{-b}{2a} + \frac{1}{2} \right\rfloor\right)$$

- The total runtime is $\mathcal{O}(n)$ (after sorting): we do at most $n-1$ merges.

H: Heavy Hauling

Problem Author: Ragnar Groot Koerkamp

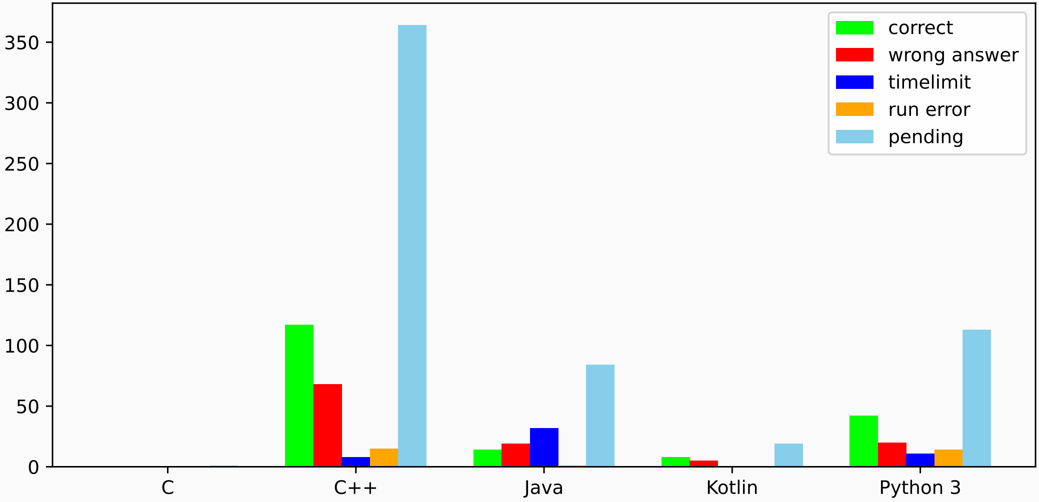
- **Problem:** Given n boxes at given positions. Moving a box d positions costs d^2.
  What is the minimal cost to make all box positions distinct?
- **Solution:** Add every box from left to right, maintaining the optimal placement by maintaining the cost function for every group of boxes.
- If two groups of boxes touch or overlap, merge them into one group by summing their (possibly translated) costs.
  - This new group may overlap with its preceding group after the merge, so merge recursively.
- For every group with cost $C(x) = ax^2 + bx + c$, the minimal cost is:

$$C\left(\left\lfloor \frac{-b}{2a} + \frac{1}{2} \right\rfloor\right)$$

- The total runtime is $\mathcal{O}(n)$ (after sorting): we do at most $n - 1$ merges.

Statistics: 10 submissions, 0 accepted, 9 unknown

Language stats



ETV Board also did their best!

Random facts

Jury work

- 298 commits

[1] *After* codegolfing

Random facts

Jury work

- 298 commits
- 375 secret test cases ($\approx$ 31 per problem!)

[1] *After* codegolfing

Random facts

Jury work

- 298 commits
- 375 secret test cases ($\approx$ 31 per problem!)
- 153 jury + proofreader solutions

[1] *After* codegolfing

Random facts

Jury work

- 298 commits
- 375 secret test cases ($\approx$ 31 per problem!)
- 153 jury + proofreader solutions
- The minimum[1] number of lines the jury needed to solve all problems is

$$5 + 3 + 8 + 9 + 3 + 2 + 25 + 9 + 3 + 4 + 6 + 2 = 79$$

On average 6.6 lines per problem, down from 7.5 in last year's preliminaries

[1] *After* codegolfing

Thanks to:

The proofreaders

Jaap Eldering
Kevin Verbeek
Mark van Helvoort
Nicky Gerritsen
Thomas Verwoerd

The jury

Boas Kluiving
Jorke de Vlas
Ludo Pulles
Maarten Sijm
Ragnar Groot Koerkamp
Reinier Schmiermann
Ruben Brokkelkamp
Wessel van Woerden