BAPC 2022

Solutions presentation

October 22, 2022

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \ldots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \ldots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.
- Verification:

$$\frac{1}{n}\sum_{i=1}^{n}(\sqrt{x/x'}a_i)^2 = \frac{x}{x'} \cdot \frac{1}{n}\sum_{i=1}^{n} a_i^2 = \frac{x}{x'} \cdot x' = x.$$

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \ldots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.
- Verification:

$$\frac{1}{n}\sum_{i=1}^{n}(\sqrt{x/x'}a_i)^2 = \frac{x}{x'} \cdot \frac{1}{n}\sum_{i=1}^{n} a_i^2 = \frac{x}{x'} \cdot x' = x.$$

- Exception: if $x' = 0$, return $0, \ldots, 0$.

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \dots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \dots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.
- Verification:

$$\frac{1}{n}\sum_{i=1}^{n}(\sqrt{x/x'}a_i)^2 = \frac{x}{x'} \cdot \frac{1}{n}\sum_{i=1}^{n} a_i^2 = \frac{x}{x'} \cdot x' = x.$$

- Exception: if $x' = 0$, return $0, \dots, 0$.
  - And apparently, we forgot to test the case `-1 1`, where the sum is 0

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \ldots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.
- Verification:
$$\frac{1}{n}\sum_{i=1}^{n}(\sqrt{x/x'}a_i)^2 = \frac{x}{x'} \cdot \frac{1}{n}\sum_{i=1}^{n} a_i^2 = \frac{x}{x'} \cdot x' = x.$$
- Exception: if $x' = 0$, return $0, \ldots, 0$.
  - And apparently, we forgot to test the case `-1 1`, where the sum is 0
- Pitfall: multiplying two `ints` causes integer overflow, use `long` instead!

E: Equalising Audio

Problem Author: Abe Wits

- **Problem:** Normalise the amplitudes $a_1, \ldots, a_n$ to a perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$.
- **Solution:** First, compute the current perceived loudness $x' = \frac{1}{n}\sum_{i=1}^{n} a_i^2$.
  Then, output $\sqrt{x/x'} \cdot a_1, \ldots, \sqrt{x/x'} \cdot a_n$ with sufficient precision.
- Verification:
$$\frac{1}{n}\sum_{i=1}^{n}(\sqrt{x/x'}a_i)^2 = \frac{x}{x'} \cdot \frac{1}{n}\sum_{i=1}^{n} a_i^2 = \frac{x}{x'} \cdot x' = x.$$
- Exception: if $x' = 0$, return $0, \ldots, 0$.
  - And apparently, we forgot to test the case `-1 1`, where the sum is 0
- Pitfall: multiplying two `ints` causes integer overflow, use `long` instead!

Statistics: 127 submissions, 50 accepted, 6 unknown

B: Bellevue

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.

B: Bellevue

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.
- **Observation:** If point P is blocking the view of the edge of the island from point Q, you can see more sea in P than Q.

B: Bellevue

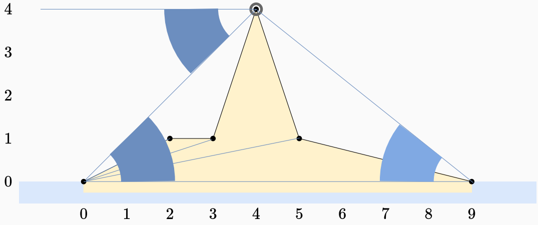
Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.
- **Observation:** If point P is blocking the view of the edge of the island from point Q, you can see more sea in P than Q.
- The answer is always an angle from the start/end of the island to another point.

B: Bellevue

Problem Author: Ragnar Groot Koerkamp

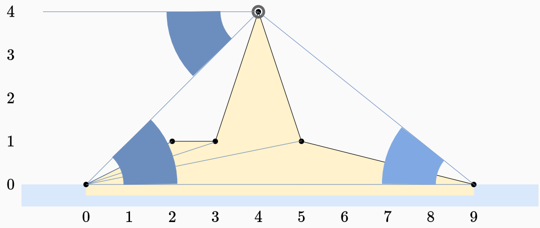
- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.
- **Observation:** If point P is blocking the view of the edge of the island from point Q, you can see more sea in P than Q.
- The answer is always an angle from the start/end of the island to another point.
- **Solution:** take the maximum angle around the start/end.



B: Bellevue

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.
- **Observation:** If point P is blocking the view of the edge of the island from point Q, you can see more sea in P than Q.
- The answer is always an angle from the start/end of the island to another point.
- **Solution:** take the maximum angle around the start/end.

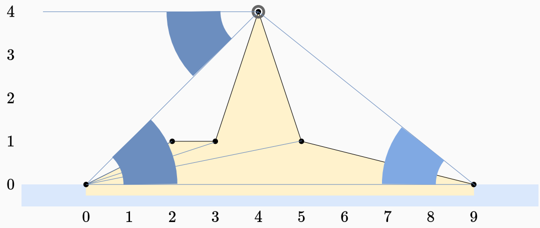


- **Alternative solution:** compute the convex hull and iterate over it.

B: Bellevue

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given the profile of an island, find the point with the largest viewing angle of the sea.
- **Observation:** If point P is blocking the view of the edge of the island from point Q, you can see more sea in P than Q.
- The answer is always an angle from the start/end of the island to another point.
- **Solution:** take the maximum angle around the start/end.



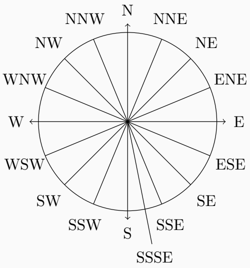
- **Alternative solution:** compute the convex hull and iterate over it.

Statistics: 102 submissions, 41 accepted, 18 unknown

F: Failing Flagship

Problem Author: Ruben Brokkelkamp

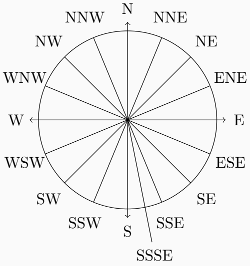
- **Problem:** Compute the minimum angle in degrees between two wind directions.



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

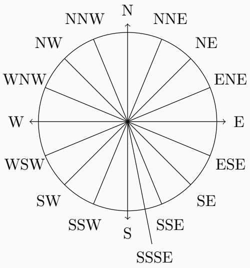
- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

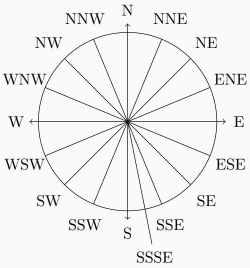
- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**
  - Convert both wind directions into degrees $d_1 \geq d_2$.



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

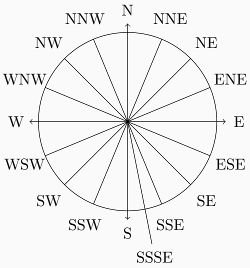
- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**
  - Convert both wind directions into degrees $d_1 \geq d_2$.
  - **Observation:** For every extra letter the degrees it represents halves: $45, 22.5, 11.25, 6.125, \ldots$.



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

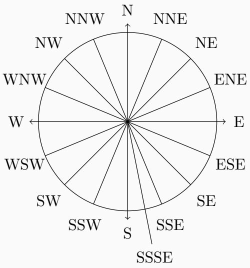
- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**
  - Convert both wind directions into degrees $d_1 \geq d_2$.
  - **Observation:** For every extra letter the degrees it represents halves: $45, 22.5, 11.25, 6.125, \ldots$.
  - Return $\min(d_2 - d_1, 360 + d_1 - d_2)$.



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

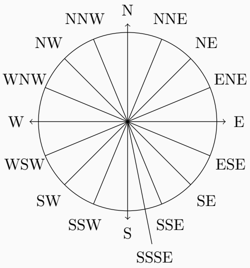
- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**
  - Convert both wind directions into degrees $d_1 \geq d_2$.
  - **Observation:** For every extra letter the degrees it represents halves: $45, 22.5, 11.25, 6.125, \ldots$.
  - Return $\min(d_2 - d_1, 360 + d_1 - d_2)$.
- **Remark:** Only 29 characters are needed for the required precision.



F: Failing Flagship

Problem Author: Ruben Brokkelkamp

- **Problem:** Compute the minimum angle in degrees between two wind directions.
- **Solution:**
  - Convert both wind directions into degrees $d_1 \geq d_2$.
  - **Observation:** For every extra letter the degrees it represents halves: $45, 22.5, 11.25, 6.125, \ldots$.
  - Return $\min(d_2 - d_1, 360 + d_1 - d_2)$.
- **Remark:** Only 29 characters are needed for the required precision.



Statistics: 144 submissions, 39 accepted, 21 unknown

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.
- **Solution:** For every query, do a breadth-/depth-first search, multiplying all conversion ratios on the path from the one to the other unit.
  - Runtime: $\mathcal{O}(q \cdot n)$, which is fast enough.

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.
- **Solution:** For every query, do a breadth-/depth-first search, multiplying all conversion ratios on the path from the one to the other unit.
  - Runtime: $\mathcal{O}(q \cdot n)$, which is fast enough.
- **Faster solution:** Precalculate (or cache) the conversion ratio between all pairs of units.
  - Runtime: $\mathcal{O}(n^2)$ to precalculate and $\mathcal{O}(1)$ per query, so $\mathcal{O}(n^2 + q)$.

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.
- **Solution:** For every query, do a breadth-/depth-first search, multiplying all conversion ratios on the path from the one to the other unit.
  - Runtime: $\mathcal{O}(q \cdot n)$, which is fast enough.
- **Faster solution:** Precalculate (or cache) the conversion ratio between all pairs of units.
  - Runtime: $\mathcal{O}(n^2)$ to precalculate and $\mathcal{O}(1)$ per query, so $\mathcal{O}(n^2 + q)$.
- Smallest/largest results are $10^{\pm 303}$, which fits in `double` (limit: $9 \cdot 10^{\pm 307}$).

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.
- **Solution:** For every query, do a breadth-/depth-first search, multiplying all conversion ratios on the path from the one to the other unit.
  - Runtime: $\mathcal{O}(q \cdot n)$, which is fast enough.
- **Faster solution:** Precalculate (or cache) the conversion ratio between all pairs of units.
  - Runtime: $\mathcal{O}(n^2)$ to precalculate and $\mathcal{O}(1)$ per query, so $\mathcal{O}(n^2 + q)$.
- Smallest/largest results are $10^{\pm 303}$, which fits in `double` (limit: $9 \cdot 10^{\pm 307}$).
- **Pitfall:** In C++, people used `cout << fixed << setprecision(12)`. However, for values smaller than 10^{-6}, this gives too low (*relative*) precision...

I: Imperfect Imperial Units

Problem Author: Abe Wits

- **Problem:** Convert values from one unit to another.
- **Solution:** For every query, do a breadth-/depth-first search, multiplying all conversion ratios on the path from the one to the other unit.
  - Runtime: $\mathcal{O}(q \cdot n)$, which is fast enough.
- **Faster solution:** Precalculate (or cache) the conversion ratio between all pairs of units.
  - Runtime: $\mathcal{O}(n^2)$ to precalculate and $\mathcal{O}(1)$ per query, so $\mathcal{O}(n^2 + q)$.
- Smallest/largest results are $10^{\pm 303}$, which fits in `double` (limit: $9 \cdot 10^{\pm 307}$).
- **Pitfall:** In C++, people used `cout << fixed << setprecision(12)`. However, for values smaller than 10^{-6}, this gives too low (*relative*) precision...

Statistics: 163 submissions, 31 accepted, 69 unknown

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).
- **Greedy solution:** Start with $[0, 1)$, and keep growing the interval until it is allowed, say $[0, a)$.

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).
- **Greedy solution:** Start with $[0, 1)$, and keep growing the interval until it is allowed, say $[0, a)$.
- Then start a new interval $[a, a+1)$, and keep growing it until it is allowed, say $[a, b)$.

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).
- **Greedy solution:** Start with $[0, 1)$, and keep growing the interval until it is allowed, say $[0, a)$.
- Then start a new interval $[a, a + 1)$, and keep growing it until it is allowed, say $[a, b)$.
- Continue until reaching the end. This uses exactly n queries in total.

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).
- **Greedy solution:** Start with $[0, 1)$, and keep growing the interval until it is allowed, say $[0, a)$.
- Then start a new interval $[a, a + 1)$, and keep growing it until it is allowed, say $[a, b)$.
- Continue until reaching the end. This uses exactly n queries in total.

D: Dividing DNA

Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a set of *forbidden* (*present*) intervals, partition $[0, n)$ into as many disjoint (*absent*) intervals as possible.
- **Observation 1:** If an interval is forbidden, all shorter intervals are also forbidden!
- **Observation 2:** Each interval only needs to be *just* long enough to be allowed (*absent*).
- **Greedy solution:** Start with $[0, 1)$, and keep growing the interval until it is allowed, say $[0, a)$.
- Then start a new interval $[a, a+1)$, and keep growing it until it is allowed, say $[a, b)$.
- Continue until reaching the end. This uses exactly n queries in total.

Statistics: 32 submissions, 10 accepted, 17 unknown

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.
- Time to code any x consecutive characters (without saving) =

  time to write $x - 1$ characters $+$ 1 $+$ expected time needed to recover from crashing:

$$T(x) = T(x-1) + 1 + p \cdot (r + T(x)) = \frac{T(x-1) + 1 + p \cdot r}{1-p} \quad \textbf{or} \quad T(x) = \frac{r+1}{p} \cdot ((1-p)^{-x} - 1)$$

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.
- Time to code any x consecutive characters (without saving) =
  time to write $x-1$ characters + 1 + expected time needed to recover from crashing:

$$T(x) = T(x-1)+1+p\cdot(r+T(x)) = \frac{T(x-1)+1+p\cdot r}{1-p} \quad \textbf{or} \quad T(x) = \frac{r+1}{p}\cdot((1-p)^{-x}-1)$$

- Calculate time to code all characters between position 0 and x,
  minimising the total time by trying to click "Save" after character k:

$$DP(x) = \min\left(T(x), \min_{0\le k\le x}\left(DP(k)+t+T(x-k)\right)\right)$$

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.
- Time to code any x consecutive characters (without saving) =
  time to write $x-1$ characters + 1 + expected time needed to recover from crashing:

$$T(x) = T(x-1)+1+p\cdot(r+T(x)) = \frac{T(x-1)+1+p\cdot r}{1-p} \quad \textbf{or} \quad T(x) = \frac{r+1}{p}\cdot((1-p)^{-x}-1)$$

- Calculate time to code all characters between position 0 and x,
  minimising the total time by trying to click “Save” after character k:

$$DP(x) = \min\left(T(x), \min_{0\leq k\leq x}\left(DP(k)+t+T(x-k)\right)\right)$$

- Final answer is $DP(c)+t$ (because we should save the code at the end, costing t time).

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.
- Time to code any x consecutive characters (without saving) $=$

  time to write $x-1$ characters $+\ 1\ +$ expected time needed to recover from crashing:

$$T(x) = T(x-1)+1+p\cdot(r+T(x)) = \frac{T(x-1)+1+p\cdot r}{1-p} \quad \textbf{or} \quad T(x) = \frac{r+1}{p}\cdot((1-p)^{-x}-1)$$

- Calculate time to code all characters between position 0 and x,

  minimising the total time by trying to click "Save" after character k:

$$DP(x) = \min\left(T(x), \min_{0\leq k\leq x}\left(DP(k)+t+T(x-k)\right)\right)$$

- Final answer is $DP(c)+t$ (because we should save the code at the end, costing t time).
- Run-time complexity: $\mathcal{O}(c^2)$

C: Crashing Competition Computer

Problem Author: Jorke de Vlas

- **Problem:** What is the expected time to write c characters of code on a crash-prone computer?
- **Solution:** Dynamic programming over the length of the code.
- Time to code any x consecutive characters (without saving) $=$
  time to write $x-1$ characters $+$ 1 $+$ expected time needed to recover from crashing:

$$T(x) = T(x-1)+1+p\cdot(r+T(x)) = \frac{T(x-1)+1+p\cdot r}{1-p} \quad \textbf{or} \quad T(x) = \frac{r+1}{p}\cdot((1-p)^{-x}-1)$$

- Calculate time to code all characters between position 0 and x,
  minimising the total time by trying to click "Save" after character k:

$$DP(x) = \min\left(T(x), \min_{0\le k\le x}\left(DP(k)+t+T(x-k)\right)\right)$$

- Final answer is $DP(c)+t$ (because we should save the code at the end, costing t time).
- Run-time complexity: $\mathcal{O}(c^2)$

Statistics: 56 submissions, 10 accepted, 35 unknown

K: Kiosk Construction

Problem Author: Reinier Schmiermann

- **Problem:** Find the optimal kiosk position for a given camping layout.

K: Kiosk Construction

Problem Author: Reinier Schmiermann

- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k, p)$), then calculate $\min_k(\max_p(d(k, p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).

K: Kiosk Construction

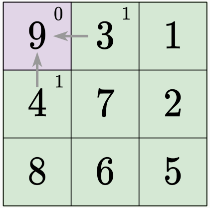
Problem Author: Reinier Schmiermann

- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k, p)$), then calculate $\min_k(\max_p(d(k, p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.
  - The rules of walking between plots are fixed given a destination plot p, so do floodfill (BFS/DFS) starting from every destination plot p.
  - From a plot a, walk to neighbouring plots b if, according to the procedure, you can walk from b to a given the destination plot p.

K: Kiosk Construction

Problem Author: Reinier Schmiermann

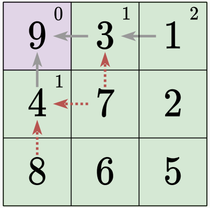
- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k, p)$),
  then calculate $\min_k(\max_p(d(k, p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.



K: Kiosk Construction

Problem Author: Reinier Schmiermann

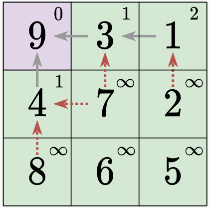
- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k,p)$), then calculate $\min_k(\max_p(d(k,p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.



K: Kiosk Construction

Problem Author: Reinier Schmiermann

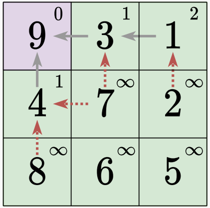
- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k, p)$), then calculate $\min_k(\max_p(d(k, p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.



K: Kiosk Construction

Problem Author: Reinier Schmiermann

- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k,p)$), then calculate $\min_k(\max_p(d(k,p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.

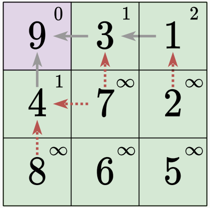


- Run-time complexity: $\mathcal{O}(n^2)$ (with $n = h \cdot w$).

K: Kiosk Construction

Problem Author: Reinier Schmiermann

- **Problem:** Find the optimal kiosk position for a given camping layout.
- **Solution:** Find the shortest path from every kiosk location k to every plot p ($d(k,p)$), then calculate $\min_k(\max_p(d(k,p)))$.
  - But, doing n^2 times BFS/DFS from every possible kiosk location to every plot is too slow ($\mathcal{O}(n^3)$).
- **Optimisation:** Calculate distances the other way around: from every plot to every kiosk location.



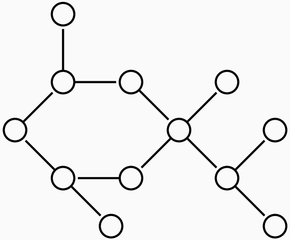
- Run-time complexity: $\mathcal{O}(n^2)$ (with $n = h \cdot w$).

Statistics: 31 submissions, 8 accepted, 15 unknown

H: House Numbering

Problem Author: Reinier Schmiermann

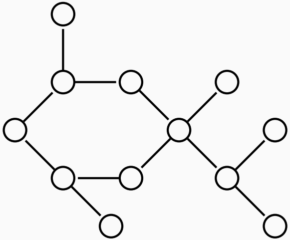
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.



H: House Numbering

Problem Author: Reinier Schmiermann

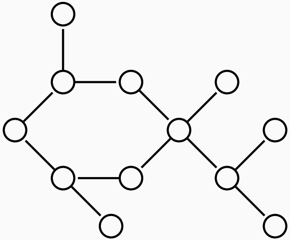
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.



H: House Numbering

Problem Author: Reinier Schmiermann

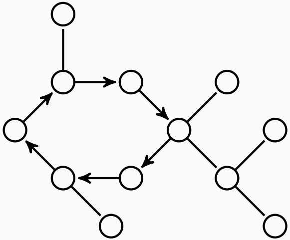
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.
- **Observation 2:** Because the number of nodes is equal to the number of edges the graph contains exactly one cycle.



H: House Numbering

Problem Author: Reinier Schmiermann

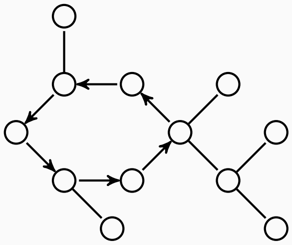
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.
- **Observation 2:** Because the number of nodes is equal to the number of edges the graph contains exactly one cycle.
- **Observation 3:** The cycle has to be oriented clockwise or counterclockwise.



H: House Numbering

Problem Author: Reinier Schmiermann

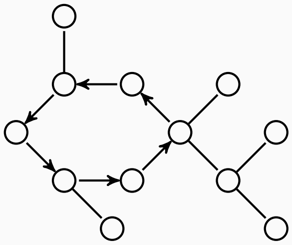
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.
- **Observation 2:** Because the number of nodes is equal to the number of edges the graph contains exactly one cycle.
- **Observation 3:** The cycle has to be oriented clockwise or counterclockwise.



H: House Numbering

Problem Author: Reinier Schmiermann

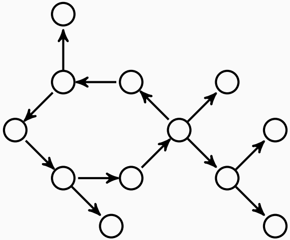
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.
- **Observation 2:** Because the number of nodes is equal to the number of edges the graph contains exactly one cycle.
- **Observation 3:** The cycle has to be oriented clockwise or counterclockwise.
- **Observation 4:** The cycle has trees attached to it for which the house number 1 has to face outward.



H: House Numbering

Problem Author: Reinier Schmiermann

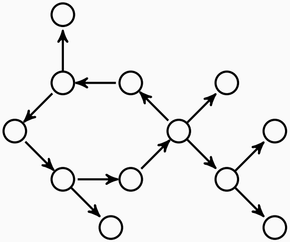
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Observation 1:** Every node can have at most one edge with house number 1 starting at it.
- **Observation 2:** Because the number of nodes is equal to the number of edges the graph contains exactly one cycle.
- **Observation 3:** The cycle has to be oriented clockwise or counterclockwise.
- **Observation 4:** The cycle has trees attached to it for which the house number 1 has to face outward.



H: House Numbering

Problem Author: Reinier Schmiermann

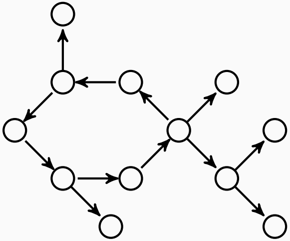
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**



H: House Numbering

Problem Author: Reinier Schmiermann

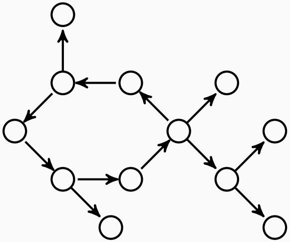
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**
  - First, find the cycle.



H: House Numbering

Problem Author: Reinier Schmiermann

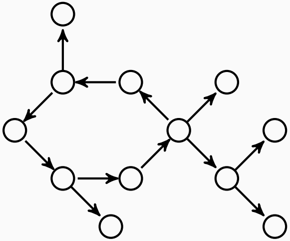
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**
  - First, find the cycle.
  - Orient cycle clockwise and propagate house numbers through trees. Check if this is valid, i.e., there are no two equal high house numbers at a node.



H: House Numbering

Problem Author: Reinier Schmiermann

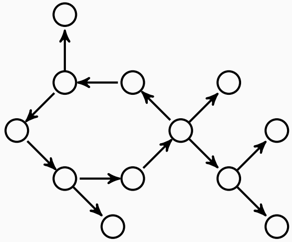
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**
  - First, find the cycle.
  - Orient cycle clockwise and propagate house numbers through trees. Check if this is valid, i.e., there are no two equal high house numbers at a node.
  - If not ok, orient cycle counter clockwise and check validity.



H: House Numbering

Problem Author: Reinier Schmiermann

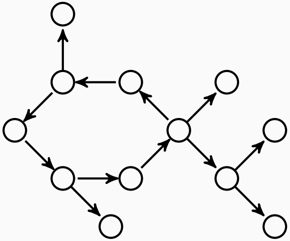
- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**
  - First, find the cycle.
  - Orient cycle clockwise and propagate house numbers through trees. Check if this is valid, i.e., there are no two equal high house numbers at a node.
  - If not ok, orient cycle counter clockwise and check validity.
  - If one of the two works, print it. Otherwise, print "impossible".



H: House Numbering

Problem Author: Reinier Schmiermann

- **Problem:** Given a graph with n nodes and edges, and h house numbers for an edge, determine whether house numbers can be assigned such that there is no intersection where two edges start with the same house number.
- **Solution:**
  - First, find the cycle.
  - Orient cycle clockwise and propagate house numbers through trees. Check if this is valid, i.e., there are no two equal high house numbers at a node.
  - If not ok, orient cycle counter clockwise and check validity.
  - If one of the two works, print it. Otherwise, print "impossible".



Statistics: 31 submissions, 5 accepted, 15 unknown

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most 12 000 queries: "`Is integer` h_i `less than` y`?`"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most 12 000 queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!
- **Better idea:** First test if a column is better than the current best. If not, skip it.

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!
- **Better idea:** First test if a column is better than the current best. If not, skip it.
  - Problem: The maximum could increase in each column!

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!
- **Better idea:** First test if a column is better than the current best. If not, skip it.
  - Problem: The maximum could increase in each column!
- **Solution:** Randomize the order of columns! Now, only $\ln(w)$ increments are expected!

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y?"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!
- **Better idea:** First test if a column is better than the current best. If not, skip it.
  - Problem: The maximum could increase in each column!
- **Solution:** Randomize the order of columns! Now, only $\ln(w)$ increments are expected!
- Total expected queries: $n + \ln(w) \cdot \log_2(n)$.

J: Jagged Skyline

Problem Author: Reinier Schmiermann

- **Problem:** Given $w \leq 10\,000$ integers $0 \leq h_i \leq 10^{18}$, find the maximum in at most $12\,000$ queries: "`Is integer` h_i `less than` y`?`"
- **First idea:** binary search per column: $n \cdot \log_2(n)$ queries.
  - Only search the range above the current maximum?
  - In fact, we can only do 1.2 query per column. Most columns must be handled in 1 query exactly!
- **Better idea:** First test if a column is better than the current best. If not, skip it.
  - Problem: The maximum could increase in each column!
- **Solution:** Randomize the order of columns! Now, only $\ln(w)$ increments are expected!
- Total expected queries: $n + \ln(w) \cdot \log_2(n)$.

Statistics: 140 submissions, 7 accepted, 80 unknown

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce
  - Split the volume into $100 \times 100 \times 100$ boxes of size $10^7 \times 10^7 \times 10^7$, and iterate over the pairs in each box.

---

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce
  - Split the volume into $100 \times 100 \times 100$ boxes of size $10^7 \times 10^7 \times 10^7$, and iterate over the pairs in each box.
  - Problem: The minimum distance may cross a boundary between boxes.

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce
  - Split the volume into $100 \times 100 \times 100$ boxes of size $10^7 \times 10^7 \times 10^7$, and iterate over the pairs in each box.
  - Problem: The minimum distance may cross a boundary between boxes.
  - Solution: Iterate over all pairs of points in touching boxes as well.

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce
  - Split the volume into $100 \times 100 \times 100$ boxes of size $10^7 \times 10^7 \times 10^7$, and iterate over the pairs in each box.
  - Problem: The minimum distance may cross a boundary between boxes.
  - Solution: Iterate over all pairs of points in touching boxes as well.
  - Expected running time: $\mathcal{O}(k \cdot (n/k)^2 + k) = \mathcal{O}(n^2/k + k)$, where k is the number of boxes.

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

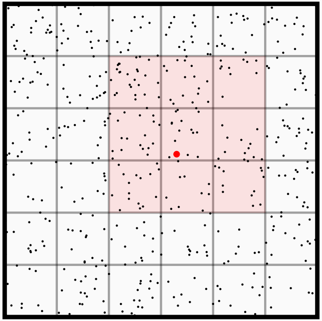
- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- Naive quadratic solution is too slow, and sweepline methods are complicated!
- To get a faster solution, make use of the fact that the input is random.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6. [1]
- **Solution:** Local bruteforce
  - Split the volume into $100 \times 100 \times 100$ boxes of size $10^7 \times 10^7 \times 10^7$, and iterate over the pairs in each box.
  - Problem: The minimum distance may cross a boundary between boxes.
  - Solution: Iterate over all pairs of points in touching boxes as well.
  - Expected running time: $\mathcal{O}(k \cdot (n/k)^2 + k) = \mathcal{O}(n^2/k + k)$, where k is the number of boxes.
  - Note: due to the birthday paradox, there will practically always be a box with at least 2 points.

[1] Or at least, *almost always* ;-)

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Solution:** Local bruteforce



L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.
- **Alternative solution 2:** Sorted bruteforce.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.
- **Alternative solution 2:** Sorted bruteforce.
  - Sort the points by x. The average x-distance between two points is $10^9/n = 10^4$.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.
- **Alternative solution 2:** Sorted bruteforce.
  - Sort the points by x. The average x-distance between two points is $10^9/n = 10^4$.
  - Points > 100 positions apart are expected to have distance $> 100 \cdot 10^4 = 10^6$.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.
- **Alternative solution 2:** Sorted bruteforce.
  - Sort the points by x. The average x-distance between two points is $10^9/n = 10^4$.
  - Points > 100 positions apart are expected to have distance $> 100 \cdot 10^4 = 10^6$.
  - Consider all pairs of indices (i, j) with $|i - j| \leq 100$ for an $\mathcal{O}(100n)$ solution.

L: Lowest Latency

Problem Author: Reinier Schmiermann

- **Problem:** Given $n = 10^5$ random points in $V = [0, 10^9)^3$, find the shortest distance between two points.
- **Observation:** Because the points are i.i.d. uniformly random, the answer is less than 10^6.
- **Alternative solution 1:** Divide & Conquer.
  - Sort the points by x and split into two groups of size $n/2$. Solve the two halves by recursively splitting in half-sized groups.
  - Let d be the best solution found recursively. Consider all pairs of points within distance d from the boundary to handle the *merge* step.
- **Alternative solution 2:** Sorted bruteforce.
  - Sort the points by x. The average x-distance between two points is $10^9/n = 10^4$.
  - Points > 100 positions apart are expected to have distance $> 100 \cdot 10^4 = 10^6$.
  - Consider all pairs of indices (i, j) with $|i - j| \leq 100$ for an $\mathcal{O}(100n)$ solution.

Statistics: 63 submissions, 10 accepted, 35 unknown

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell / \ell!)$ time.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell / \ell!)$ time.
- For $\ell = 3, 4$: this is too slow so use *meet-in-the-middle*:

$P_u = \{a_i + a_j \mid i < j < u\}$	a_u	

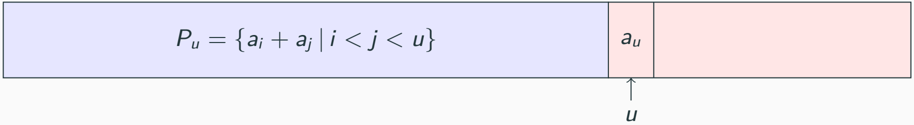
$\uparrow$

u

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell / \ell!)$ time.
- For $\ell = 3, 4$: this is too slow so use *meet-in-the-middle*:

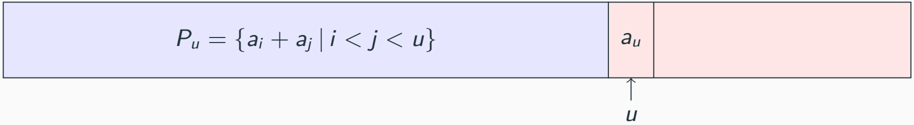


- For $\ell = 3$: loop over $u = 1, \ldots, n$, update P_u and then take $s \in P_u$ closest to $S_\ell - a_u$.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell / \ell!)$ time.
- For $\ell = 3, 4$: this is too slow so use *meet-in-the-middle*:

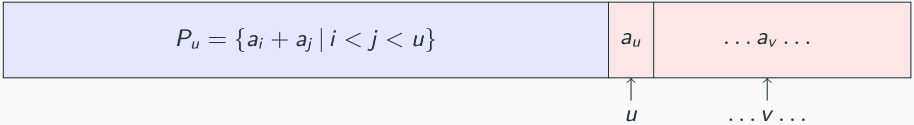


- For $\ell = 3$: loop over $u = 1, \ldots, n$, update P_u and then take $s \in P_u$ closest to $S_\ell - a_u$.
- Use an ordered set (BBST) for P_u, giving the time complexity $\mathcal{O}(n^2 \log n)$.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell / \ell!)$ time.
- For $\ell = 3, 4$: this is too slow so use *meet-in-the-middle*:

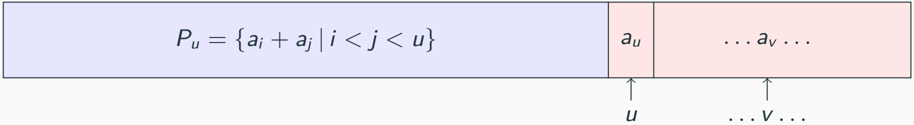


- For $\ell = 3$: loop over $u = 1, \ldots, n$, update P_u and then take $s \in P_u$ closest to $S_\ell - a_u$.
- Use an ordered set (BBST) for P_u, giving the time complexity $\mathcal{O}(n^2 \log n)$.
- For $\ell = 4$: For fixed u, loop over v with $v > u$ and pick $s \in P_u$ closest to $S_\ell - a_u - a_v$. This is still $\mathcal{O}(n^2 \log n)$.

A: Adjusted Average

Problem Author: Ludo Pulles

- **Problem:** Given $n \leq 1500$ integers a_i, remove at most $k \leq 4$ of them to get an average as close as possible to the target $\overline{x}$.
- When removing exactly $0 \leq \ell \leq k$ numbers, we need to find ℓ integers with sum as close as possible to $S_\ell = \sum_i a_i - \ell \cdot \overline{x}$.
- For $\ell \leq 2$: iterate over all combinations $\longrightarrow \mathcal{O}(n^\ell/\ell!)$ time.
- For $\ell = 3, 4$: this is too slow so use *meet-in-the-middle*:



- For $\ell = 3$: loop over $u = 1, \ldots, n$, update P_u and then take $s \in P_u$ closest to $S_\ell - a_u$.
- Use an ordered set (BBST) for P_u, giving the time complexity $\mathcal{O}(n^2 \log n)$.
- For $\ell = 4$: For fixed u, loop over v with $v > u$ and pick $s \in P_u$ closest to $S_\ell - a_u - a_v$. This is still $\mathcal{O}(n^2 \log n)$.

Statistics: 25 submissions, 3 accepted, 13 unknown

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \dots, c_{k-1}]$, the counts for each remainder.
    - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.
  - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.
  - **Simpler alternative:** Merge the largest remainder with another one, and update the state. $\rightarrow$ Too slow when counts are $1 \times 4, 30 \times 5, 30 \times 6, 30 \times 7$.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.
  - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.
  - **Simpler alternative:** Merge the largest remainder with another one, and update the state. $\rightarrow$ Too slow when counts are $1 \times 4, 30 \times 5, 30 \times 6, 30 \times 7$.
  - **Instead:** merge the least-occurring element with one of the others.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.
  - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.
  - **Simpler alternative:** Merge the largest remainder with another one, and update the state. $\rightarrow$ Too slow when counts are $1 \times 4, 30 \times 5, 30 \times 6, 30 \times 7$.
  - **Instead:** merge the least-occurring element with one of the others.
  - **Even simpler:** remove any one of the remaining elements. If this makes the total sum be $0 \mod k$, add one.

G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.
  - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.
  - **Simpler alternative:** Merge the largest remainder with another one, and update the state. $\rightarrow$ Too slow when counts are $1 \times 4, 30 \times 5, 30 \times 6, 30 \times 7$.
  - **Instead:** merge the least-occurring element with one of the others.
  - **Even simpler:** remove any one of the remaining elements. If this makes the total sum be $0 \mod k$, add one.
- Instead of 4-deep nested loops, we can use a dictionary of tuples.

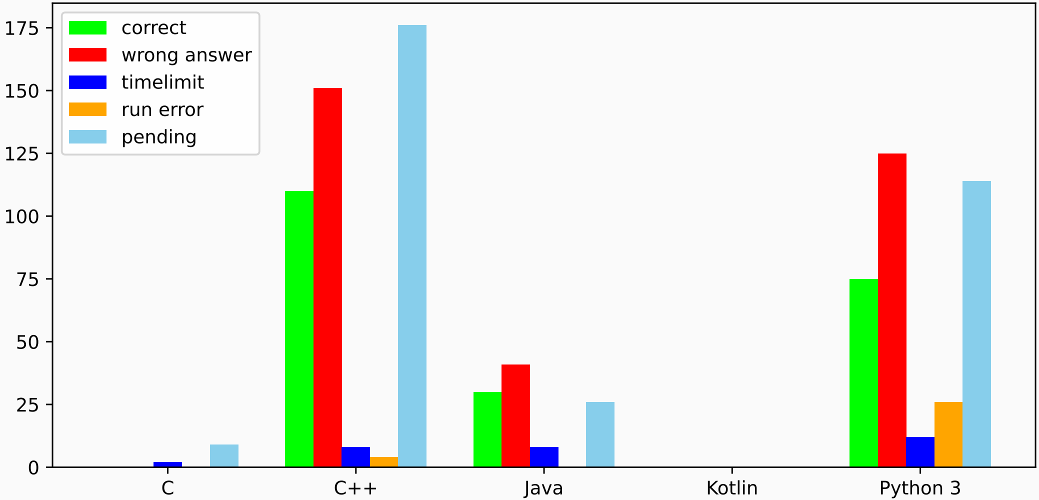
G: Grinding Gravel

Problem Author: Daan van Gent, Onno Berrevoets

- **Problem:** Given $n \leq 100$ integers, split them into groups of size $k \leq 8$ making as few cuts as possible.
- **Equivalent problem:** Given n integers, partition them into as many groups as possible with sum a multiple of k.
- Greedy 1: Each number $x \geq k$ is replaced by $x \mod k$. Count the numbers with each remainder.
- Greedy 2: For $x < k/2$, we can pair up x and $k - x$. Each $x = 0$ is its own group.
- We are left with at most 4 different values: 1 or 7, 2 or 6, 3 or 5, and at most one 4.
- Now, do a DP on state $[c_1, \ldots, c_{k-1}]$, the counts for each remainder.
  - For each precomputed (minimal) subset with sum $0 \mod k$ remove it and recurse.
  - **Simpler alternative:** Merge the largest remainder with another one, and update the state. $\rightarrow$ Too slow when counts are $1 \times 4, 30 \times 5, 30 \times 6, 30 \times 7$.
  - **Instead:** merge the least-occurring element with one of the others.
  - **Even simpler:** remove any one of the remaining elements. If this makes the total sum be $0 \mod k$, add one.
- Instead of 4-deep nested loops, we can use a dictionary of tuples.

Statistics: 5 submissions, 1 accepted, 1 unknown

Language stats



Random facts

Jury work

- 721 commits, of which 434 for the main contest

Random facts

Jury work

- 721 commits, of which 434 for the main contest
- 604 secret test cases (last year: 693) ($= 50\frac{1}{3}$ per problem!)

Random facts

Jury work

- 721 commits, of which 434 for the main contest
- 604 secret test cases (last year: 693) ($= 50\frac{1}{3}$ per problem!)
- 165 jury solutions (last year: 177)

Random facts

Jury work

- 721 commits, of which 434 for the main contest
- 604 secret test cases (last year: 693) ($= 50\frac{1}{3}$ per problem!)
- 165 jury solutions (last year: 177)
- The minimum[2] number of lines the jury needed to solve all problems is

$$14 + 3 + 5 + 1 + 4 + 4 + 27 + 34 + 14 + 15 + 18 + 4 = 143$$

On average 11.9 lines per problem, up from 9.6 in BAPC 2021 or 6.6 in preliminaries 2022

[2] *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

Want to join the jury? Submit to the Call for Problems of BAPC 2023 at:

`https://jury.bapc.eu/`