Freshmen Programming Contests 2025

Solutions presentation

By the Freshmen Programming Contests 2025 jury for:

- AAPJE in Amsterdam
- FPC in Delft
- FYPC in Eindhoven
- GAPC in Groningen
- Contest in Mons

May 3, 2025

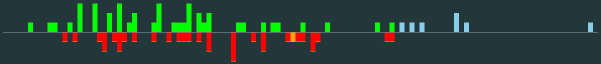
Other universities will have their contests in the coming weeks.

Please, do not post/discuss the problems online before

Saturday 17 May 2025 at 17:00

G: Gambler’s Dilemma

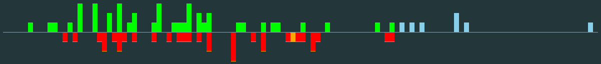
Problem author: Wietze Koops



Problem: Determine whether two playing cards have any of the four given properties.

G: Gambler’s Dilemma

Problem author: Wietze Koops

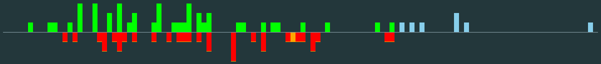


Problem: Determine whether two playing cards have any of the four given properties.

Solution: For each property, check whether the cards match it.

G: Gambler's Dilemma

Problem author: Wietze Koops



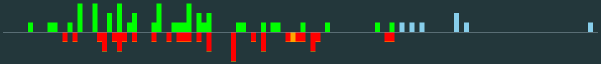
Problem: Determine whether two playing cards have any of the four given properties.

Solution: For each property, check whether the cards match it.

Pitfall: Be careful of off-by-one errors when calculating the rank of a card.

G: Gambler's Dilemma

Problem author: Wietze Koops



Problem: Determine whether two playing cards have any of the four given properties.

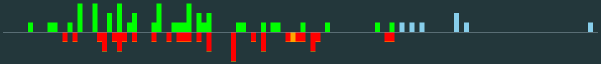
Solution: For each property, check whether the cards match it.

Pitfall: Be careful of off-by-one errors when calculating the rank of a card.

Running time: $\mathcal{O}(1)$.

G: Gambler's Dilemma

Problem author: Wietze Koops



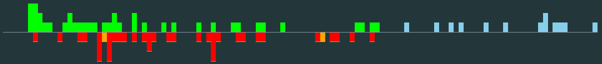
Problem: Determine whether two playing cards have any of the four given properties.

Solution: For each property, check whether the cards match it.

Pitfall: Be careful of off-by-one errors when calculating the rank of a card.

Running time: $\mathcal{O}(1)$.

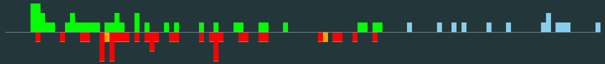
Statistics: 82 submissions, 42 accepted, 7 unknown



Problem: Minimize the area of one rectangle that cannot overlap with another.

B: Bakfiets

Problem author: Jeroen Op de Beek

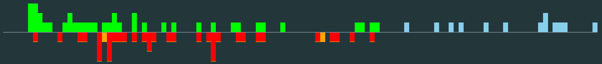


Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

B: Bakfiets

Problem author: Jeroen Op de Beek



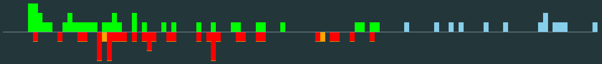
Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

B: Bakfiets

Problem author: Jeroen Op de Beek



Problem: Minimize the area of one rectangle that cannot overlap with another.

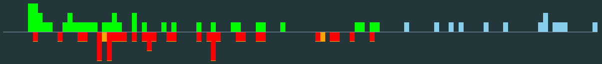
Observation 1: The bottle packaging can be in two possible orientations.

Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.

B: Bakfiets

Problem author: Jeroen Op de Beek

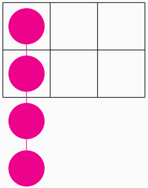


Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

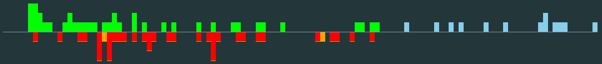
Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.



B: Bakfiets

Problem author: Jeroen Op de Beek

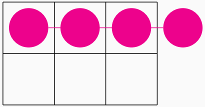


Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

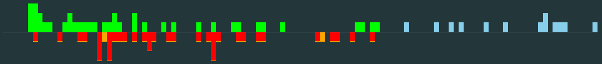
Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.



B: Bakfiets

Problem author: Jeroen Op de Beek



Problem: Minimize the area of one rectangle that cannot overlap with another.

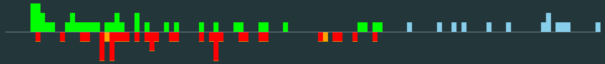
Observation 1: The bottle packaging can be in two possible orientations.

Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.

B: Bakfiets

Problem author: Jeroen Op de Beek



Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

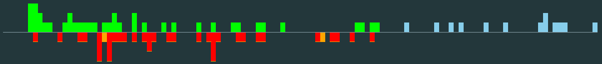
Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.

Running time: $\mathcal{O}(1)$.

B: Bakfiets

Problem author: Jeroen Op de Beek



Problem: Minimize the area of one rectangle that cannot overlap with another.

Observation 1: The bottle packaging can be in two possible orientations.

Observation 2: You can have at most 10^{18} bottles, hence we need 64-bit integers.

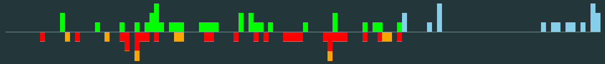
Solution: Compute $w \cdot h - \max(\min(w, a) \cdot \min(h, b), \min(w, b) \cdot \min(h, a))$.

Running time: $\mathcal{O}(1)$.

Statistics: 90 submissions, 39 accepted, 13 unknown

J: Jumbled Keys

Problem author: Arnoud van der Leer



Problem: Decipher a message, using a series of mapped words.

J: Jumbled Keys

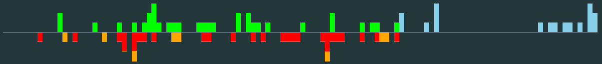
Problem author: Arnoud van der Leer

Problem: Decipher a message, using a series of mapped words.

Solution: Use a map! Process every pair of words, and map every letter in the first word to the letter it corresponds to in the second word.

J: Jumbled Keys

Problem author: Arnoud van der Leer



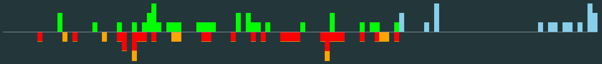
Problem: Decipher a message, using a series of mapped words.

Solution: Use a map! Process every pair of words, and map every letter in the first word to the letter it corresponds to in the second word.

Edge case: If 25 letters are mapped, the 26th letter maps to the only letter that has no other letter mapped to it.

J: Jumbled Keys

Problem author: Arnoud van der Leer



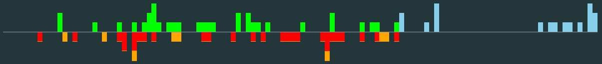
Problem: Decipher a message, using a series of mapped words.

Solution: Use a map! Process every pair of words, and map every letter in the first word to the letter it corresponds to in the second word.

Edge case: If 25 letters are mapped, the 26th letter maps to the only letter that has no other letter mapped to it.

Running time: $\mathcal{O}(n \cdot \ell)$, where ℓ is the average length of the words.

J: Jumbled Keys



Problem author: Arnoud van der Leer

Problem: Decipher a message, using a series of mapped words.

Solution: Use a map! Process every pair of words, and map every letter in the first word to the letter it corresponds to in the second word.

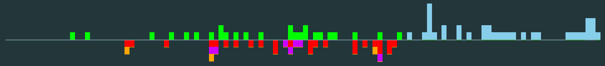
Edge case: If 25 letters are mapped, the 26th letter maps to the only letter that has no other letter mapped to it.

Running time: $\mathcal{O}(n \cdot \ell)$, where ℓ is the average length of the words.

Statistics: 86 submissions, 33 accepted, 17 unknown

H: Hopelessly Hungover

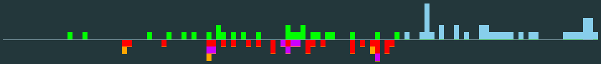
Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

H: Hopelessly Hungover

Problem author: Wietze Koops

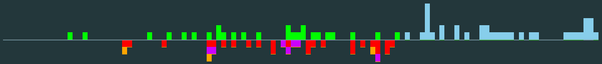


Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

H: Hopelessly Hungover

Problem author: Wietze Koops



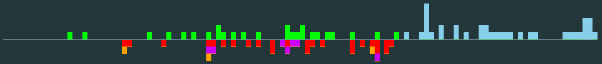
Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

H: Hopelessly Hungover

Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

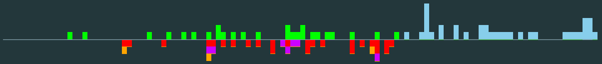
Observation 1: The input limits are very small, all integers are up to a 1000.

Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

Solution: We can simulate the process for $7n + 14$ days. Whenever the number of facts is n we stop. After $7n + 14$ days we can be sure that we will never learn all n facts.

H: Hopelessly Hungover

Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

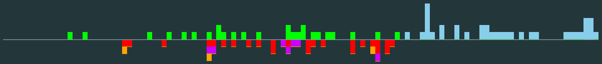
Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

Solution: We can simulate the process for $7n + 14$ days. Whenever the number of facts is n we stop. After $7n + 14$ days we can be sure that we will never learn all n facts.

Fun fact: The actual maximum number of days you have to wait is 6976.

H: Hopelessly Hungover

Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

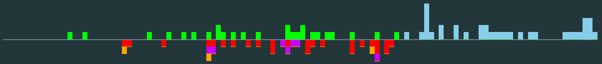
Solution: We can simulate the process for $7n + 14$ days. Whenever the number of facts is n we stop. After $7n + 14$ days we can be sure that we will never learn all n facts.

Fun fact: The actual maximum number of days you have to wait is 6976.

Running time: $\mathcal{O}(7n)$ (the 7 is to signify the dependence on the number of days in the week).

H: Hopelessly Hungover

Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

Solution: We can simulate the process for $7n + 14$ days. Whenever the number of facts is n we stop. After $7n + 14$ days we can be sure that we will never learn all n facts.

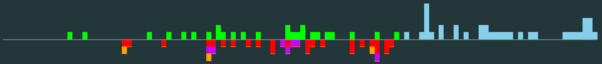
Fun fact: The actual maximum number of days you have to wait is 6976.

Running time: $\mathcal{O}(7n)$ (the 7 is to signify the dependence on the number of days in the week).

Pitfall: It is possible to solve this problem in $O(1)$ by case-working on the ending day, and using formulas involving ceil division. This solution is prone to mistakes.

H: Hopelessly Hungover

Problem author: Wietze Koops



Problem: Each weekday you can learn k facts, each day in the weekend you forget m facts. Given a starting day of the week, how many days do you need to wait before you know all n facts?

Observation 1: The input limits are very small, all integers are up to a 1000.

Observation 2: In the worstcase in which we can eventually know n facts, the net effect of one week is adding 1 fact. So after $7n + O(1)$ days we can give up.

Solution: We can simulate the process for $7n + 14$ days. Whenever the number of facts is n we stop. After $7n + 14$ days we can be sure that we will never learn all n facts.

Fun fact: The actual maximum number of days you have to wait is 6976.

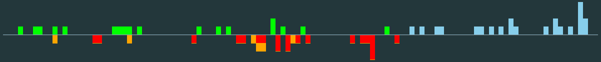
Running time: $\mathcal{O}(7n)$ (the 7 is to signify the dependence on the number of days in the week).

Pitfall: It is possible to solve this problem in $O(1)$ by case-working on the ending day, and using formulas involving ceil division. This solution is prone to mistakes.

Statistics: 97 submissions, 26 accepted, 36 unknown

A: Array Annihilation

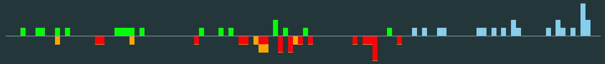
Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

A: Array Annihilation

Problem author: Leon van der Waal

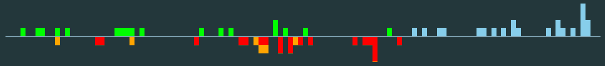


Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

Observation: Every value in the array needs to be decremented along with either of its neighbours. So for each position i, it is necessary that $a_i \geq a_{i-1} + a_{i+1}$.

A: Array Annihilation

Problem author: Leon van der Waal



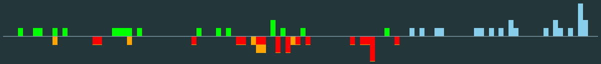
Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

Observation: Every value in the array needs to be decremented along with either of its neighbours. So for each position i, it is necessary that $a_i \geq a_{i-1} + a_{i+1}$.

Solution: If this inequality holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

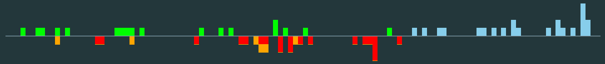
Observation: Every value in the array needs to be decremented along with either of its neighbours. So for each position i, it is necessary that $a_i \geq a_{i-1} + a_{i+1}$.

Solution: If this inequality holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Pitfall: Be sure to handle the first and last value in the array: $a_1 \leq a_2$ and $a_{n-1} \geq a_n$.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

Observation: Every value in the array needs to be decremented along with either of its neighbours. So for each position i, it is necessary that $a_i \geq a_{i-1} + a_{i+1}$.

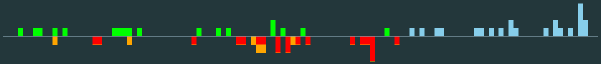
Solution: If this inequality holds for each triplet of positions in the array: “`possible`”. Else: “`impossible`”.

Pitfall: Be sure to handle the first and last value in the array: $a_1 \leq a_2$ and $a_{n-1} \geq a_n$.

Running time: $\mathcal{O}(n)$.

A: Array Annihilation

Problem author: Leon van der Waal



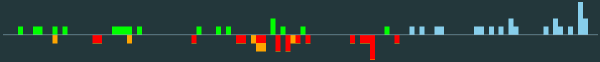
Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

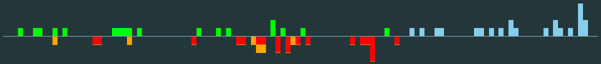
Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

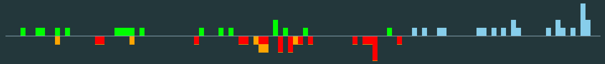
Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.
- The total number of length-2, length-3 segments starting at 1 is a_1.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

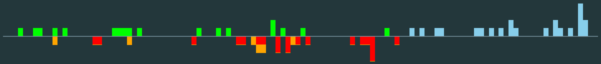
Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.
- The total number of length-2, length-3 segments starting at 1 is a_1.
- Those segments cover $[1, 2]$ or $[1, 2, 3]$. This means a_2 -= a_1, a_3 -= δ, where $\delta \in [0, a_1]$.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

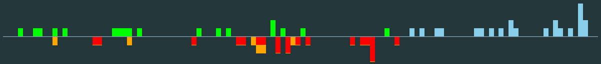
Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.
- The total number of length-2, length-3 segments starting at 1 is a_1.
- Those segments cover $[1, 2]$ or $[1, 2, 3]$. This means a_2 -= a_1, a_3 -= δ, where $\delta \in [0, a_1]$.
- If the original $a_1, \ldots, a_n$ satisfied condition, one can select δ such that $a_2 - a_1, a_3 - \delta, a_4, \ldots, a_n$ satisfies the condition.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

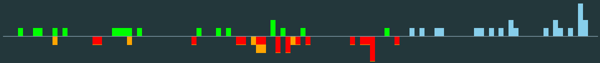
Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.
- The total number of length-2, length-3 segments starting at 1 is a_1.
- Those segments cover $[1, 2]$ or $[1, 2, 3]$. This means $a_2 \mathrel{-}= a_1$, $a_3 \mathrel{-}= \delta$, where $\delta \in [0, a_1]$.
- If the original $a_1, \ldots, a_n$ satisfied condition, one can select δ such that $a_2 - a_1, a_3 - \delta, a_4, \ldots, a_n$ satisfies the condition.
- Using induction, the condition is sufficient.

A: Array Annihilation

Problem author: Leon van der Waal



Problem: Given an array, is it possible to make all values equal to 0 when repeatedly decrementing two or more consecutive values in the array.

Solution: If $a_i \geq a_{i-1} + a_{i+1}$ holds for each triplet of positions in the array: "`possible`". Else: "`impossible`".

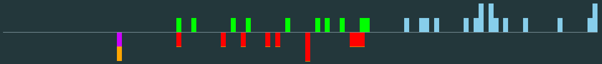
Proof: It's clear the condition is necessary. We need to show that it's sufficient.

- It's sufficient to restrict use to length-2 and length-3 segments only.
- The total number of length-2, length-3 segments starting at 1 is a_1.
- Those segments cover $[1, 2]$ or $[1, 2, 3]$. This means $a_2 \mathrel{-}= a_1$, $a_3 \mathrel{-}= \delta$, where $\delta \in [0, a_1]$.
- If the original $a_1, \ldots, a_n$ satisfied condition, one can select δ such that $a_2 - a_1, a_3 - \delta, a_4, \ldots, a_n$ satisfies the condition.
- Using induction, the condition is sufficient.

Statistics: 66 submissions, 18 accepted, 22 unknown

D: Delicious Trees

Problem author: Jeroen Op de Beek



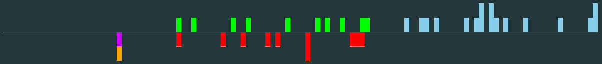
Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

D: Delicious Trees

Problem author: Jeroen Op de Beek



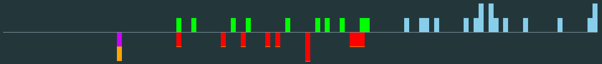
Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

D: Delicious Trees

Problem author: Jeroen Op de Beek



Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

Observation 3: Assuming we can split the big tree into n trees of size 1,
the answer is only "`impossible`" when $k > n$.

D: Delicious Trees

Problem author: Jeroen Op de Beek

Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

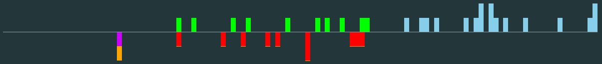
Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

Observation 3: Assuming we can split the big tree into n trees of size 1, the answer is only "`impossible`" when $k > n$.

Solution: First calculate the depth of each vertex in the tree using BFS/DFS, then remove vertices one-by-one from largest to smallest depth.

D: Delicious Trees

Problem author: Jeroen Op de Beek



Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

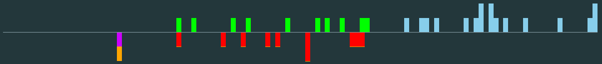
Observation 3: Assuming we can split the big tree into n trees of size 1, the answer is only "`impossible`" when $k > n$.

Solution: First calculate the depth of each vertex in the tree using BFS/DFS, then remove vertices one-by-one from largest to smallest depth.

Red herring: The first sample cuts off larger AVL trees on purpose.

D: Delicious Trees

Problem author: Jeroen Op de Beek



Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

Observation 3: Assuming we can split the big tree into n trees of size 1, the answer is only "`impossible`" when $k > n$.

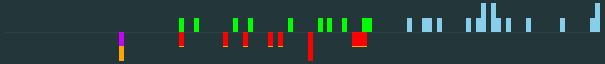
Solution: First calculate the depth of each vertex in the tree using BFS/DFS, then remove vertices one-by-one from largest to smallest depth.

Red herring: The first sample cuts off larger AVL trees on purpose.

Running time: Dominated by sorting by depth: $\mathcal{O}(n \log n)$.

D: Delicious Trees

Problem author: Jeroen Op de Beek



Problem: Find any way to cut the AVL tree into some predetermined number of smaller AVL trees, or say this is impossible.

Observation 1: An AVL tree with only one vertex, is also an AVL tree.

Observation 2: Removing the deepest vertex from the tree can only decrease the depth of the largest of the two subtrees of any ancestor, so this will never introduce imbalanced vertices.

Observation 3: Assuming we can split the big tree into n trees of size 1, the answer is only "`impossible`" when $k > n$.

Solution: First calculate the depth of each vertex in the tree using BFS/DFS, then remove vertices one-by-one from largest to smallest depth.

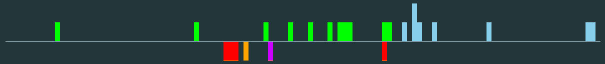
Red herring: The first sample cuts off larger AVL trees on purpose.

Running time: Dominated by sorting by depth: $\mathcal{O}(n \log n)$.

Statistics: 39 submissions, 10 accepted, 17 unknown

E: Equation Extrapolation

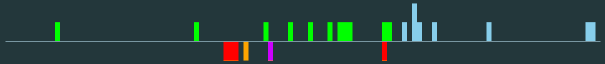
Problem author: Jeroen Op de Beek



Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

E: Equation Extrapolation

Problem author: Jeroen Op de Beek

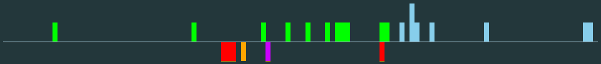


Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

Observation: Since each coefficient a_i satisfies $0 \leq a_i \leq 9$, querying at any $x > 9$ returns all coefficients directly in base-x.

E: Equation Extrapolation

Problem author: Jeroen Op de Beek



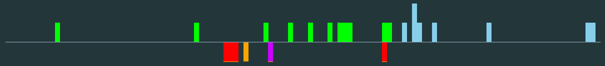
Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

Observation: Since each coefficient a_i satisfies $0 \leq a_i \leq 9$, querying at any $x > 9$ returns all coefficients directly in base-x.

Solution: Query with $x = 10$, then extract the coefficients by reading the returned number's digits in reverse.

E: Equation Extrapolation

Problem author: Jeroen Op de Beek



Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

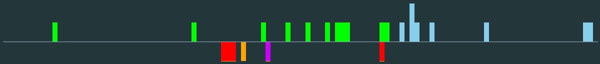
Observation: Since each coefficient a_i satisfies $0 \leq a_i \leq 9$, querying at any $x > 9$ returns all coefficients directly in base-x.

Solution: Query with $x = 10$, then extract the coefficients by reading the returned number's digits in reverse.

Running time: $\mathcal{O}(d)$.

E: Equation Extrapolation

Problem author: Jeroen Op de Beek



Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

Observation: Since each coefficient a_i satisfies $0 \leq a_i \leq 9$, querying at any $x > 9$ returns all coefficients directly in base-x.

Solution: Query with $x = 10$, then extract the coefficients by reading the returned number's digits in reverse.

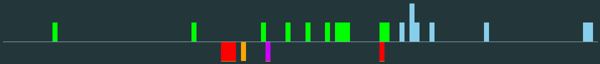
Running time: $\mathcal{O}(d)$.

Fun fact: Can be solved in one very short line of Python!

```
print("!", *input("?  10\n")[::-1])
```

E: Equation Extrapolation

Problem author: Jeroen Op de Beek



Problem: Recover the coefficients of a hidden polynomial $P(x)$ using at most 9 queries.

Observation: Since each coefficient a_i satisfies $0 \leq a_i \leq 9$, querying at any $x > 9$ returns all coefficients directly in base-x.

Solution: Query with $x = 10$, then extract the coefficients by reading the returned number's digits in reverse.

Running time: $\mathcal{O}(d)$.

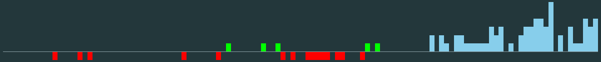
Fun fact: Can be solved in one very short line of Python!

```
print("!", *input("?  10\n")[::-1])
```

Statistics: 25 submissions, 11 accepted, 8 unknown

F: Frog and Princess

Problem author: Wietze Koops



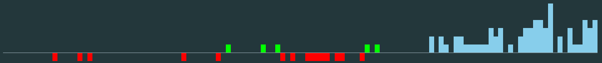
Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Observation 1: Only Euclidean distance matters: $d(f, p) = \sqrt{(x_p - x_f)^2 + (y_p - y_f)^2}$

F: Frog and Princess

Problem author: Wietze Koops



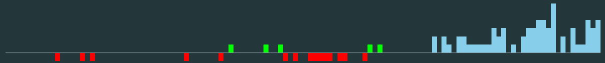
Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Observation 1: Only Euclidean distance matters: $d(f,p) = \sqrt{(x_p - x_f)^2 + (y_p - y_f)^2}$

Observation 2: If there exists a valid path, then the lengths $d(f,p), a_1, a_2, \ldots, a_k$ can form the sides of a polygon, for some k.

F: Frog and Princess

Problem author: Wietze Koops



Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Observation 1: Only Euclidean distance matters: $d(f, p) = \sqrt{(x_p - x_f)^2 + (y_p - y_f)^2}$

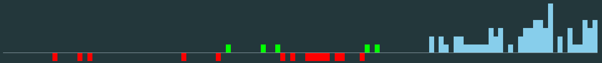
Observation 2: If there exists a valid path, then the lengths $d(f, p), a_1, a_2, \ldots, a_k$ can form the sides of a polygon, for some k.

Observation 3: Arbitrary integers $b_1, b_2, \ldots, b_k$ can form a polygon iff

$$2 \cdot \max\left(b_1, b_2, \ldots, b_k\right) \leq b_1 + b_2 + \cdots + b_k$$

F: Frog and Princess

Problem author: Wietze Koops



Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Observation 1: Only Euclidean distance matters: $d(f,p) = \sqrt{(x_p - x_f)^2 + (y_p - y_f)^2}$

Observation 2: If there exists a valid path, then the lengths $d(f,p), a_1, a_2, \ldots, a_k$ can form the sides of a polygon, for some k.

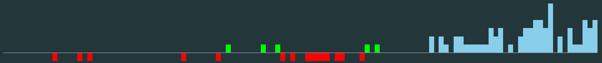
Observation 3: Arbitrary integers $b_1, b_2, \ldots, b_k$ can form a polygon iff

$$2 \cdot \max(b_1, b_2, \ldots, b_k) \leq b_1 + b_2 + \cdots + b_k$$

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

F: Frog and Princess

Problem author: Wietze Koops

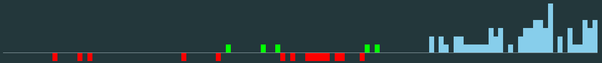


Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

F: Frog and Princess

Problem author: Wietze Koops



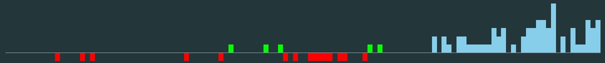
Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

Pitfall: The square root for calculation of $d(f, p)$ results in a floating point number. Using standard `double` floating-point arithmetic this is not precise enough (input into the function can go up to 10^{18} while `double`s only have a relative precision of $\approx 10^{-16}$).

F: Frog and Princess

Problem author: Wietze Koops



Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

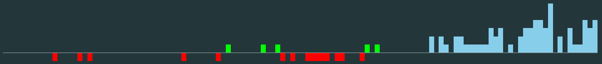
Pitfall: The square root for calculation of $d(f, p)$ results in a floating point number. Using standard `double` floating-point arithmetic this is not precise enough (input into the function can go up to 10^{18} while `doubles` only have a relative precision of $\approx 10^{-16}$).

Fixes: Use `long double` and `sqrtl` in C++, or `BigDecimal.sqrt()` in Java, or rewrite the polygon formula to use $d(f, p)^2$, which fits in a 64-bit integer:

$$d(f, p) \leq x \iff x \geq 0 \wedge d(f, p)^2 \leq x^2$$

F: Frog and Princess

Problem author: Wietze Koops



Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

Pitfall: The square root for calculation of $d(f, p)$ results in a floating point number. Using standard `double` floating-point arithmetic this is not precise enough (input into the function can go up to 10^{18} while `double`s only have a relative precision of $\approx 10^{-16}$).

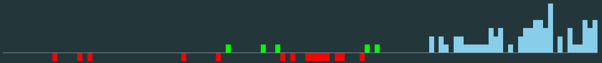
Fixes: Use `long double` and `sqrtl` in C++, or `BigDecimal.sqrt()` in Java, or rewrite the polygon formula to use $d(f, p)^2$, which fits in a 64-bit integer:

$$d(f, p) \leq x \iff x \geq 0 \wedge d(f, p)^2 \leq x^2$$

Running time: Everything can be done in linear time $\mathcal{O}(n)$.

F: Frog and Princess

Problem author: Wietze Koops



Problem: Given positions of frog and princess, find out if the frog can jump to the princess within n jumps. The ith jump should jump a distance of a_i.

Solution: Calculate prefix maximums and prefix sums of array a_i. For each prefix of length k from 1 to n, check the polygon condition and output "`yes`" if any of the checks succeed.

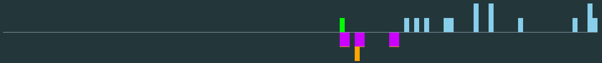
Pitfall: The square root for calculation of $d(f, p)$ results in a floating point number. Using standard `double` floating-point arithmetic this is not precise enough (input into the function can go up to 10^{18} while `doubles` only have a relative precision of $\approx 10^{-16}$).

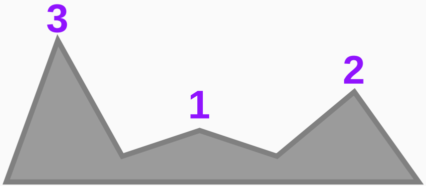
Fixes: Use `long double` and `sqrtl` in C++, or `BigDecimal.sqrt()` in Java, or rewrite the polygon formula to use $d(f, p)^2$, which fits in a 64-bit integer:

$$d(f, p) \leq x \iff x \geq 0 \land d(f, p)^2 \leq x^2$$

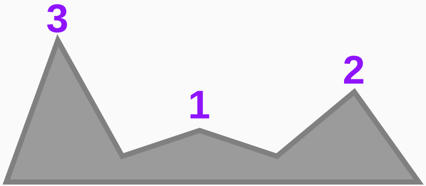
Running time: Everything can be done in linear time $\mathcal{O}(n)$.

Statistics: 86 submissions, 5 accepted, 66 unknown



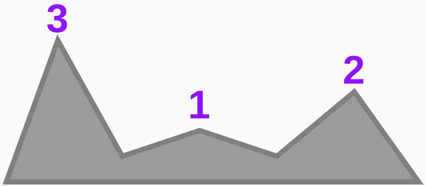


Problem: Calculate the number of “interesting formations”.



Problem: Calculate the number of “interesting formations”.

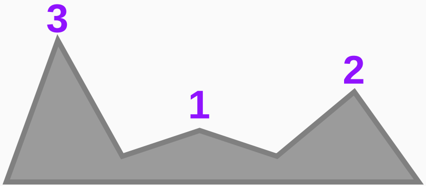
Observation 1: This is similar to counting **inversions**.



Problem: Calculate the number of “interesting formations”.

Observation 1: This is similar to counting **inversions**.

An **inversion** is a pair $i < j$ such that $h_i > h_j$.



Problem: Calculate the number of "interesting formations".

Observation 1: This is similar to counting **inversions**.

An **inversion** is a pair $i < j$ such that $h_i > h_j$.

Let's first learn how to count **inversions**.

Q: How to count **inversions**?

Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.

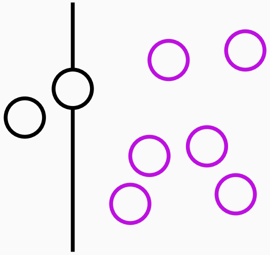
Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.



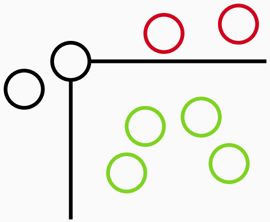
Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.



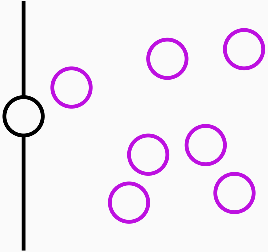
Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.



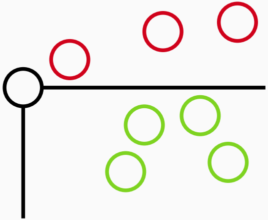
Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.



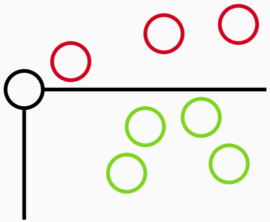
Q: How to count **inversions**?

A: We go from $i = n$ to $i = 1$ in decreasing order.

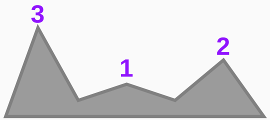
We maintain a datastructure that supports range queries.

For i, answer increases by the number of values lower than h_i in the datastructure.

After doing this, insert h_i into the datastructure.

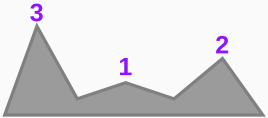


Using Fenwick tree or segment tree, $\mathcal{O}(\log(n))$ per query / update.



Problem: Calculate the number of “interesting formations”.

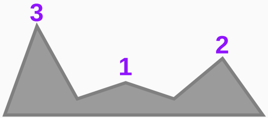
Solution: Let’s loop over i, the first / highest mountain.



Problem: Calculate the number of “interesting formations”.

Solution: Let’s loop over i, the first / highest mountain.

Suppose there are k lower mountains in our datastructure.

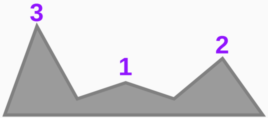


Problem: Calculate the number of “interesting formations”.

Solution: Let’s loop over i, the first / highest mountain.

Suppose there are k lower mountains in our datastructure.

We add number of pairs $\left(\frac{k(k-1)}{2}\right)$ to the answer.



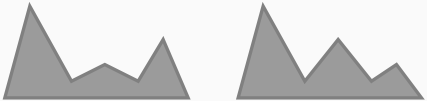
Problem: Calculate the number of "interesting formations".

Solution: Let's loop over i, the first / highest mountain.

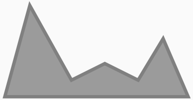
Suppose there are k lower mountains in our datastructure.

We add number of pairs $\left(\frac{k(k-1)}{2}\right)$ to the answer.

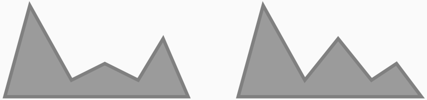
However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:

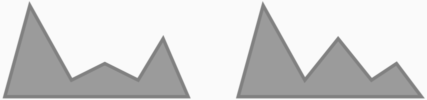


However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

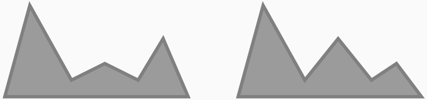
However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

We can count these by iterating over j:

However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:

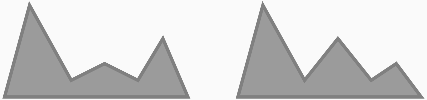


Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

We can count these by iterating over j:

We then count higher mountains to the left, and lower mountains to the right.

However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



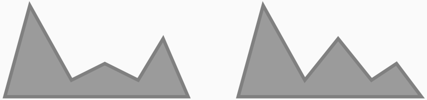
Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

We can count these by iterating over j:

We then count higher mountains to the left, and lower mountains to the right.

Then multiply and add up these counts.

However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

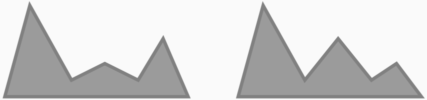
We can count these by iterating over j:

We then count higher mountains to the left, and lower mountains to the right.

Then multiply and add up these counts.

Running time: We have $\mathcal{O}(n)$ calls to a Fenwick tree / segment tree, so $\mathcal{O}(n \log(n))$ total.

However: This way, we don't distinguish between $h_i > h_k > h_j$ and $h_i > h_j > h_k$:



Solution: Therefore, we subtract the number of formations where $h_i > h_j > h_k$ from the answer.

We can count these by iterating over j:

We then count higher mountains to the left, and lower mountains to the right.

Then multiply and add up these counts.

Running time: We have $\mathcal{O}(n)$ calls to a Fenwick tree / segment tree, so $\mathcal{O}(n \log(n))$ total.

Statistics: 22 submissions, 1 accepted, 14 unknown

K: Kite Construction

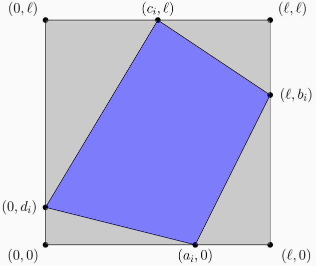
Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas.

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas.

Solution: Let $(a_i, 0)$, (ℓ, b_i), (c_i, ℓ) and $(0, d_i)$ be the coordinates of the corners of the ith quadrilateral. Then we can compute the sum of the areas by considering how much is cut off from the full $\ell \times \ell$ square:

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$



K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.

Without loss of generality assume $a_1 < a_2 < \ldots < a_n$.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < a_2 < \ldots < a_n$.

Insight: First consider minimizing $\sum_{i=1}^{n} \frac{1}{2}a_i d_i$ only. To do this, we should sort the d_i in the other order, i.e. such that $d_1 > d_2 > \ldots > d_n$.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

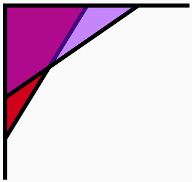
$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < a_2 < \ldots < a_n$.

Insight: First consider minimizing $\sum_{i=1}^{n} \frac{1}{2}a_i d_i$ only. To do this, we should sort the d_i in the other order, i.e. such that $d_1 > d_2 > \ldots > d_n$.

Proof: Suppose that $i < j$ (and hence $a_i < a_j$), but $d_i < d_j$. Then

$$a_i d_j + a_j d_i = a_i d_i + a_j d_j + \underbrace{(a_i - a_j)}_{>0}\underbrace{(d_j - d_i)}_{<0},$$



so swapping d_i and d_j would lead to a smaller area cut off.
Hence, whenever $i < j$ we have $d_i > d_j$.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

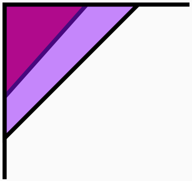
$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < a_2 < \ldots < a_n$.

Insight: First consider minimizing $\sum_{i=1}^{n} \frac{1}{2}a_i d_i$ only. To do this, we should sort the d_i in the other order, i.e. such that $d_1 > d_2 > \ldots > d_n$.

Proof: Suppose that $i < j$ (and hence $a_i < a_j$), but $d_i < d_j$. Then

$$a_i d_j + a_j d_i = a_i d_i + a_j d_j + \underbrace{(a_i - a_j)}_{>0}\underbrace{(d_j - d_i)}_{<0},$$



so swapping d_i and d_j would lead to a smaller area cut off.
Hence, whenever $i < j$ we have $d_i > d_j$.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.

Without loss of generality assume $a_1 < \ldots < a_n$.

Insight: To minimize area cut off at bottom left corner we should sort $d_1 > \ldots > d_n$.

K: Kite Construction

Problem author: Jeroen Op de Beek

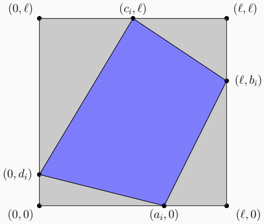
Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < \ldots < a_n$.

Insight: To minimize area cut off at bottom left corner we should sort $d_1 > \ldots > d_n$.

Then $\ell - d_1 < \ldots < \ell - d_n$, so to minimize the area cut off at the top left corner we should sort $c_1 > \ldots > c_n$.



K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

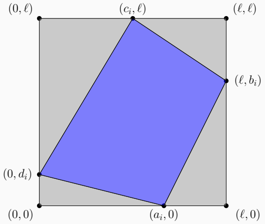
over all possible ways to order the points on each side of the square.

Without loss of generality assume $a_1 < \ldots < a_n$.

Insight: To minimize area cut off at bottom left corner we should sort $d_1 > \ldots > d_n$.

Then $\ell - d_1 < \ldots < \ell - d_n$, so to minimize the area cut off at the top left corner we should sort $c_1 > \ldots > c_n$.

Then $\ell - c_1 < \ldots < \ell - c_n$, so to minimize the area cut off at the top right corner we should sort $\ell - b_1 > \ldots > \ell - b_n$, i.e. $b_1 < \ldots < b_n$.



K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

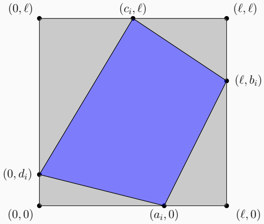
over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < \ldots < a_n$.

Insight: To minimize area cut off at bottom left corner we should sort $d_1 > \ldots > d_n$.

Then $\ell - d_1 < \ldots < \ell - d_n$, so to minimize the area cut off at the top left corner we should sort $c_1 > \ldots > c_n$.

Then $\ell - c_1 < \ldots < \ell - c_n$, so to minimize the area cut off at the top right corner we should sort $\ell - b_1 > \ldots > \ell - b_n$, i.e. $b_1 < \ldots < b_n$.

Then $b_1 < \ldots < b_n$ and $\ell - a_1 > \ldots > \ell - a_n$, so this also mimimizes the area cut off at the bottom right corner.



K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < \ldots < a_n$.

Solution: Sort the given points such that $a_1 < \ldots < a_n$, $b_1 < \ldots < b_n$, $c_1 > \ldots > c_n$, and $d_1 > \ldots > d_n$, and then compute the area with the above formula.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < \ldots < a_n$.

Solution: Sort the given points such that $a_1 < \ldots < a_n$, $b_1 < \ldots < b_n$, $c_1 > \ldots > c_n$, and $d_1 > \ldots > d_n$, and then compute the area with the above formula.

Running time: $\mathcal{O}(n \log n)$.

K: Kite Construction

Problem author: Jeroen Op de Beek

Problem: Given are $4n$ points on the perimeter of a square (with sidelength ℓ), with n points on each side. Divide these points into n quadrilaterals maximizing the sum of their areas

$$\sum_{i=1}^{n} \left(\ell^2 - \tfrac{1}{2}a_i d_i - \tfrac{1}{2}(\ell - a_i)b_i - \tfrac{1}{2}(\ell - b_i)(\ell - c_i) - \tfrac{1}{2}c_i(\ell - d_i)\right).$$

over all possible ways to order the points on each side of the square.
Without loss of generality assume $a_1 < \ldots < a_n$.

Solution: Sort the given points such that $a_1 < \ldots < a_n$, $b_1 < \ldots < b_n$, $c_1 > \ldots > c_n$, and $d_1 > \ldots > d_n$, and then compute the area with the above formula.

Running time: $\mathcal{O}(n \log n)$.

Statistics: 7 submissions, 0 accepted, 7 unknown

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

Naive solution:

- Handle each divisor d separately.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

Naive solution:

- Handle each divisor d separately.
- For each position in t', pick the character that occurs most often.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

Naive solution:

- Handle each divisor d separately.
- For each position in t', pick the character that occurs most often.

abc **A** abcabcabcabc	abc **B** abcabcabcabc	abc **B** abcabcabcabc

C: Characterithmetic

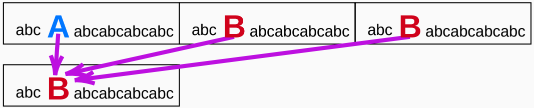
Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

Naive solution:

- Handle each divisor d separately.
- For each position in t', pick the character that occurs most often.



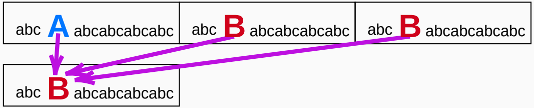
Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 1: $n \leq 10^5$, so there can be at most 128 different divisors of n.

Naive solution:

- Handle each divisor d separately.
- For each position in t', pick the character that occurs most often.



- Then count how many letters we must have changed.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

WA: But what if there is a larger divisor!?

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

WA: But what if there is a larger divisor!?

Observation 2: Then our current solution t' has a divisor $d' > 1$.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

WA: But what if there is a larger divisor!?

Observation 2: Then our current solution t' has a divisor $d' > 1$.

Observation 3: We only need to change one character to make t' indivisible.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.
- Otherwise, for each position in t' check how much more it costs to modify this letter.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.
- Otherwise, for each position in t' check how much more it costs to modify this letter.
- Greedily pick the modification that increases cost the least.

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.
- Otherwise, for each position in t' check how much more it costs to modify this letter.
- Greedily pick the modification that increases cost the least.

Running time: $\mathcal{O}(n \cdot \omega(n))$, where $\omega(n)$ is the number of divisors of n.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.
- Otherwise, for each position in t' check how much more it costs to modify this letter.
- Greedily pick the modification that increases cost the least.

Running time: $\mathcal{O}(n \cdot \omega(n))$, where $\omega(n)$ is the number of divisors of n.

Homework: $\mathcal{O}(n \log\log(n))$ is possible with some optimizations.

C: Characterithmetic

Problem author: Jeroen Op de Beek

Problem: For every divisor d of n, find the minimum number of changes to make $t = t' \odot d$ for some string t'.

Observation 3: We only need to change one character to make t' indivisible.

Correct solution:

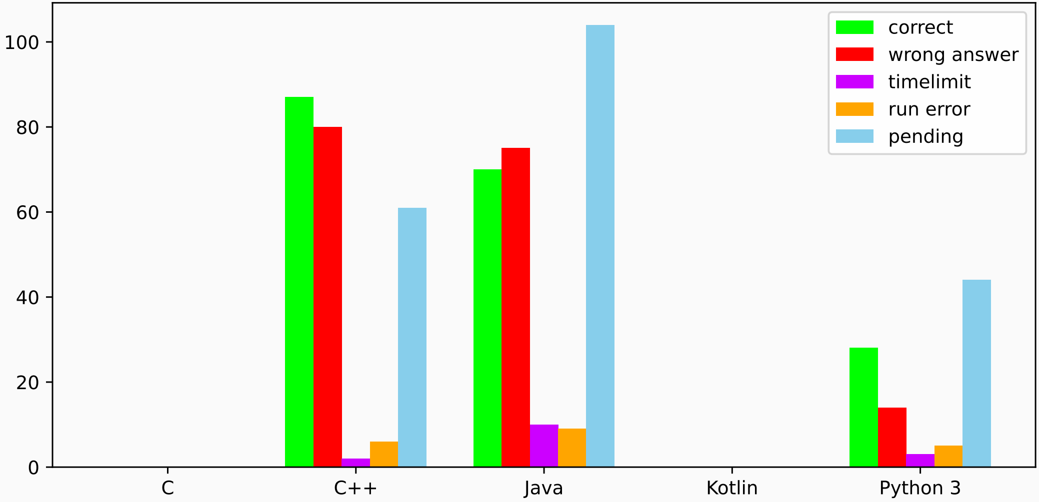
- Check if t' obtained from the naive solution is indivisible.
- In this case, we are done.
- Otherwise, for each position in t' check how much more it costs to modify this letter.
- Greedily pick the modification that increases cost the least.

Running time: $\mathcal{O}(n \cdot \omega(n))$, where $\omega(n)$ is the number of divisors of n.

Homework: $\mathcal{O}(n \log\log(n))$ is possible with some optimizations.

Statistics: 2 submissions, 0 accepted, 2 unknown

Language stats



Random facts

Jury work

- 423 commits (last year: 447)

Random facts

Jury work

- 423 commits (last year: 447)
- 625 secret test cases (last year: 357)

Random facts

Jury work

- 423 commits (last year: 447)
- 625 secret test cases (last year: 357)
- 160 accepted jury/proofreader solutions (last year: 120)

Random facts

Jury work

- 423 commits (last year: 447)
- 625 secret test cases (last year: 357)
- 160 accepted jury/proofreader solutions (last year: 120)
- The minimum[1] number of lines the jury needed to solve all problems is

$$2 + 1 + 6 + 5 + 1 + 5 + 2 + 2 + 6 + 3 + 6 = 39$$

On average 3.5 lines per problem, down from 6.0 last year

[1] *After* codegolfing

Thanks to the proofreaders:

- Arnoud van der Leer (TU Delft)
- Dany Sluijk (TU Delft)
- Davina van Meer (Delft)
- Mattia Marziali (RU Groningen)
- Michael Zündorf 🎈 (KIT Karlsruhe / NWERC jury)
- Pavel Kunyavskiy (JetBrains Amsterdam)
- Pierre Vandenhove (UMons)
- Thomas Verwoerd (TU Delft, Kotlin Hero 🎈)
- Thore Husfeldt (ITU Copenhagen / BAPC Jury)
- Wendy Yi (KIT Karlsruhe / NWERC jury)

Thanks to the Jury for the Freshmen Programming Contests:

- Alice Sayutina (VU Amsterdam)
- Angel Karchev (TU Delft)
- Bálint Kollmann (TU Delft)
- Jeroen Op de Beek (TU Delft)
- Leon van der Waal (TU Delft)
- Liudas Staniulis (VU Amsterdam)
- Maarten Sijm (TU Delft)
- Mihail Bankov (TU Delft)
- Moham Balfakeih (TU Delft)
- Wietze Koops (Radboud Nijmegen / RU Groningen)

Open online contest

Want to solve the problems you could not finish?
Or have friends that like to solve algorithmic problems?

`https://fpcs2025.bapc.eu/`

Saturday 17 May 2025 13:00–17:00

Please, do not post/discuss the problems online before this time!

Future contest fun

Excited to participate in the next contest?

Register for DAPC (20 September) at `wisv.ch/dapc`

Want to organize these contests?

Join the CHipCie: `wisv.ch/chipcieinterest`

Want to create programming problems for FPC next year?

Either join the CHipCie, or contact Maarten Sijm