Commentaries on Problems

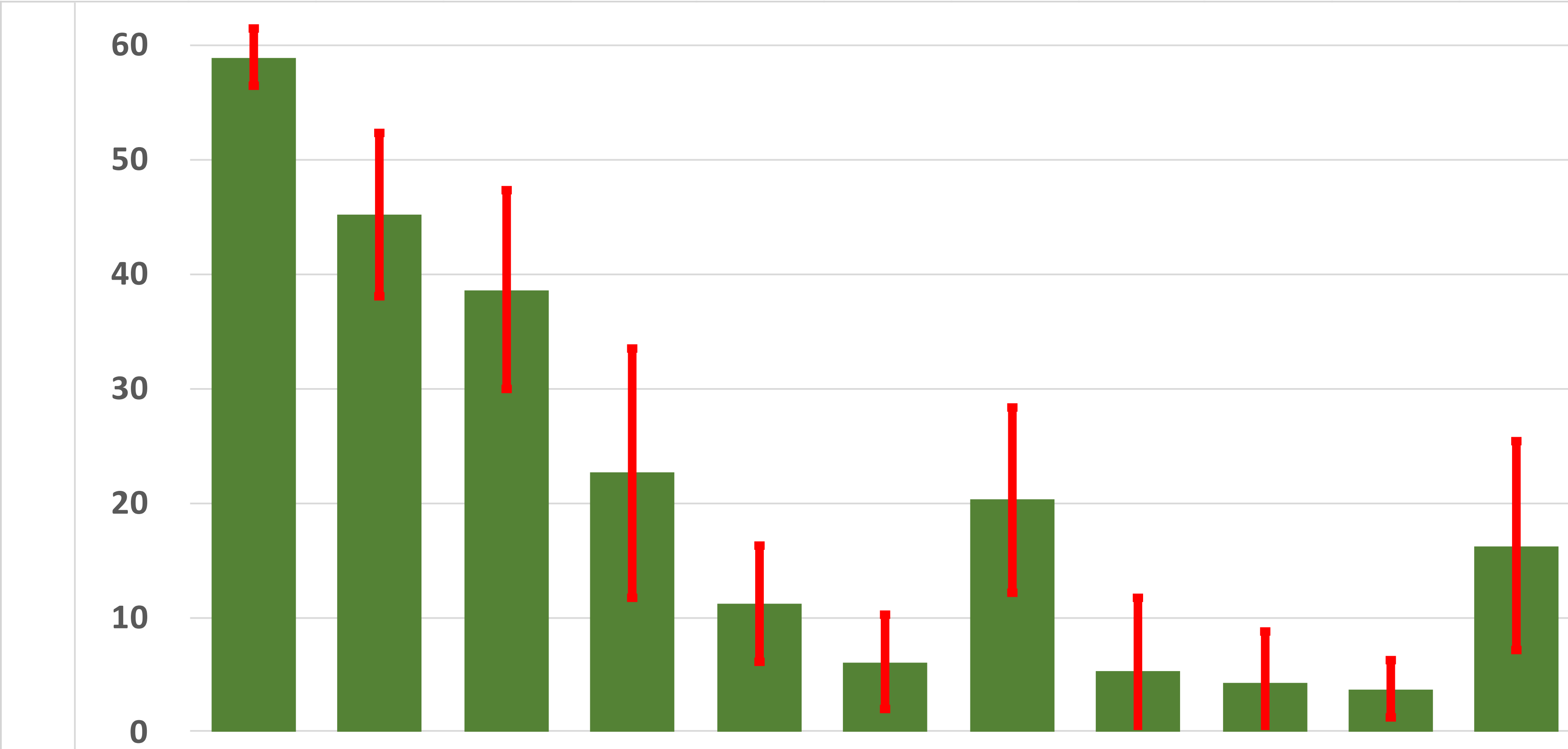
JUDGE TEAM

ICPC 2018 ASIA YOKOHAMA REGIONAL

Estimated Order of Difficulty

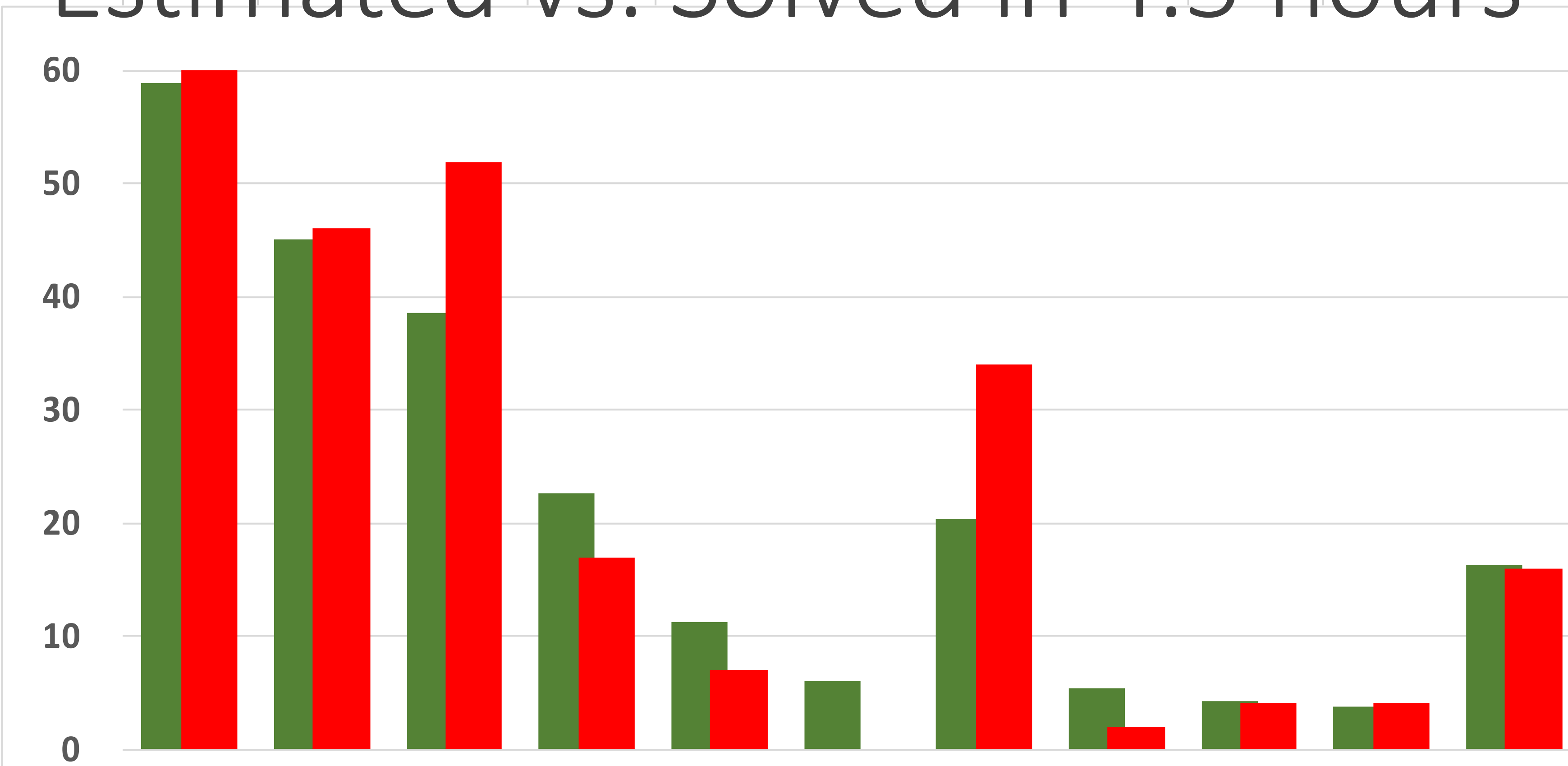
	← Easiest								Hardest →		
Coding	C	A	B	G	D	K	E	H	I	J	F
Analysis	A	B	C	G	D	K	E	F	J	H	I

Predicted # of Correct Answers



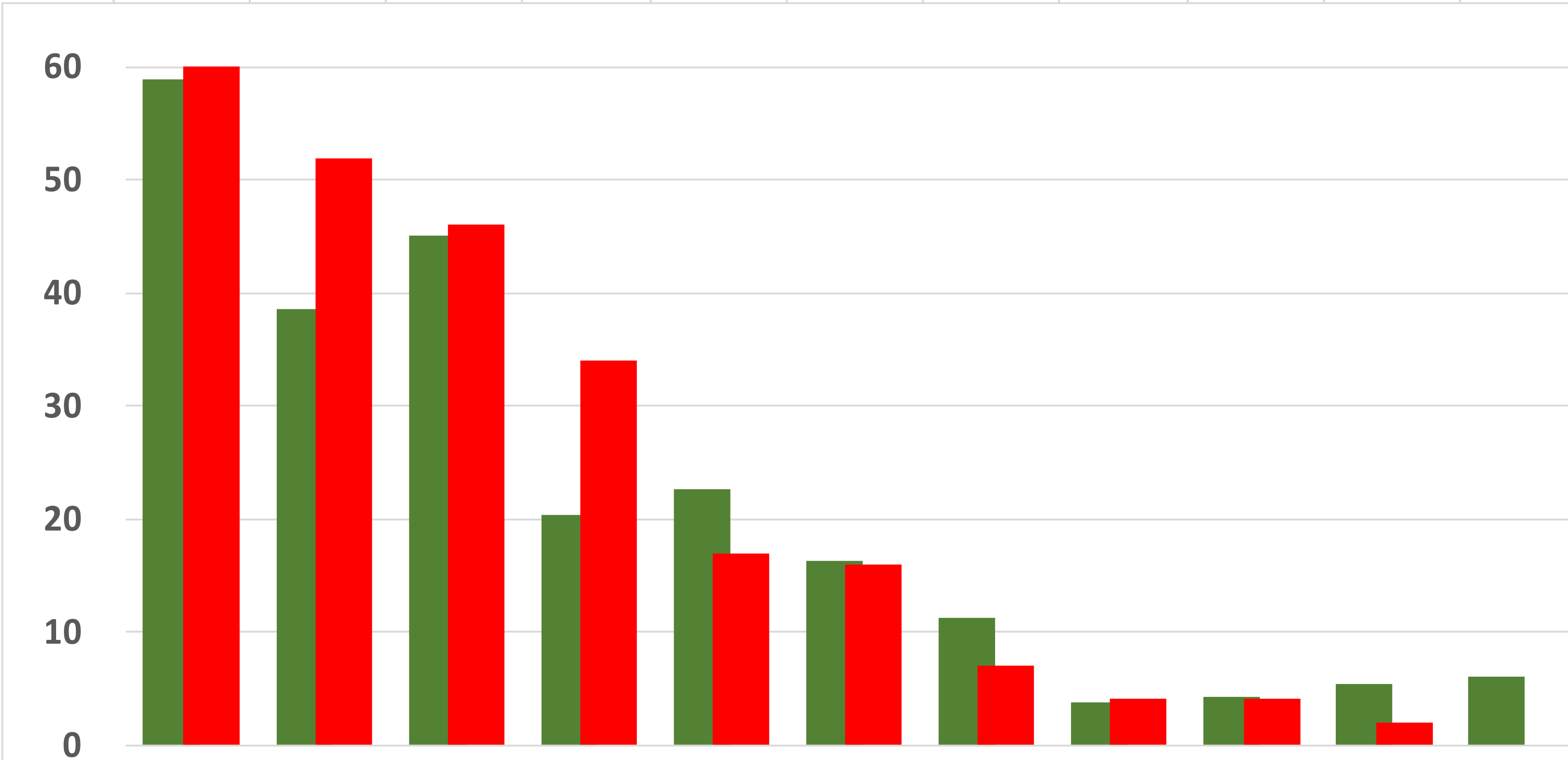
	A	B	C	D	E	F	G	H	I	J	K
Average	58.93	45.14	38.64	22.64	11.21	6.07	20.29	5.36	4.29	3.71	16.21
Std. Dev.	2.52	7.14	8.69	10.90	5.13	4.13	8.13	6.39	4.48	2.52	9.13

Estimated vs. Solved in 4.5 hours



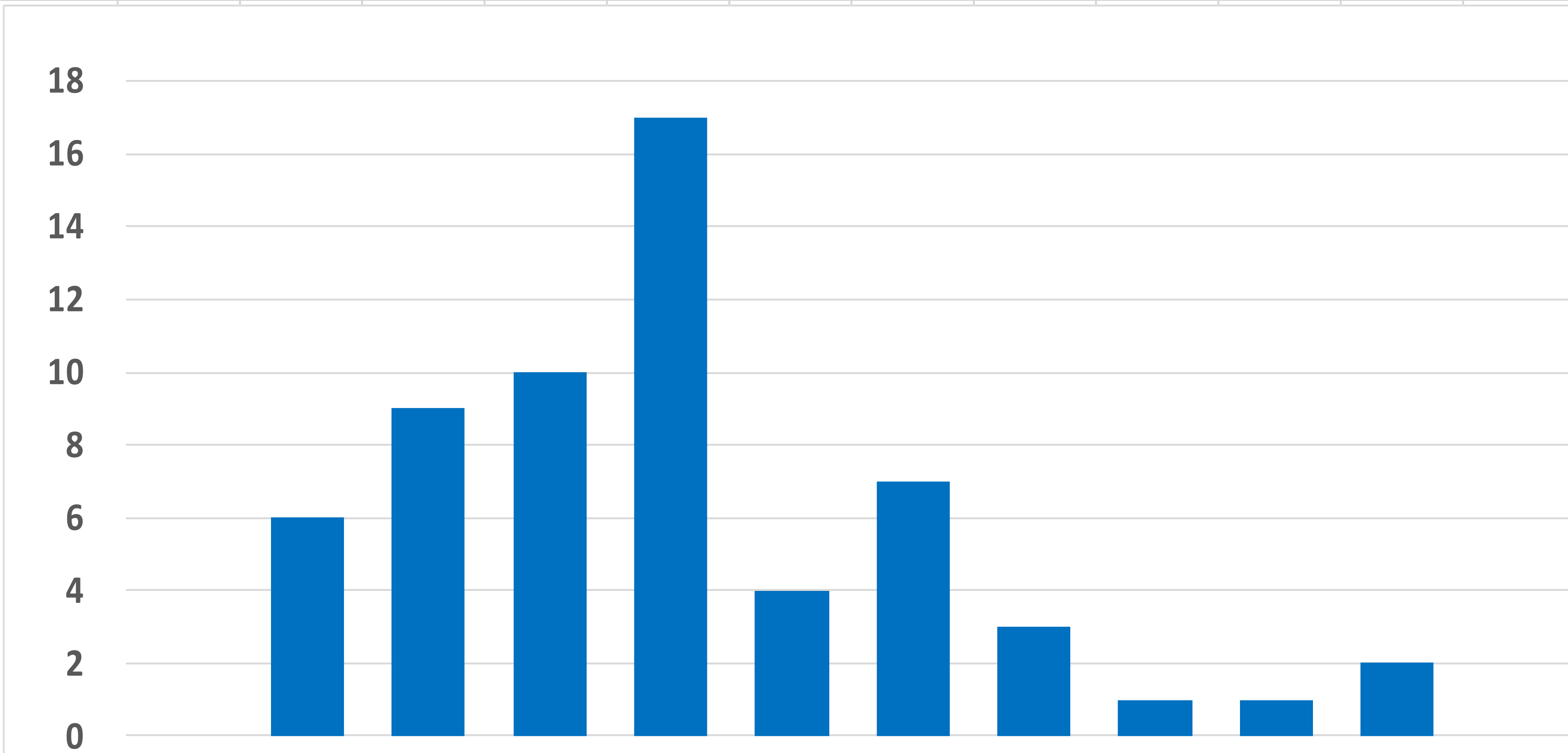
	A	B	C	D	E	F	G	H	I	J	K
Estimated	58.93	45.14	38.64	22.64	11.21	6.07	20.29	5.36	4.29	3.71	16.21
Actual	60	46	52	17	7	0	34	2	4	4	16

Estimated vs. Solved in 4.5 h.



	A	C	B	G	D	K	E	J	I	H	F
Estimated	58.93	38.64	45.14	20.29	22.64	16.21	11.21	3.71	4.29	5.36	6.07
Actual	60	52	46	34	17	16	7	4	4	2	0

problems solved & # teams



#Solved	0	1	2	3	4	5	6	7	8	9	10	11
#Teams	0	6	9	10	17	4	7	3	1	1	2	0

A:Digits Are Not Just Characters

Story

`ls` command lists file names in lexicographical order with ASCII codes:

```
$ ls
file10 file20 file3
```

But digits are not just characters.

Your task is to compare file names with numbers interpreting digit sequences as numerical values.

Solution

1. Tokenize given file names into letter items and number items.
2. Compare sequences of items lexicographically.

Be aware of the end of file names:

The end of file name comes before both of letter item and number item.

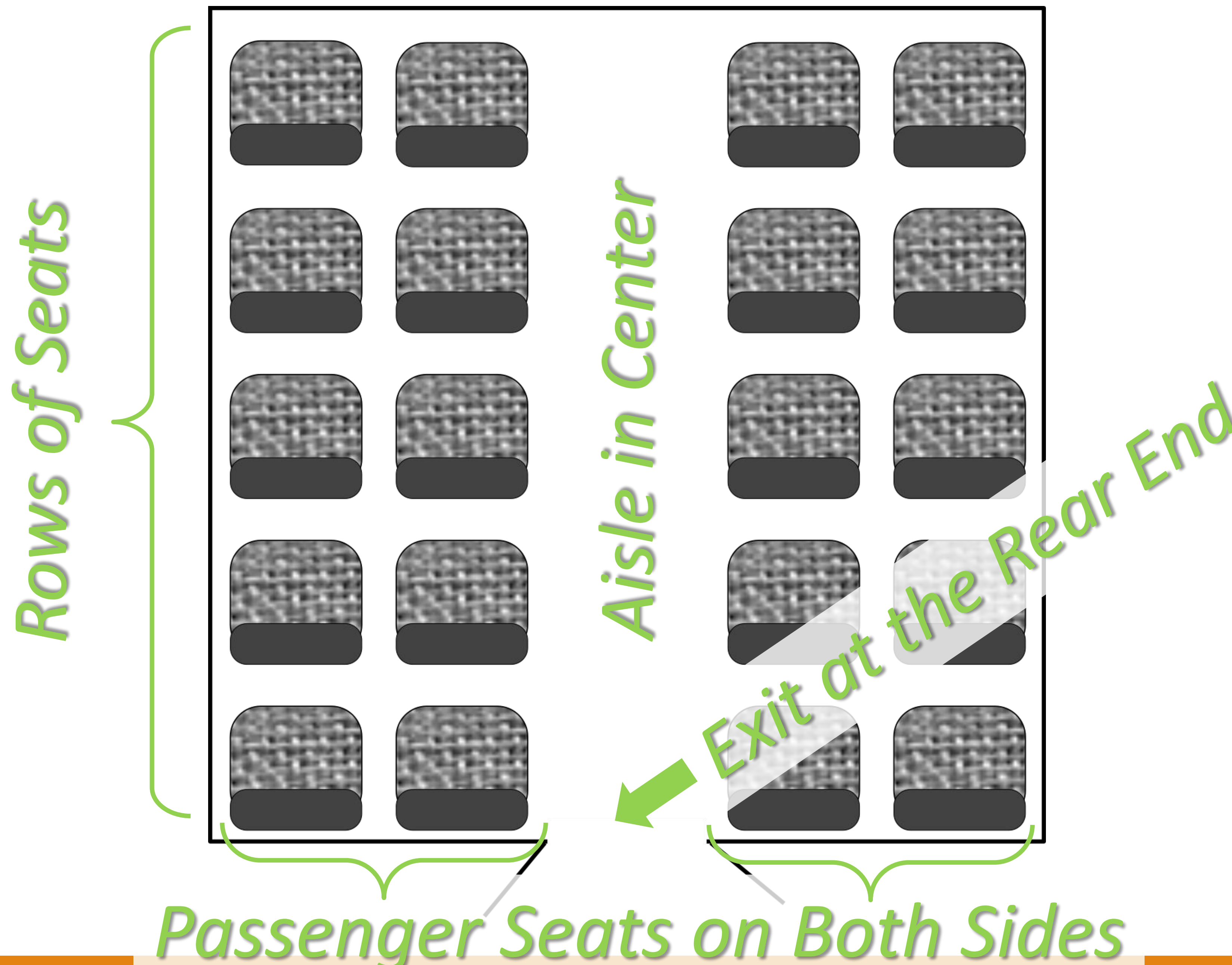
This problem was solved by all of the teams.

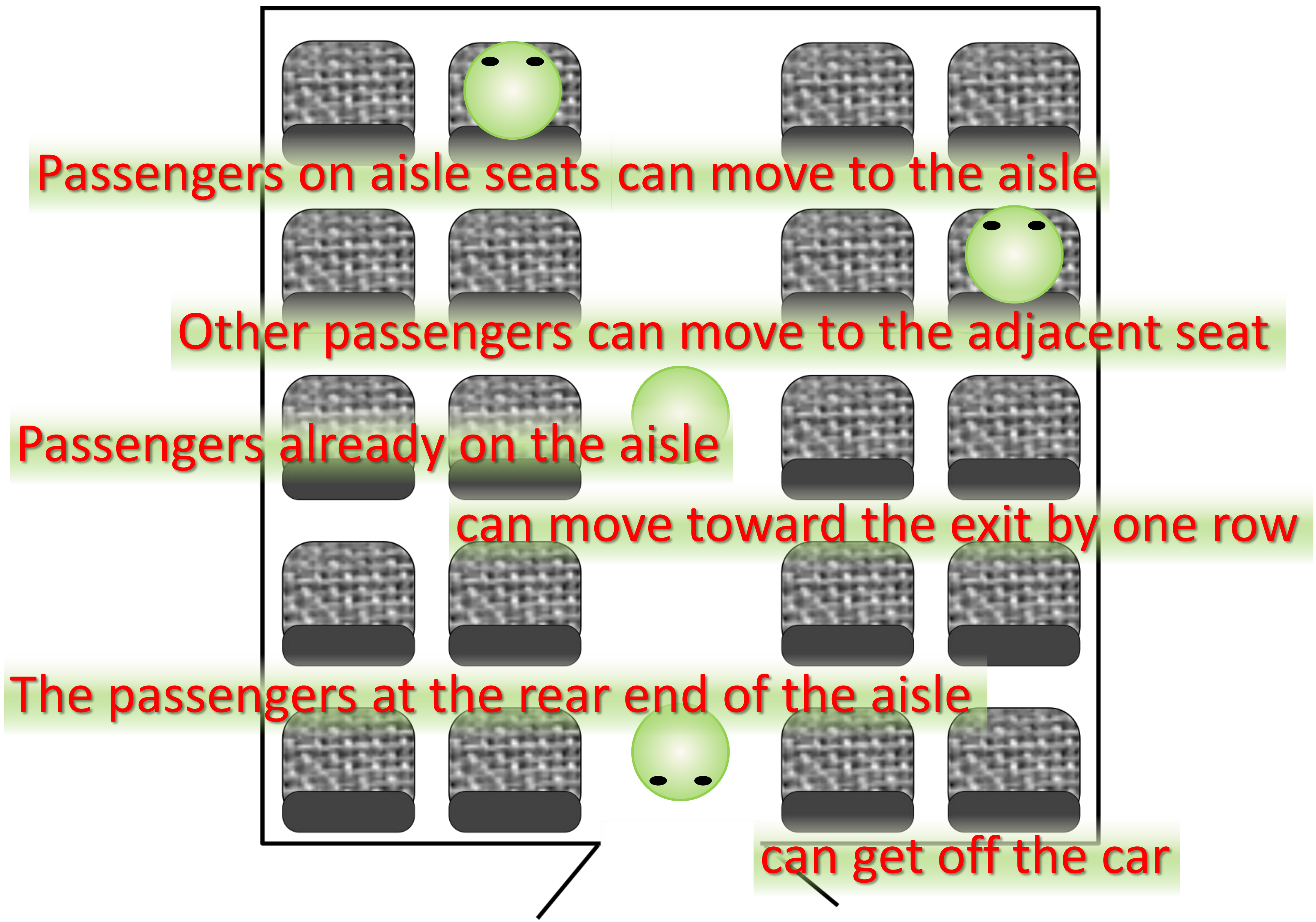
C: Emergency Evacuation

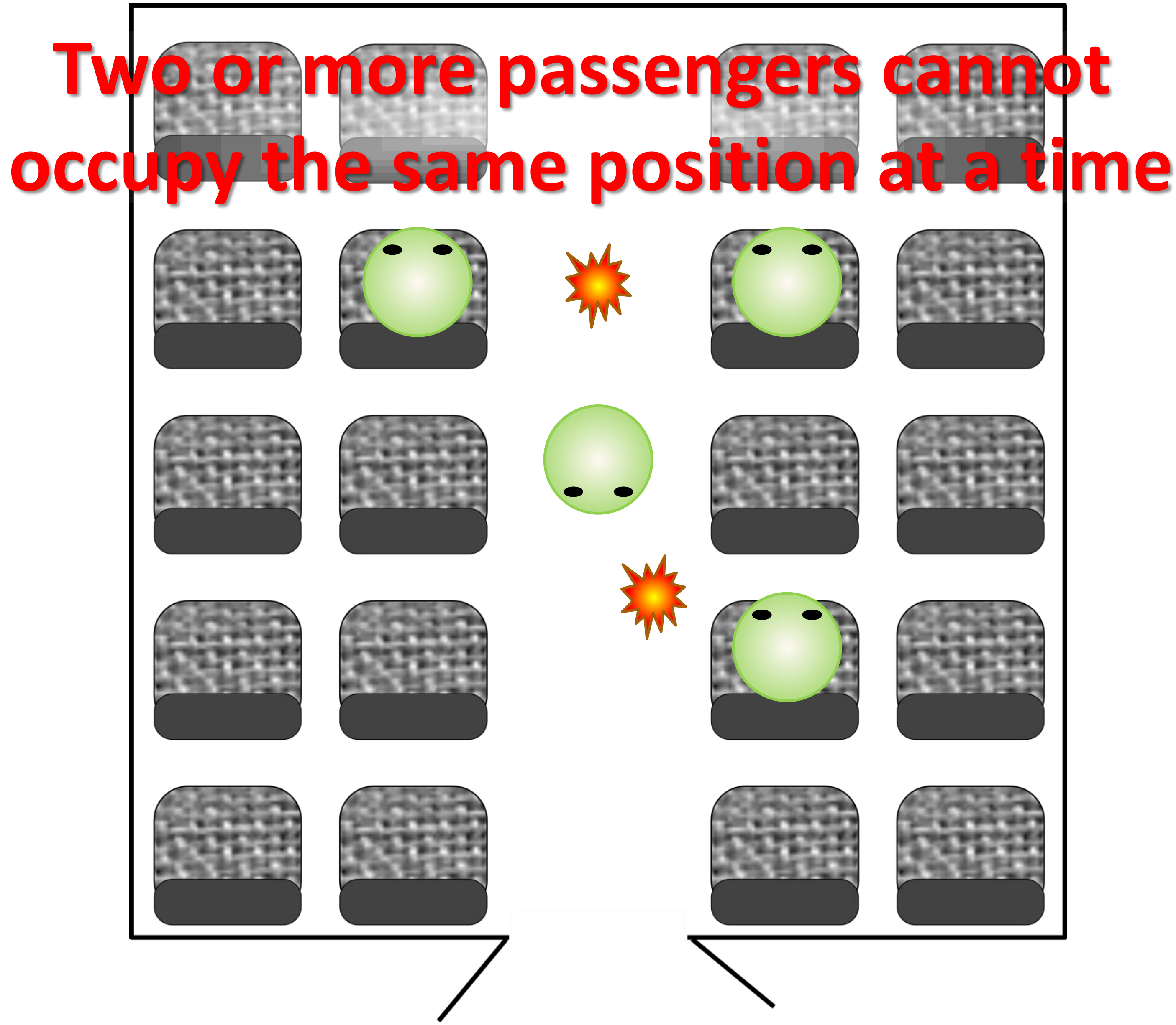
Problem:

Compute how many steps are required to evacuate all the passengers from a vehicle.

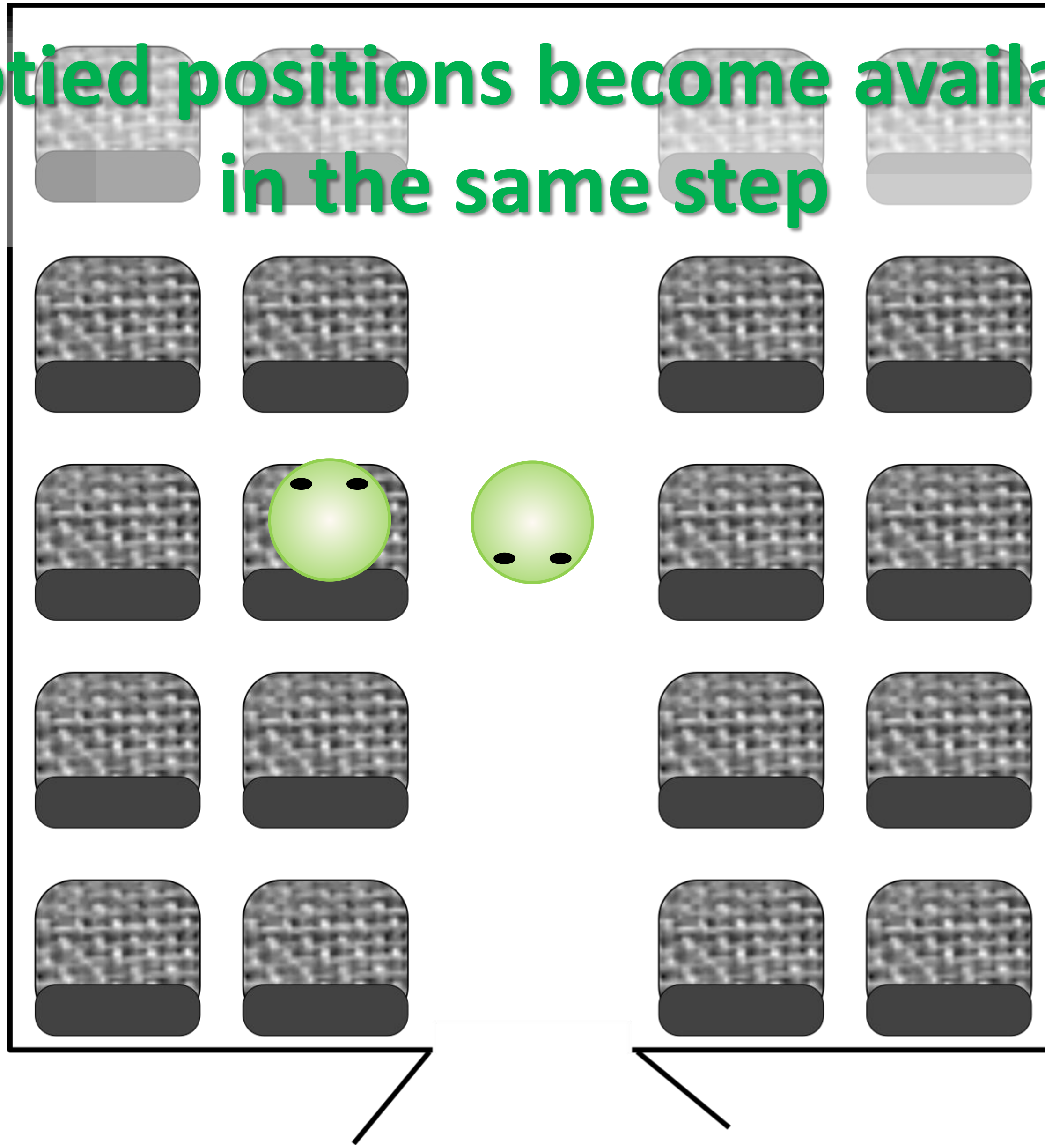
- The vehicle has an aisle in the center with rows of seats on its both side.
- The exit is at the rear end of the aisle.
- In one step, passengers on a seat can move sideways to an adjacent seat or to the aisle if on the aisle seat.
- Passengers on the aisle can move one row towards the exit, or get off from the vehicle if already at the rear end.


Rows of Seats
Aisle in Center
Exit at the Rear End
Passenger Seats on Both Sides


Passengers on aisle seats can move to the aisle
Other passengers can move to the adjacent seat
Passengers already on the aisle
can move toward the exit by one row
The passengers at the rear end of the aisle
can get off the car


Two or more passengers cannot
occupy the same position at a time

Emptied positions become available in the same step



A Reversed Problem

- Passengers will get on the car, one at a time, and walk up to their reserved seats
- Their possible moves are the reverses of the original problem

How many steps are required for all passengers to reach their reserved seats?

The answer should be the same as the original problem.

You don't have to take care of interferences!

Solving the Reversed Problem

- Let d_p be the distance of the seat from the door for passenger p
- Let s_p be the step number in which passenger p gets on the car
- The step in which passenger p reaches the seat is $d_p + s_p$

To minimize the total time $\max_p(d_p+s_p)$, passengers with the larger d_p should be given the smaller s_p

1. Compute d_p: $O(n)$
2. Sort them to decide desired s_p: $O(n \log n)$ ← *dominant*
3. Find $\max_p(d_p+s_p)$: $O(n)$

Step-wise simulation is *not* required!

B:Arithmetic Progressions

Problem Description

Find the longest Arithmetic Progressions from given set.

Given Set

0 1 6 9 3 5

Arithmetic Progressions

0, 3, 6, 9

9, 6, 3, 0

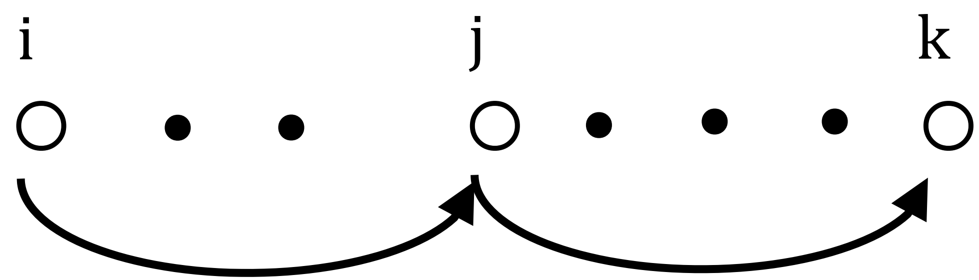
the longest ones

1, 5, 9

9, 5, 1

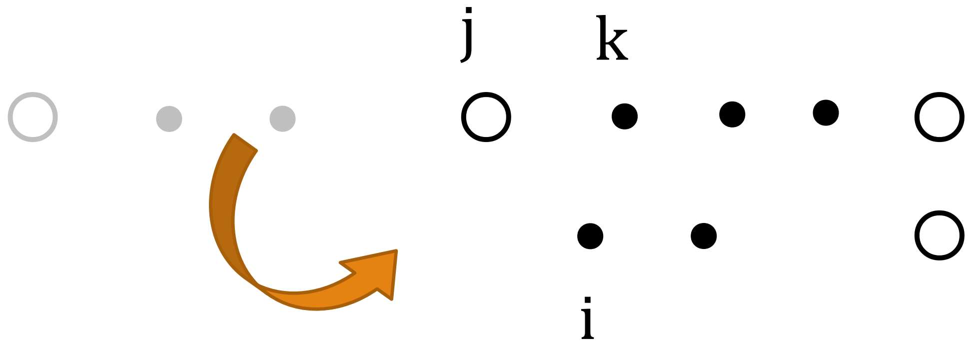
... and trivial ones

Solution



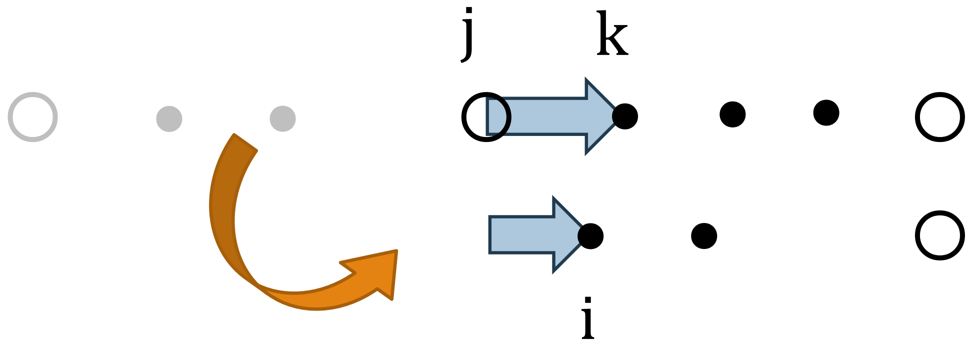
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



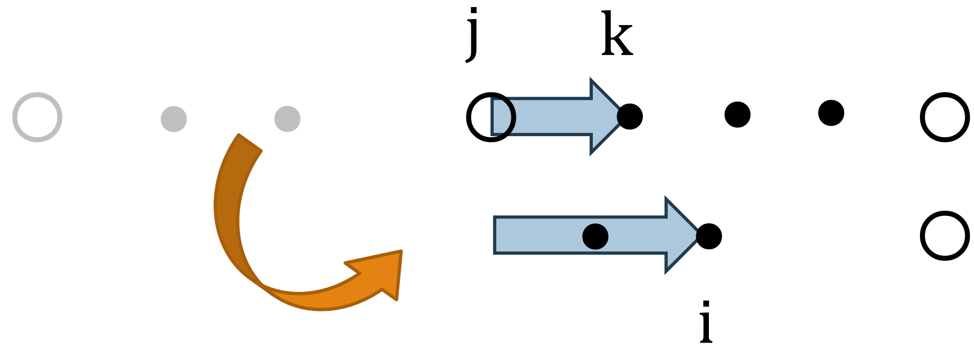
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



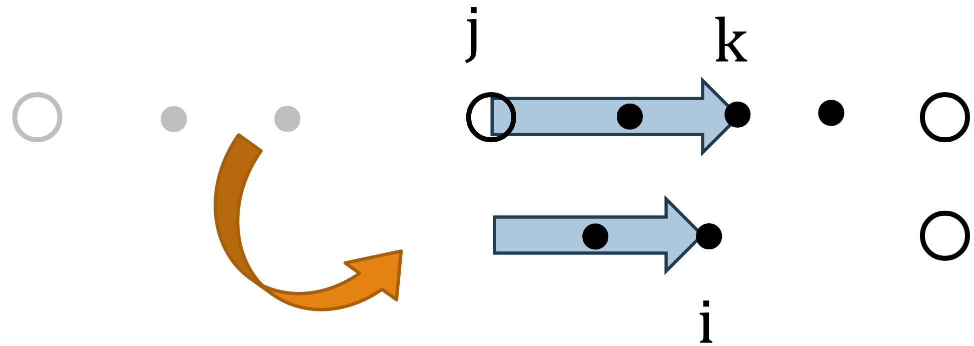
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



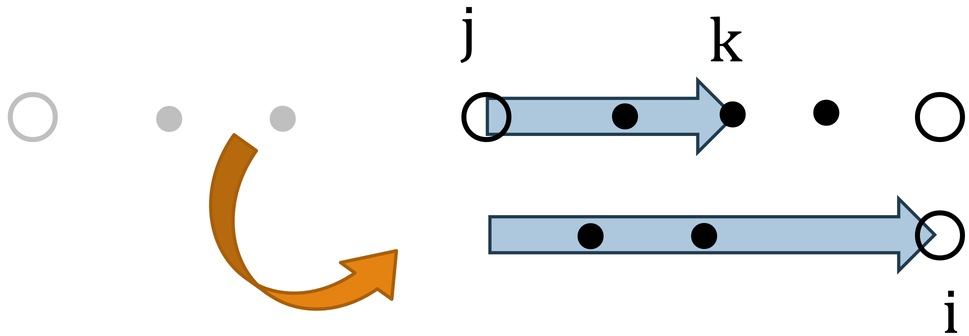
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in O (n^2).
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



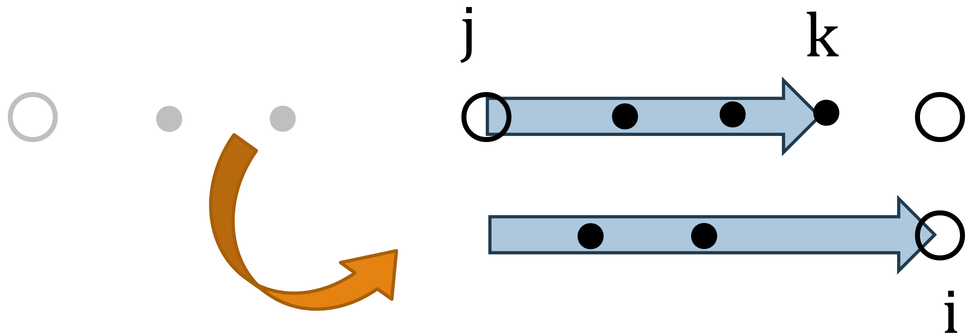
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



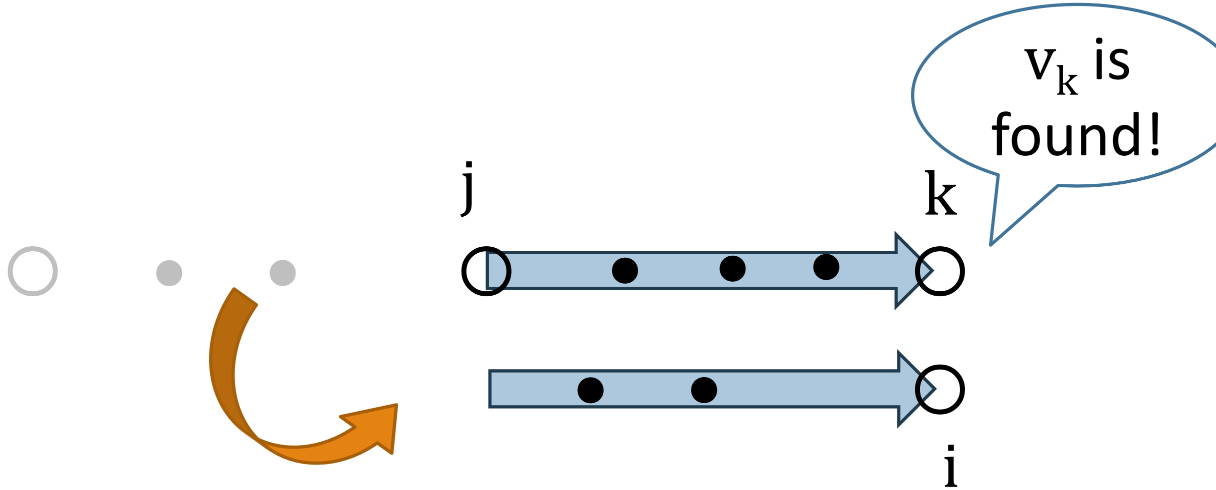
- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in $O(n^2)$.
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

Solution



- Sort the elements in the set
- For each i and j, remember k such that $v_k = v_j + (v_j - v_i)$
  - This table can be made in O (n^2).
  - Start i and k from neighbors of j.
- Check the length of the arithmetic progressions.

G: What Goes Up Must Come Down

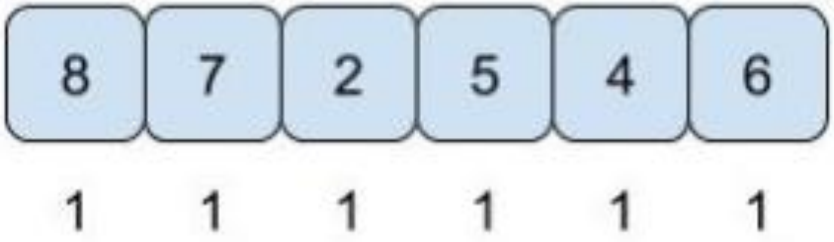
Problem

- Given an integer sequence
- Swap adjacent elements some times
- Rearrange that first some elements are increasing order, latter are decreasing order.

  $1 \leqq 4 \leqq 5 \geqq 3 \geqq 2$
- Find minimum number of swaps

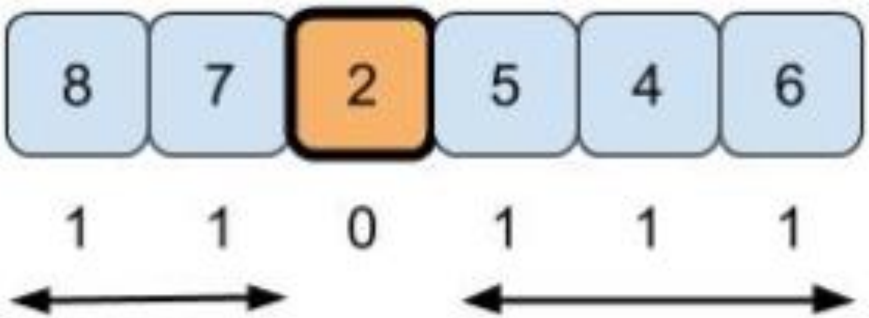
Solution

- Binary indexed tree
  O(log n) add : data[i] += v
  O(log n) sum : data[i..j]
- Initially every index has 1



Solution

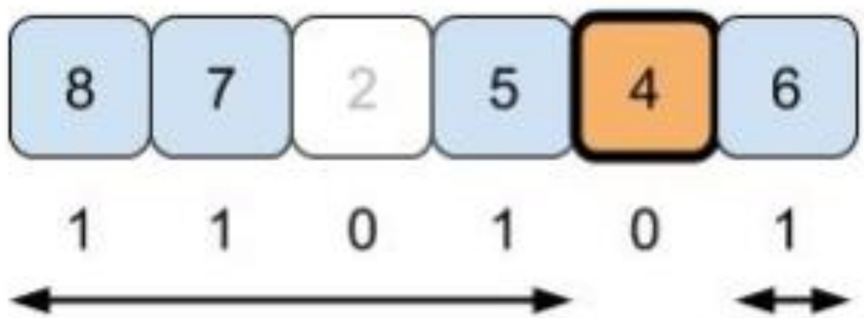
- Look at the smallest element
- Move it to leftmost or rightmost side
  answer += min(2, 3)
- Deactivate add(index, -1)



Solution

- Look at the second smallest element
- Move it to leftmost or rightmost side

  answer += min(3, 1)



Conclusion

- Count left large values and right large values each indexes
- Add small one to the answer
- Use binary indexed tree (or segment tree)

- ※ Be careful to update when the sequence contains duplicate entries.

D: Shortest Common Non-Subsequence

Background:
Longest Common Subsequence

S is a subsequence of P:

P	0	1	1	0	1	1	0	1
S			0	0	0	1		

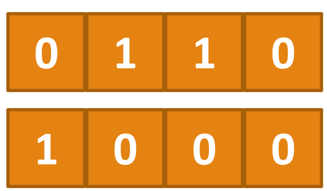
<u>Longest Common Subsequence Problem:</u>

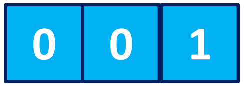
- Input: two sequences A and B
- Output: longest common subsequence

Problem

<u>Shortest Common Non-Subsequence Problem:</u>

- Input: two sequences (consisting of 0 and 1)
- Output: shortest sequence (consisting of 0 and 1) that is a subsequence of neither of two sequences



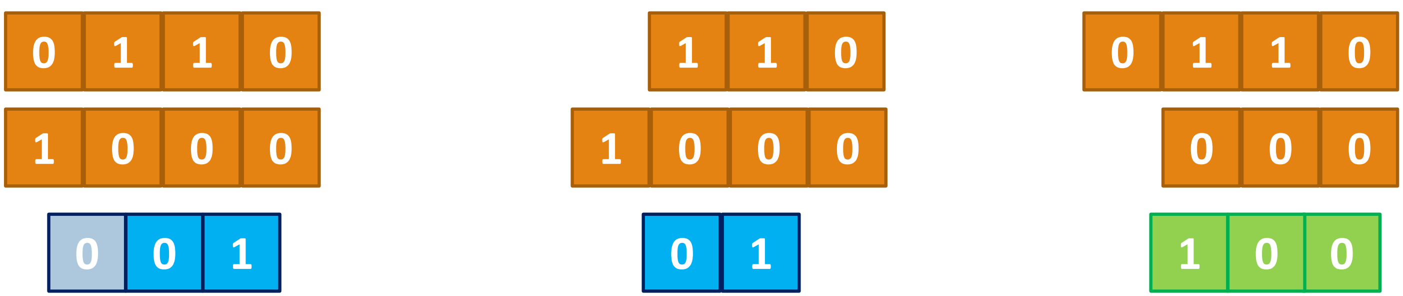


Solution --- Dynamic Programming

Observation

A SNCS of A[i,n] and B[j,m] is obtained by
- adding 0 to a SNCS of A[i0,n] and B[j0,m] or
- adding 1 to a SNCS of A[i1,n] and B[j1,m]

where i0, j0 are the indices of the next appeared 0, and i1, j1 are the indices of the next appeared 1.



Solution Recovery

We compute SNCS(i,j) for all i and j by DP

To obtain the lexicographically smallest solution, we use the following standard technique:

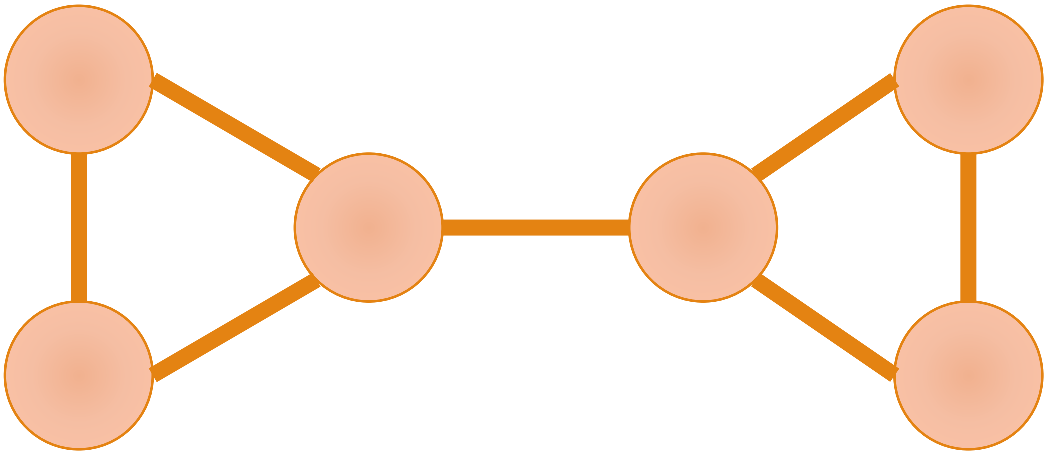
If SNCS(i,j) = SNCS(i0, j0) + 1,
there is a solution that starts from 0

Otherwise, the solution must start from 1

E: Eulerian Flight Tour

[Problem]

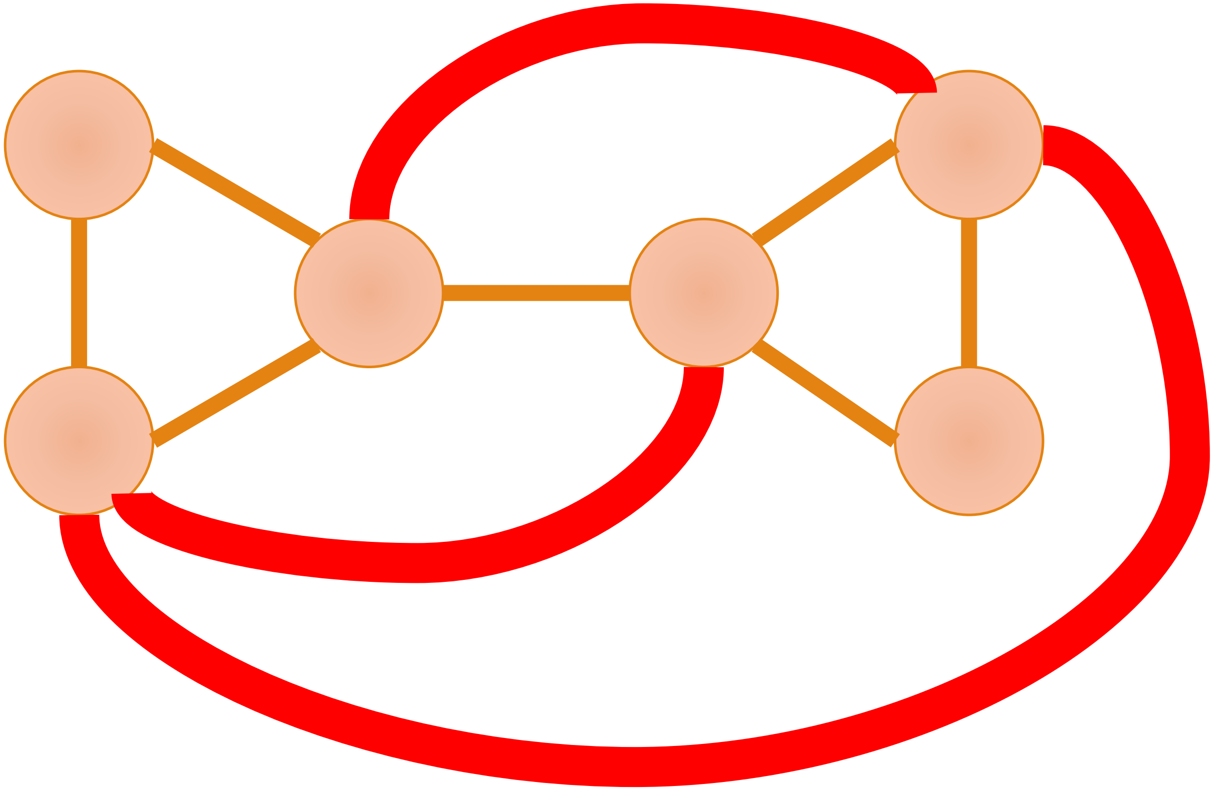
Given an undirected simple graph,



make it **Eulerian** by adding edges!

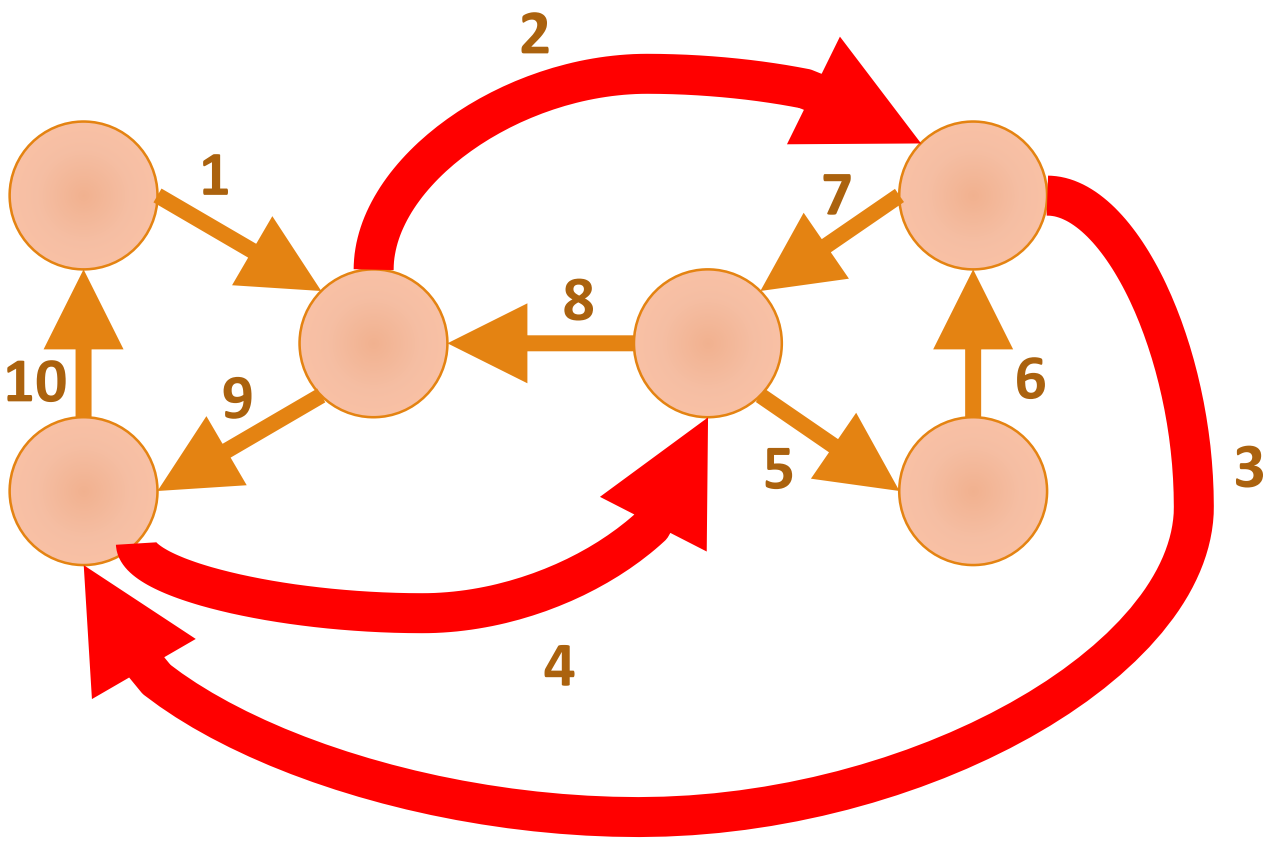
Eulerian =

Have a cycle visiting all nodes,
using all edges exactly once each



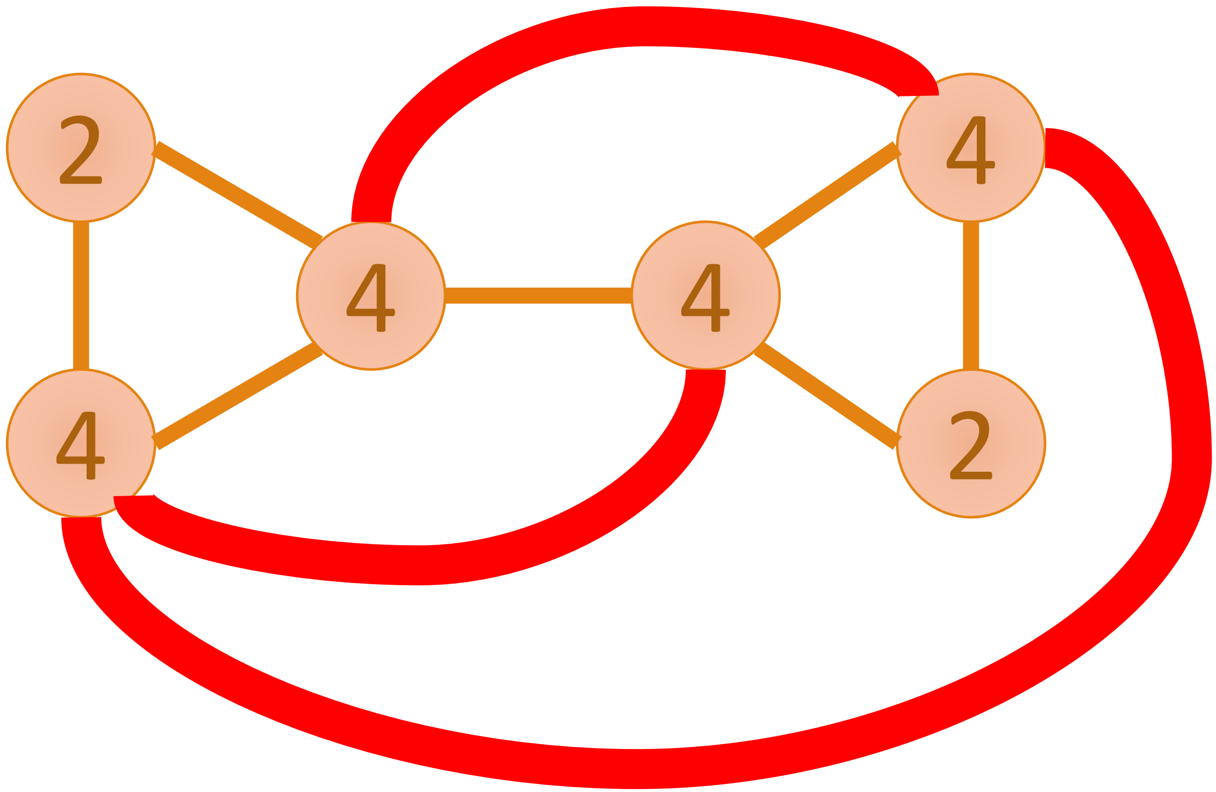
Eulerian =

Have a cycle visiting all nodes, using all edges exactly once each



[Famous Fact]

Eulerian ⇔ **Connected** and
all nodes have **even degree**



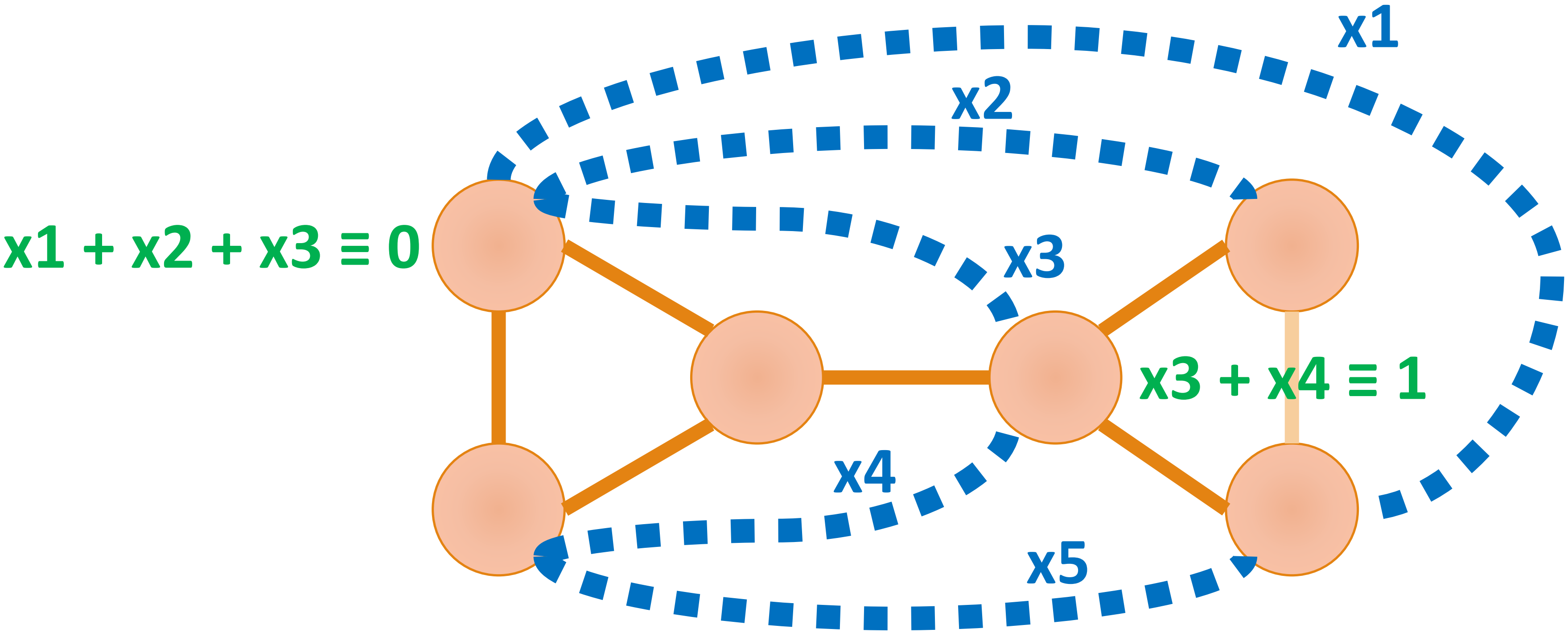
[Solution]

1. Add edges and make all nodes **even-degree**
2. Then, add more edges to make it **connected**

[Step 1: Even-Degree]

Approach 1 : Linear Equations in mod 2

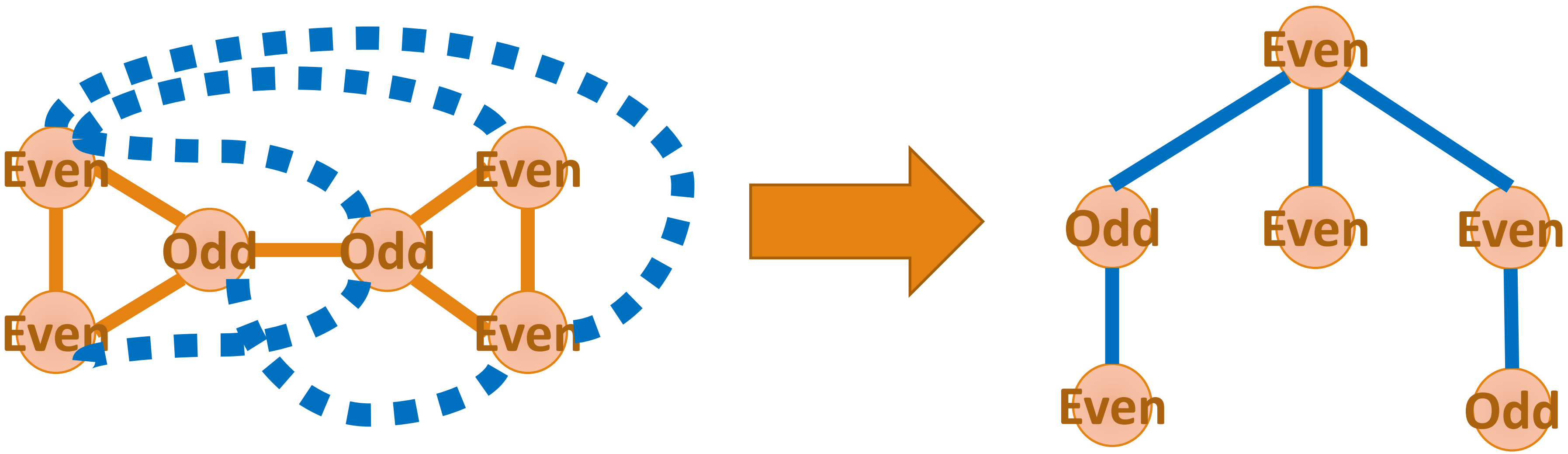
* Edge Candidate = Variable (1: use, 0: not)
* Node = Equation



[Step 1: Even-Degree]

Approach 2 : Graph Theoretic

Think about a **spanning forest** of the completement graph



for v in **bottom-up order in the spanning tree**:

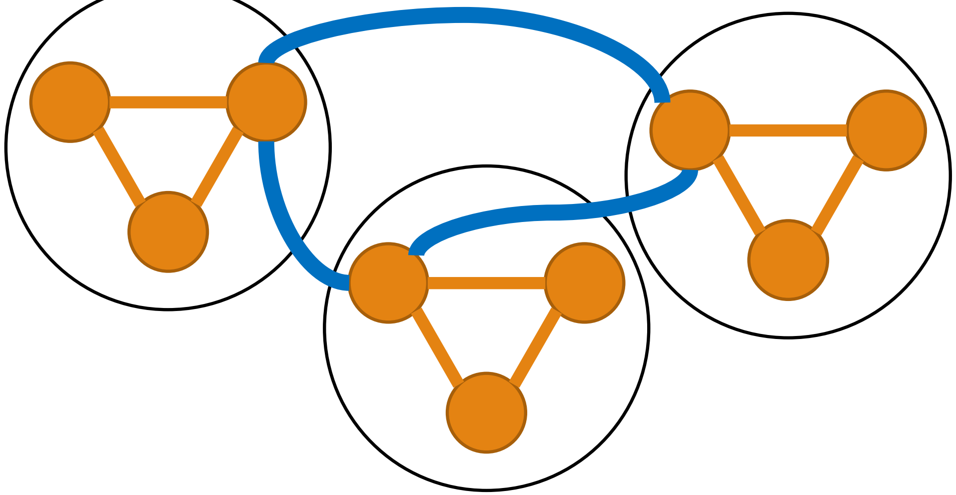
if "**deg(**v**)** in original graph" is odd:

Use the edge (v, **parent(**v**)**) and update **deg(**v**)**

[Step 2: Connected]

If the previous step generated…

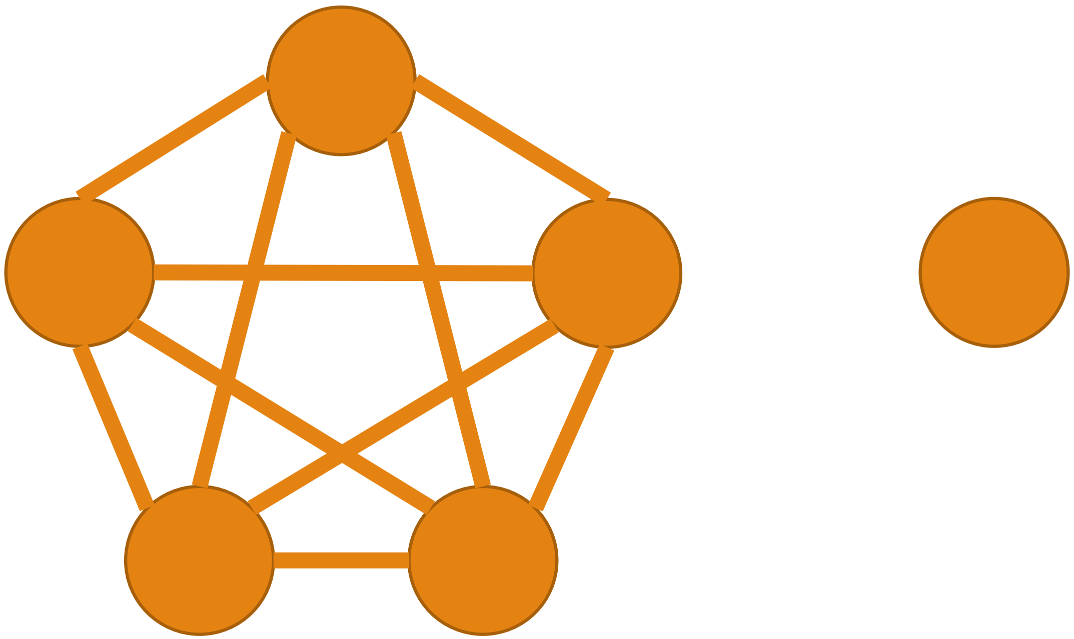
- 1 connected component ➔ Eulerian! Solved!
- 3 or more components ➔ **Connect by a cycle**



- 2 components, each have 2+ nodes ➔ cycle
- 2 components, …

[Step 2: Connected]

The only one exceptional case:



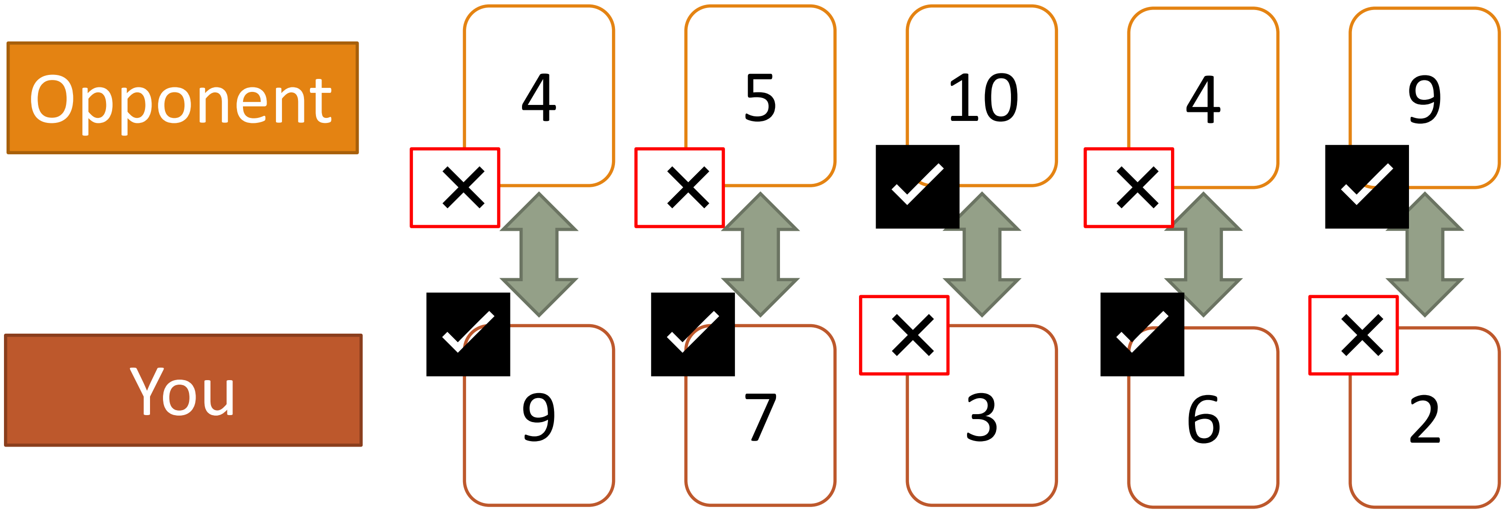
Input was already (K_{odd} + point)

For all other cases, multiple connected components can be made into one big one in some way.

K: Sixth Sense

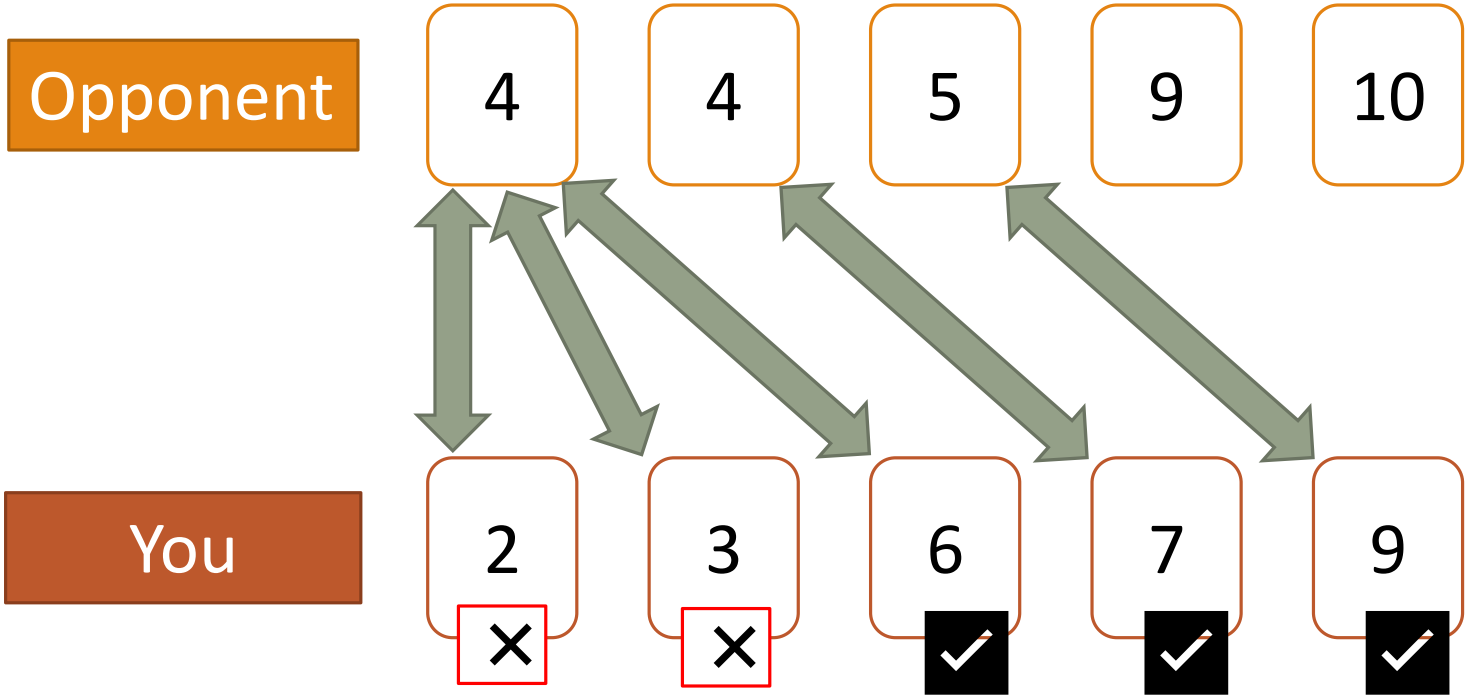
Problem

- Play a two-player trick-taking card game
  - Each player pulls out a card in every trick
  - The player pulling out a card with the larger number takes the trick
- Find the best way to win the game, assuming that you know the opponent's actions



Just to maximize the number of tricks you take is very easy

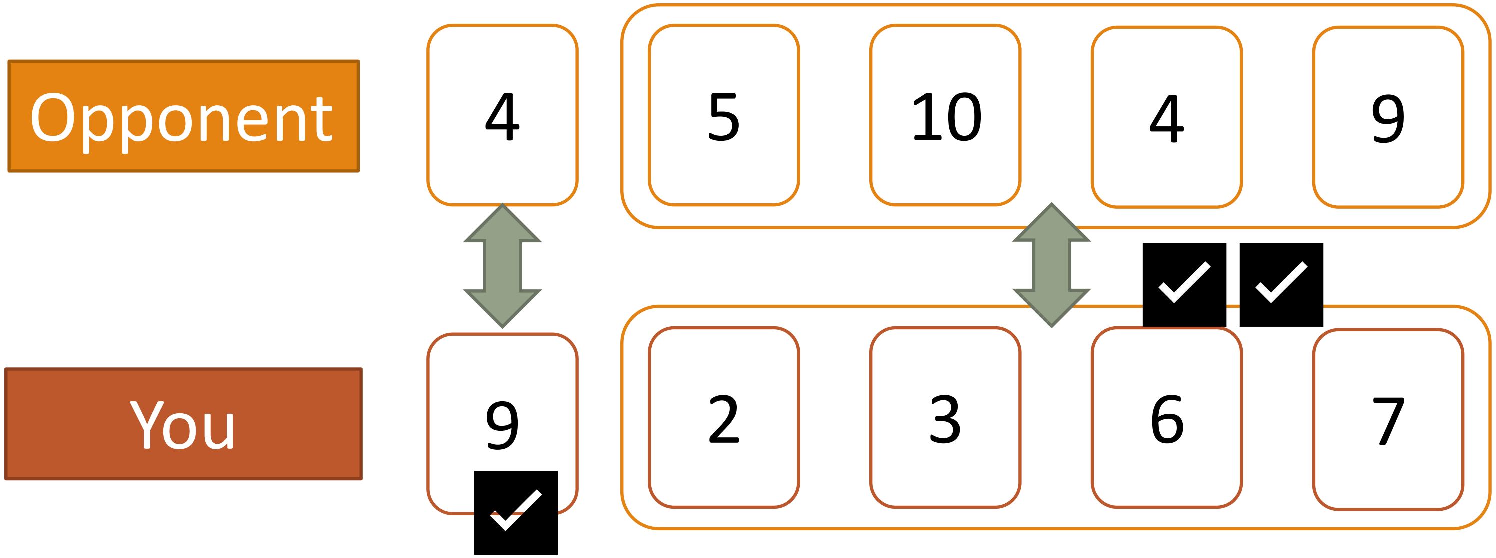
- Sort the cards and compare one by one



It takes $O(n \log n)$ time, where n is #cards

To have the lexicographically largest ordering Is harder

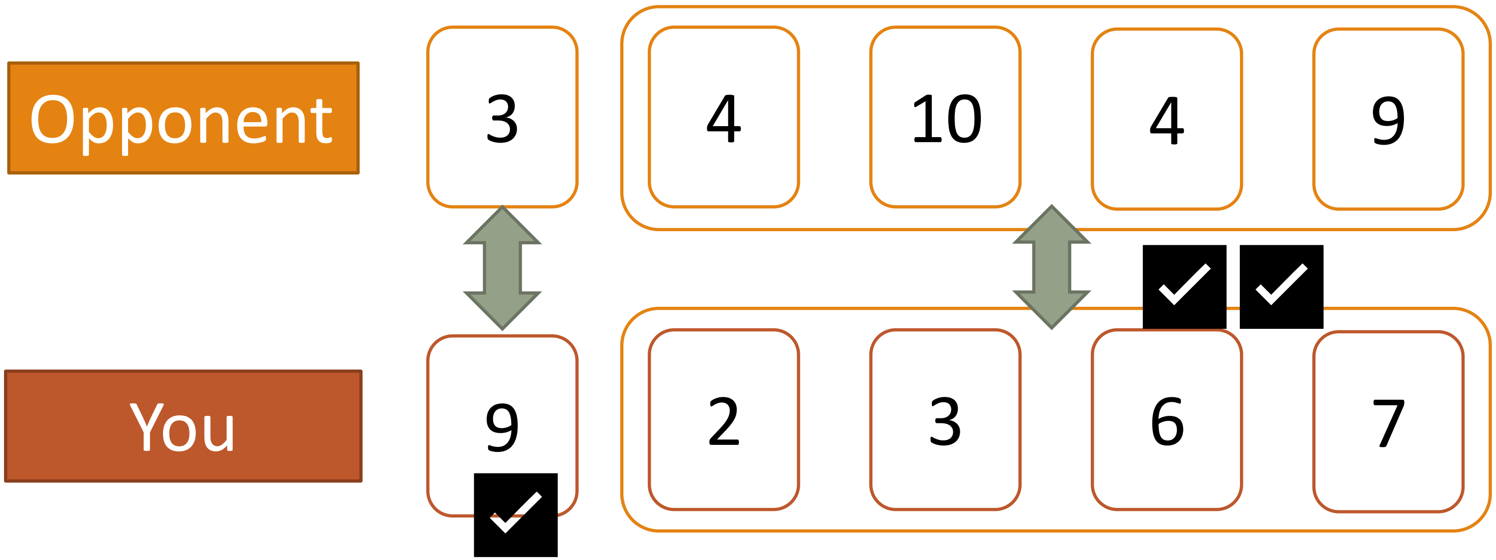
- For each trick, choose the best card
  - The largest among those with which you can take the maximum number of tricks



- For each candidate card, the best achievable number is computed in $O(n)$ time

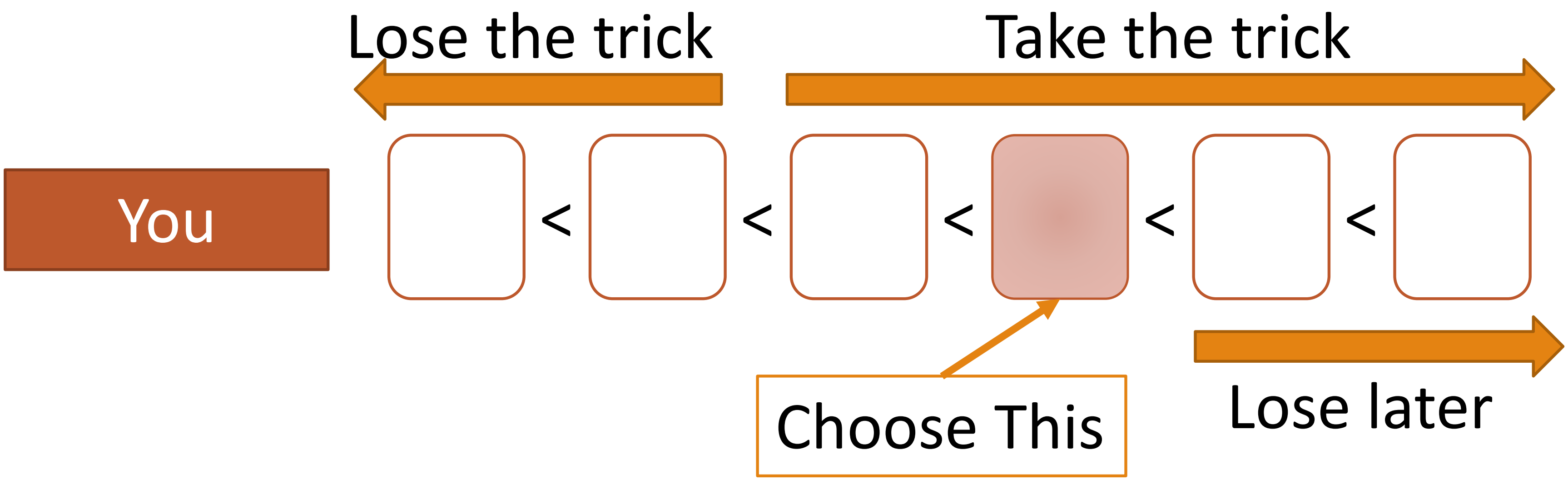
To have the lexicographically largest ordering Is harder

- If you can take this trick, you should
  - By losing this trick, you cannot increase the total number of tricks you take
- If you cannot, you will lose the trick anyway



To have the lexicographically largest ordering Is harder

- You can determine whether or not the candidate number is too large



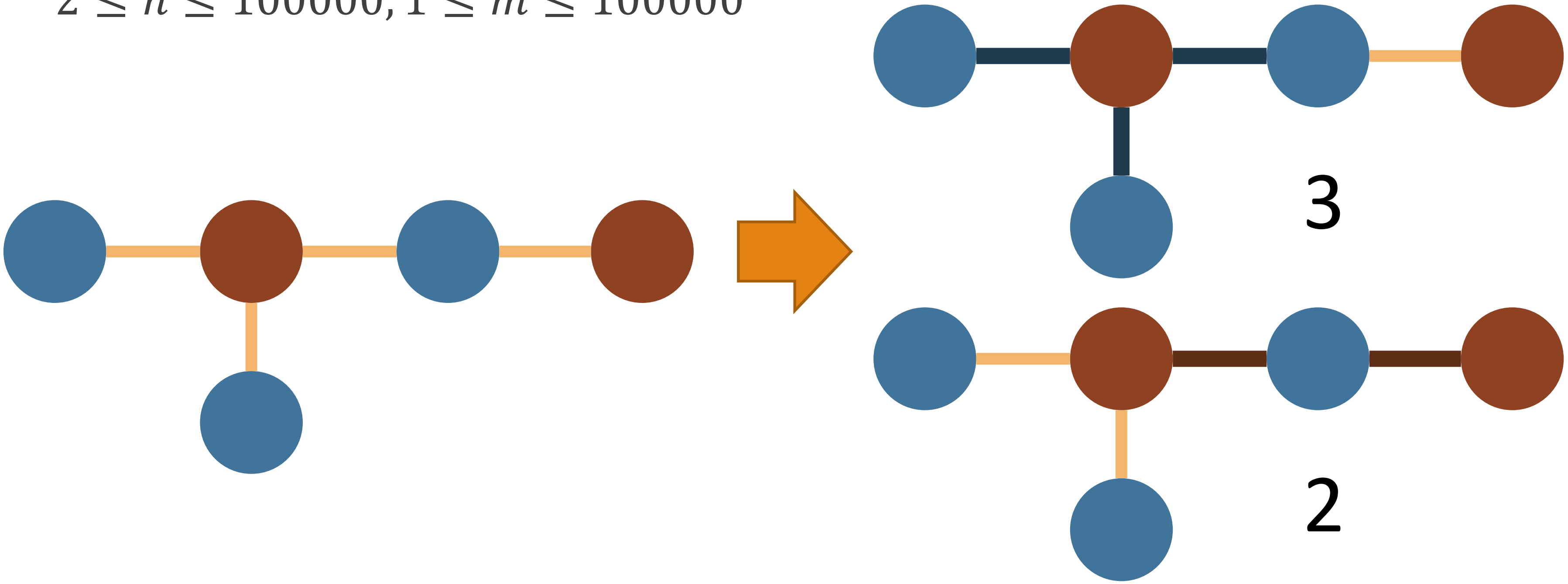
Complexity: With binary search, #candidates is $O(\log n)$
$O(n)$ time for each candidate
#tricks is $O(n)$ and so in total $O(n^2 \log n)$

J: Colorful Tree

Problem Summary

Given a tree with colored vertices and a sequence of commands
Command:
1. Change the color of a specified vertex
2. Ask the number of edges in the minimum connected subgraph of the tree containing all vertices of the specified color

$2 \le n \le 100000, 1 \le m \le 100000$



Efficiently keep/update subgraphs

It's difficult to compute the subgraph efficiently in each query command

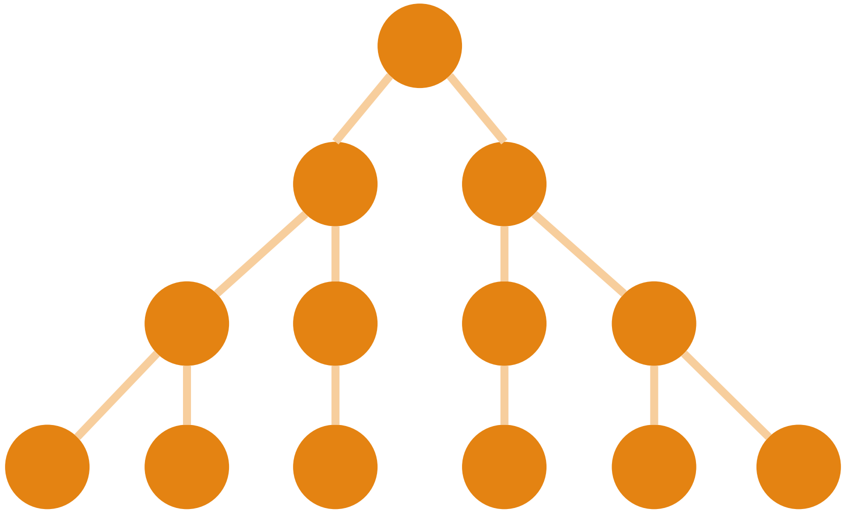
An update command just changes the color of one vertex
→ Remove one vertex from a subgraph and add one vertex to a subgraph

We can't keep all subgraphs as they are
(The total number of vertices/edges can be huge)

Which information is needed for each subgraph and how can we update it?

The change in an update command

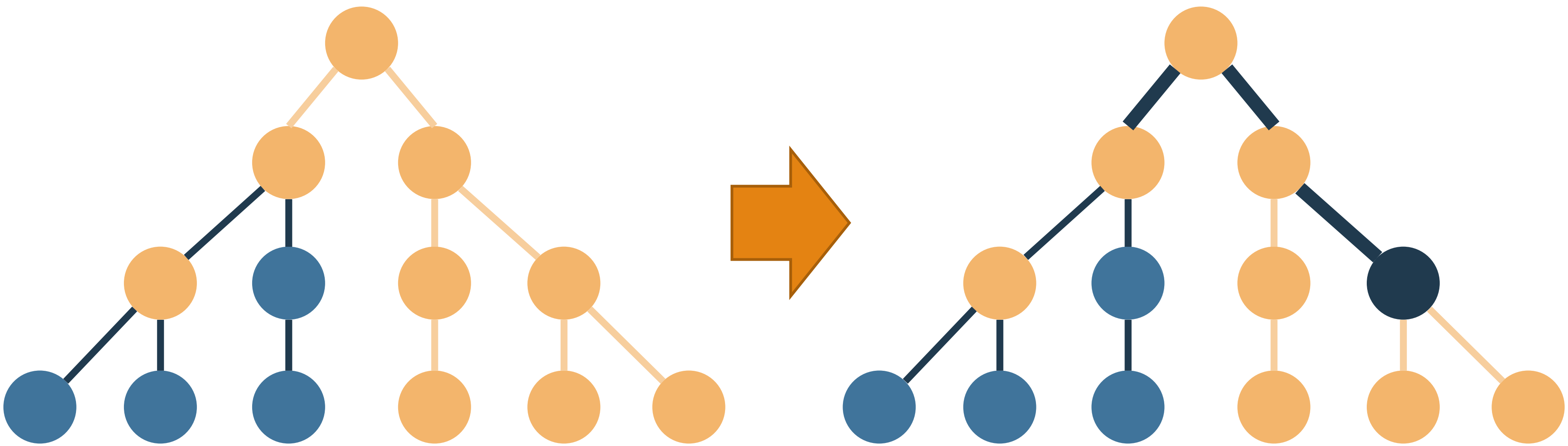
Consider as a rooted tree



What happens when adding/removing a vertex?

The change in an update command

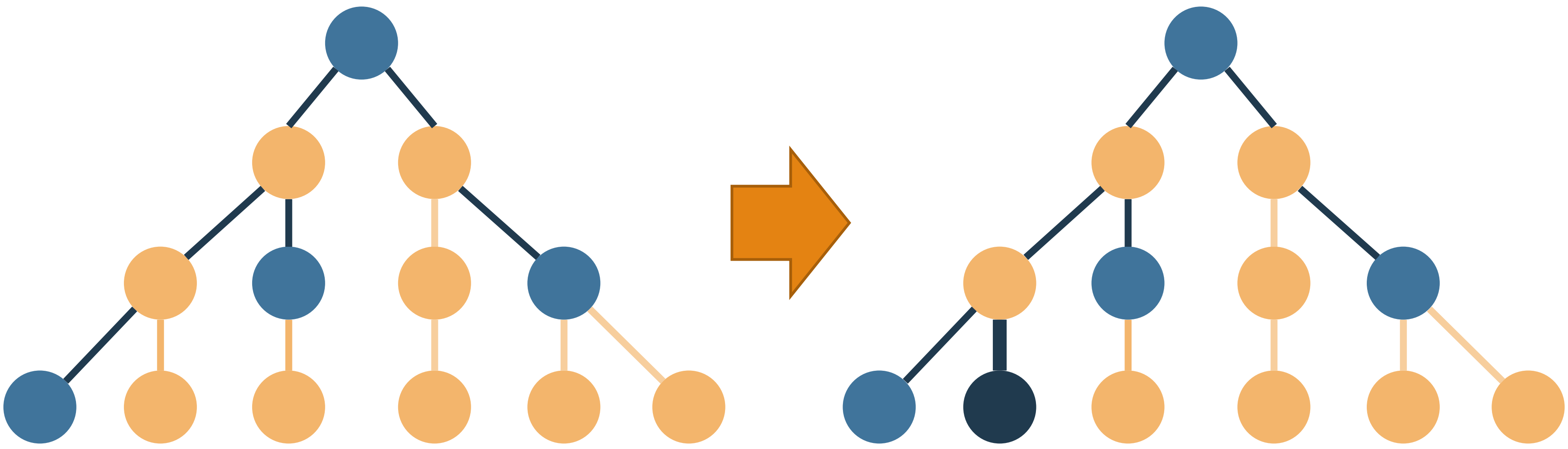
1. The LCA (lowest common ancestor) of the subgraph changes



The distance from the changed vertex to **the original LCA** is added (adding)
the new LCA is subtracted (removing)

The change in an update command

2. The LCA of the subgraph doesn't change



The distance from the changed vertex to **a vertex** is added/subtracted

Where?

The closest LCA of the changed vertex and some vertex

Efficiently keep/update subgraphs

What we should keep for each color

1. The current number of edges
2. The set of vertices having the color
3. The LCA of the subgraph

$\Downarrow$

1. Calculate the LCA → $O(\log n)$ when adding a vertex

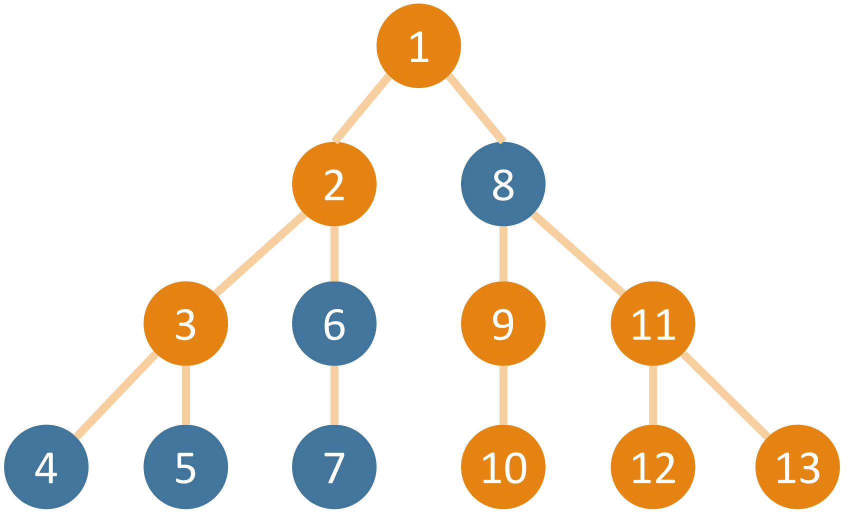
   $\boldsymbol{O(n \log n)}$ when removing a vertex

2. Find the closest LCA of the changed vertex and some vertex → $\boldsymbol{O(n \log n)}$

The total time complexity is $\boldsymbol{O((m+n)n \log n)}$

Pre-order numbering

Assign a number to each vertex by pre-order (or in/post-order)

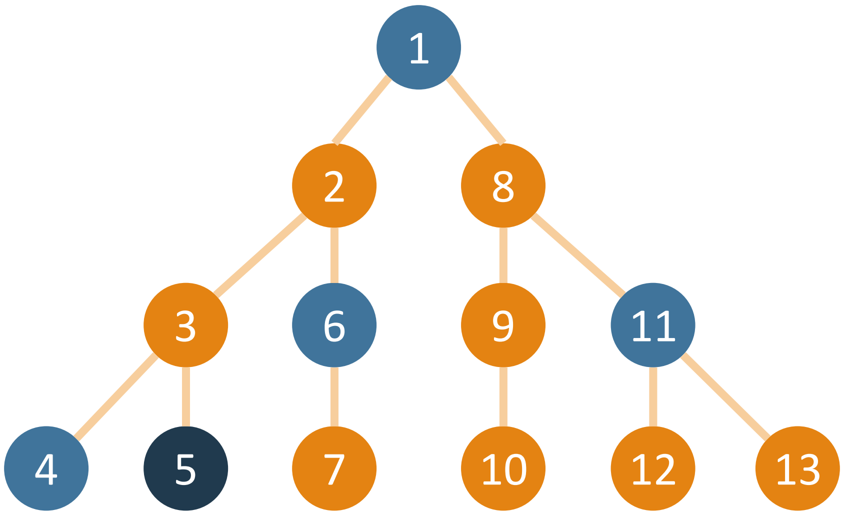


Calculate the LCA of the subgraph

→ The LCA of **the minimum number vertex** and **maximum number vertex**

Pre-order numbering

Assign a number to each vertex by pre-order (or in/post-order)



Find the closest LCA of the changed vertex and some vertex

→ The LCA of the changed vertex and
the maximum number vertex within smaller number vertices

or the LCA of the changed vertex and
the minimum number vertex within larger number vertices

Summary

What we should keep for each color

1. The current number of edges
2. The set of vertices having the color **with pre-order (or in/post-order) numbers**
3. The LCA of the subgraph

1. Calculate the LCA → $O(\log n)$
2. Find the closest LCA of the changed vertex and some vertex → $O(\log n)$

The total time complexity is $\boldsymbol{O((m+n)\log n)}$

I: Ranks

Problem

You're given an $n \times m$ matrix A over $\mathbf{F}_2$.
For all indices (i, j), determine if flipping (i, j) entry of A increases/decreases/keeps the rank of A.

Constraints: Dimensions n,m <= 1000

Rank of Matrix?

Gaussian elimination (GE) with bit-parallel:

- $O(n^2 m / b)$ (b: bit length)

Naively applying GE $n*m$ times:

- $O(n^3 m^2 / b)$: obviously TLE

Some efficient algorithm is required here.

For Last Column (1/3)

Consider from simple case: $j=m$.

- Assume that we computed r (the rank of A).
- We can compute $r^{i,m}$ by GE for $A+E^{i,m}$, where $E^{i,j}$ is matrix with (i,j) entry 1 and other ones 0.
  - $O(n^3 m / b)$ ☹
  - But we can speed up this computation.

For Last Column (2/3)

- Let A_m be the m-th column vector.
- Let δ_i be a vector with only the i-th element 1.

Now, let X:=[(A_m+δ_1) … (A_m+δ_n)]. To compute $r^{i,m}$ for all i,

- Suppose now we have [A | X] (concatenated matrix).
- Perform GE from first to the (m-1)-th column.
- Then, for each i, check if the additional GE to (m+i)-th column increases the rank. → This works in $O(n^2m / b)$. ☺

For Last Column (3/3)

Instead of $[A \mid X]$, we may do the same thing for $[A \mid I]$.

- Any row operations can be expressed as an $n \times n$ matrix S.
- Applying S to some matrix B changes it to $S \cdot B$.
- For any S, ((m+i)-th column of $S \cdot [A \mid X]$) $= S \cdot (A_m + \delta_i)$
  = (m-th + (m+i)-th column) of $S \cdot [A \mid I]$).

So, perform GE by the end of m-th column, then check (m+i)-th column.

General Case (1/3)

Now we want to compute $r^{i,j}$ for any (i, j).

We consider GE for $[A \mid I]$. Now let us assume that

- We perform GE by the end of m-th column.
- We also perform back substitution: We deleted as many 1's in A as possible.

Now the A's part will look like this (echelon form).

```
0 1 0 1 0 0 0 1 0 0 0
0 0 1 1 1 0 1 0 0 0 0
0 0 0 0 0 1 0 0 1 1 0
0 0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

General Case (2/3)

When **two or more 1's** are lined up in the j-th column in echelon form:

- Suppose that we move the j-th column to the end of columns.
- The matrix is still echelon form.
- So, it suffices to look the *I*'s part to compute $r^{i,j}$.

$$
\begin{array}{ccccccccccc}
0&1&0&1&0&0&0&1&0&0&0\\
0&0&1&1&1&0&1&0&0&0&0\\
0&0&0&0&0&1&0&0&1&1&0\\
0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0\\
&&&j&&&&&&&
\end{array}
\qquad
\begin{array}{ccccccccccc}
0&1&0&0&0&0&1&0&0&0&1\\
0&0&1&1&0&1&0&0&0&0&1\\
0&0&0&0&1&0&0&1&1&0&0\\
0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0\\
&&&&&&&&&&m
\end{array}
$$

General Case (3/3)

When **only one 1** is lined up in the j-th column:

- Again, suppose we move the j-th to the end.
- Now the matrix is not echelon form.
- Suppose that we swap rows and then move columns.
- This results in echelon form, since we performed back substitution.
- Again, it suffices to look the *I*’s part to compute $r^{i,j}$.

```
01010001000
00111010000
00000100110
00000011111
00000000000
00000000000
```

```
00100010001
01110100000
00001001100
00000111110
00000000000
00000000000
```

```
01110100000
00001001100
00000111110
00100010001
00000000000
00000000000
```

```
01101000100
00010110000
00001111010
00000000111
00000000000
00000000000
```

Conclusion

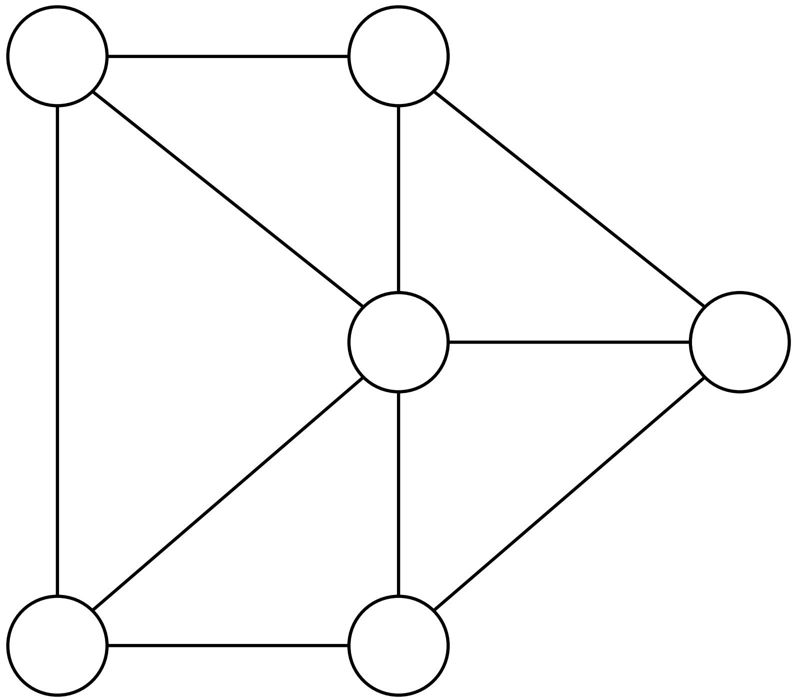
Time complexity: $O(n^2(n+m) / b)$

There're other solutions.

- Do similar thing through LR decomposition
- $O(n^3 \log n / b)$ solution (simpler?)

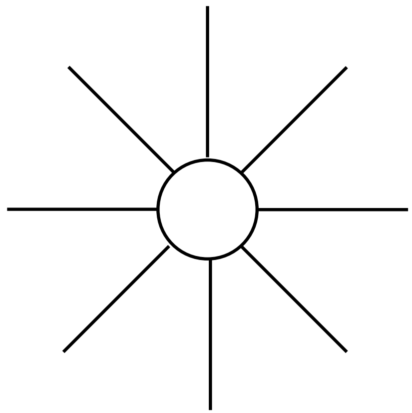
H: Four-Coloring

Find a 4-coloring of a planar graph.



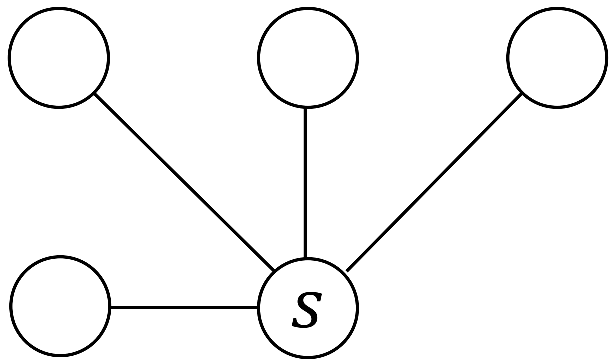
Special Constraints

- Edge ⇒ Straight line segment
- Inclinations are multiples of 45˚



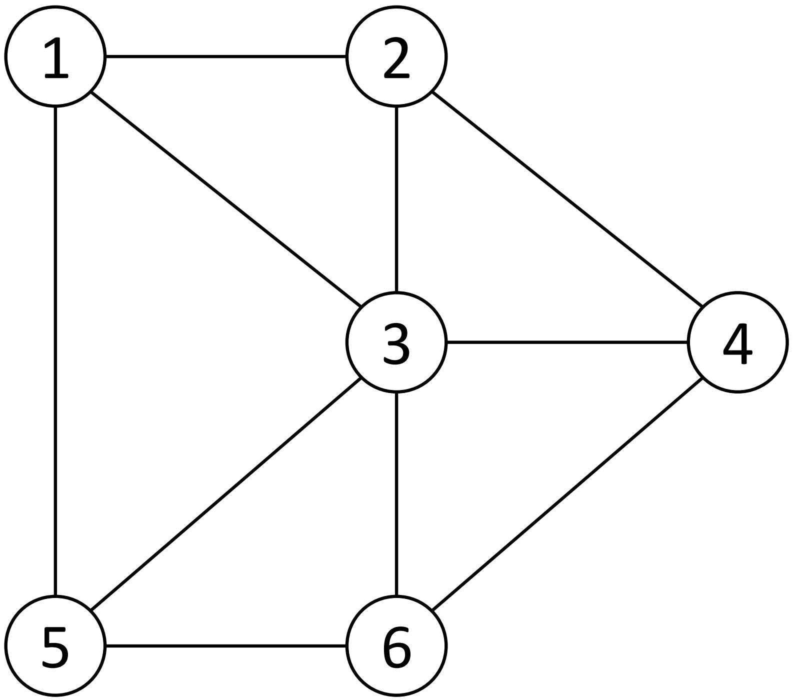
Key Observation

Bottom right vertex s has degree ≤ 4



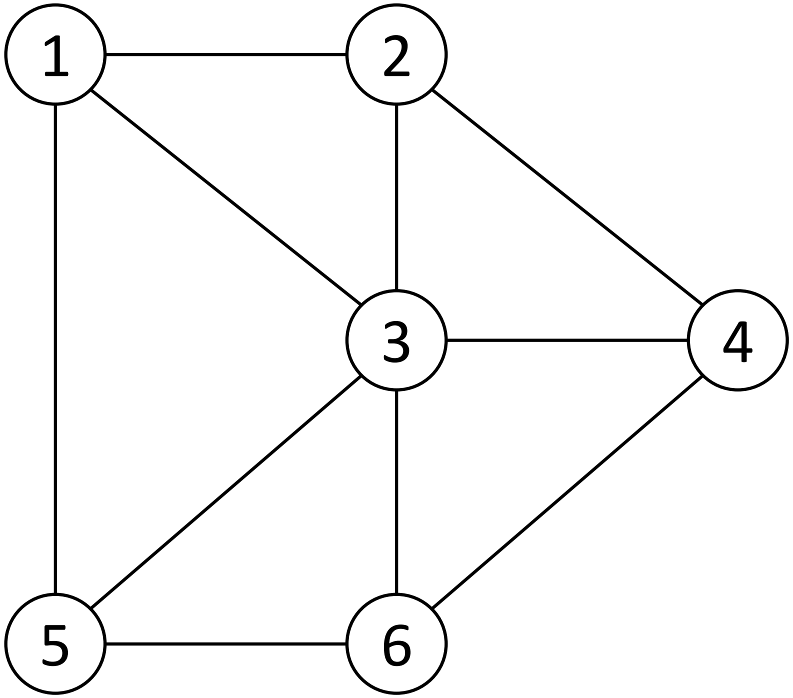
Algorithm

Lexicographically sort the vertices



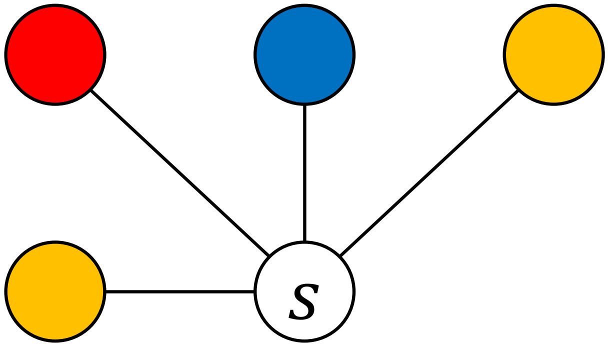
Algorithm

Extend the 4-coloring of $G[\{1, \ldots, i\}]$ to a 4-coloring of $G[\{1, \ldots, i+1\}]$.



Case 1

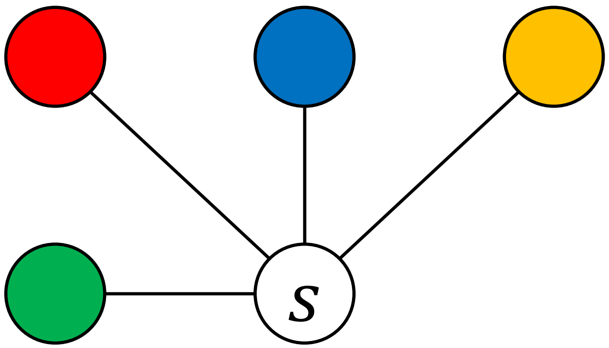
$N(s)$ uses at most 3 colors
$\Rightarrow$ use the remaining color



Case 2

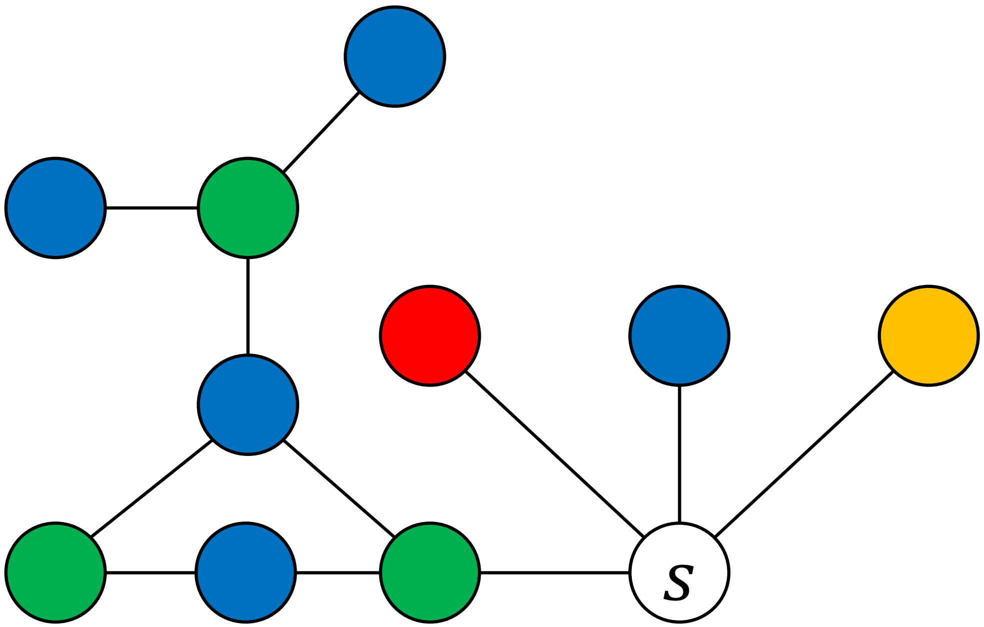
$N(s)$ uses 4 colors

$\Rightarrow$ try to change green to blue



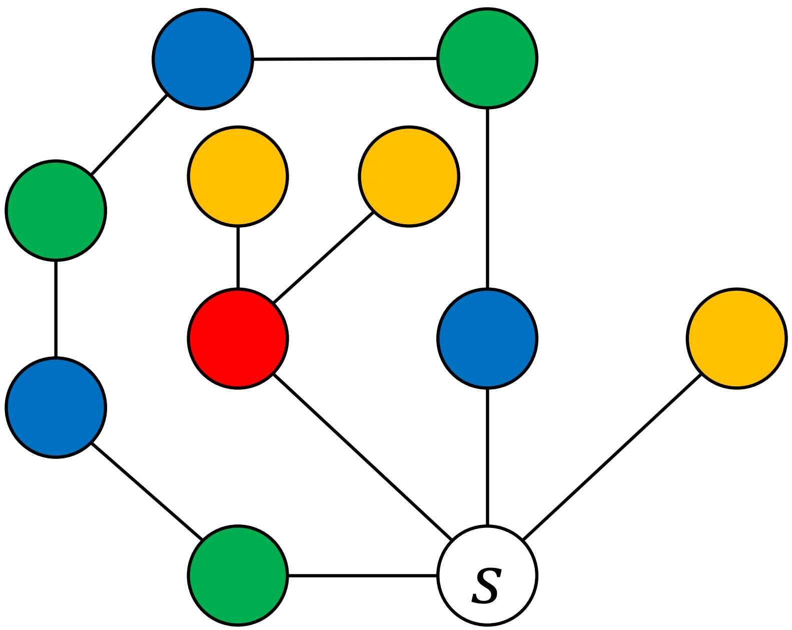
Case 2a

No green-blue alternating paths
⇒ swap green and blue



Case 2b

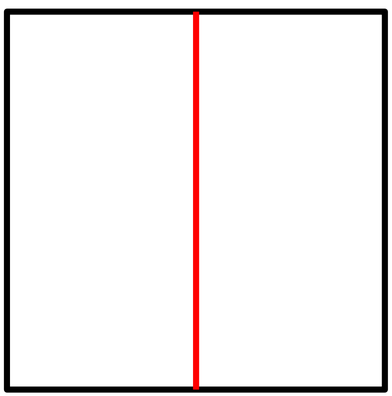
A green-blue alternating path exists
$\Rightarrow$ no red-orange alternating paths



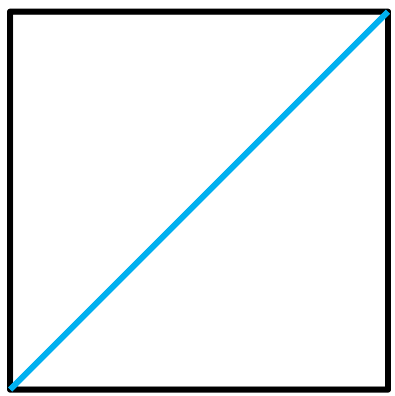
Problem F: Fair Chocolate-Cutting

Problem Summary

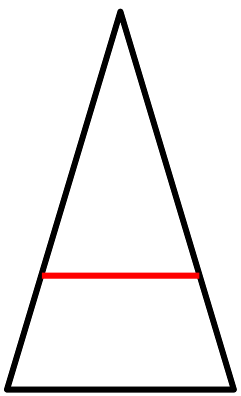
Given a convex polygon,
compute the minimum and maximum lengths of line segments that divide the polygon into two equal areas.



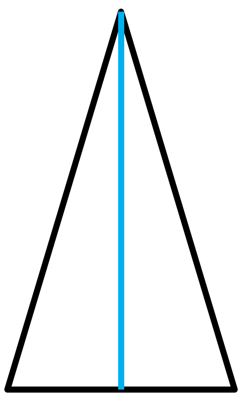
minimum
length



maximum
length



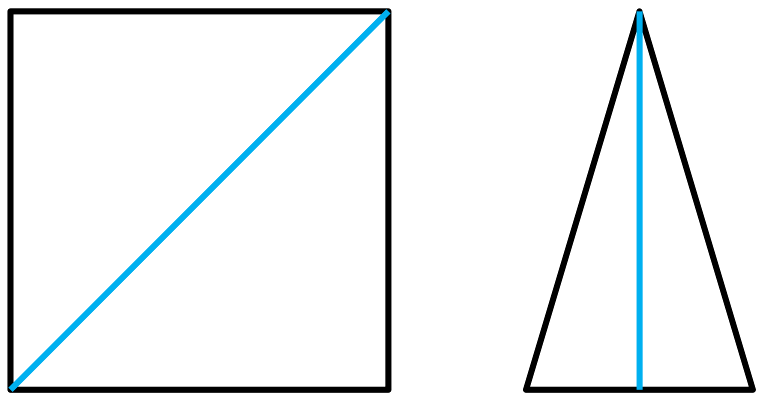
minimum
length



maximum
length

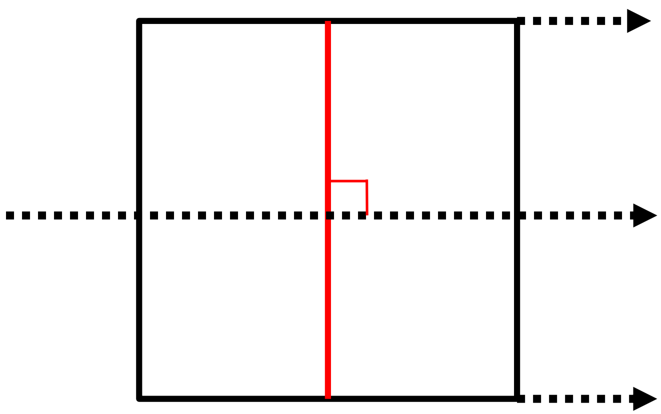
Observation to Solution (1)

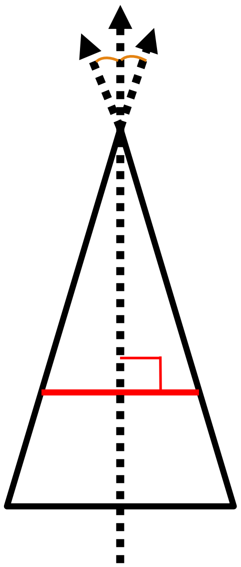
The length of line segment gets (local) maximum when **either (or both) of end points is on a vertex**.



Observation to Solution (2)

The length of line segment gets (local) minimum when
  (a) either (or both) of end points is on a vertex, or
  (b) **the line segment is perpendicular
  to the angle bisector line***.





* Proof sketch: consider when the differential of length is equal to 0.

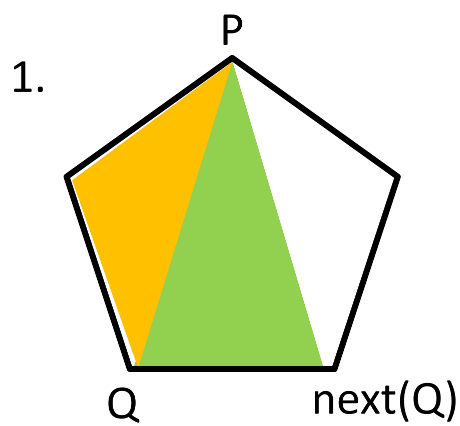
Solution: O(n) (O(n^2) may do)

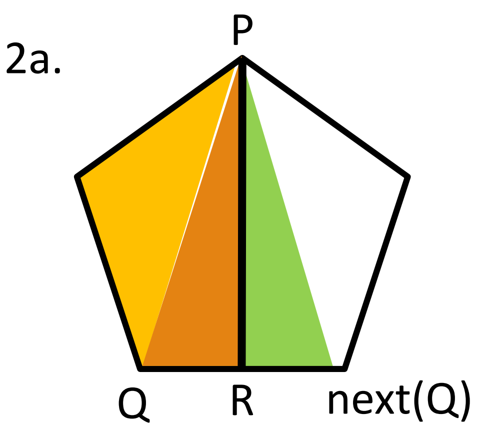
1. (Initialization) Choose a vertex P and find a vertex Q.
   - ✓ area(P…Q) <= total-area/2 && area(P…next(Q)) > total-area/2

2. Repeat following procedure until P moves around

   2a. (segment 1) find a point R on edge Q--next(Q)

   - ✓ area(P…R) = total-area/2 ➔ Update max and min

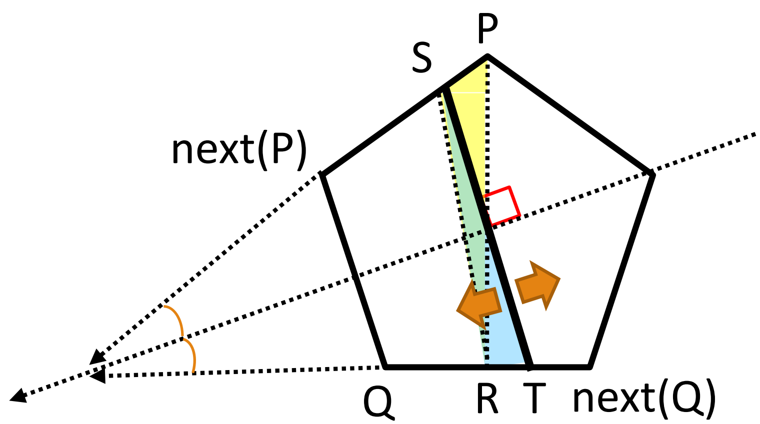




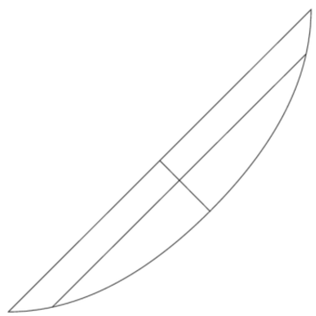
Solution: O(n)

2b. (segment 2) find point S on edge P--next(P) and point T on edge Q--next(Q)

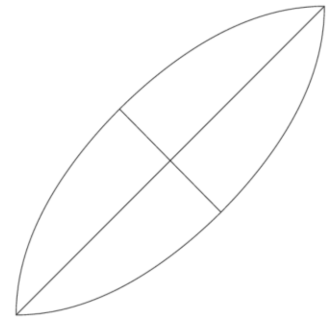
- ✓ area(S…T) = total-area/2
- ✓ S--T is perpendicular to bisector(P--next(P), next(Q)--Q)
  - Solve an equation or use bisection method ➔ update min



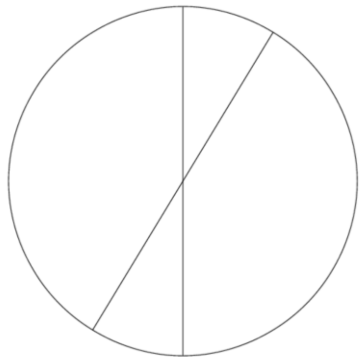
Judge's Testcase Gallery



n = 2171
Min = 15277.342
Max = 76840.956



n = 4260
Min =   40456.550
Max = 122057.944



n = 1360
Min = 15205.205
Max = 15252.000

n = 3
Min = 0.000500
Max = 74999.538

What happened in the last 30 minutes?

That is the question.