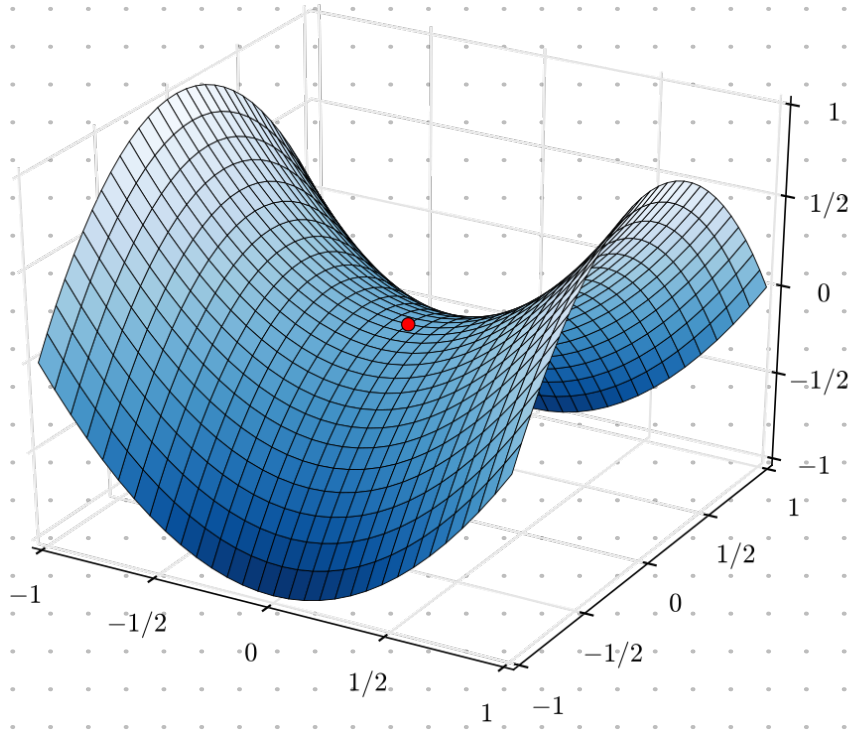
Saddle points faster than sorting



Algorithms and Data Structures Today

Singapore, July 23.

Saddle points faster than sorting

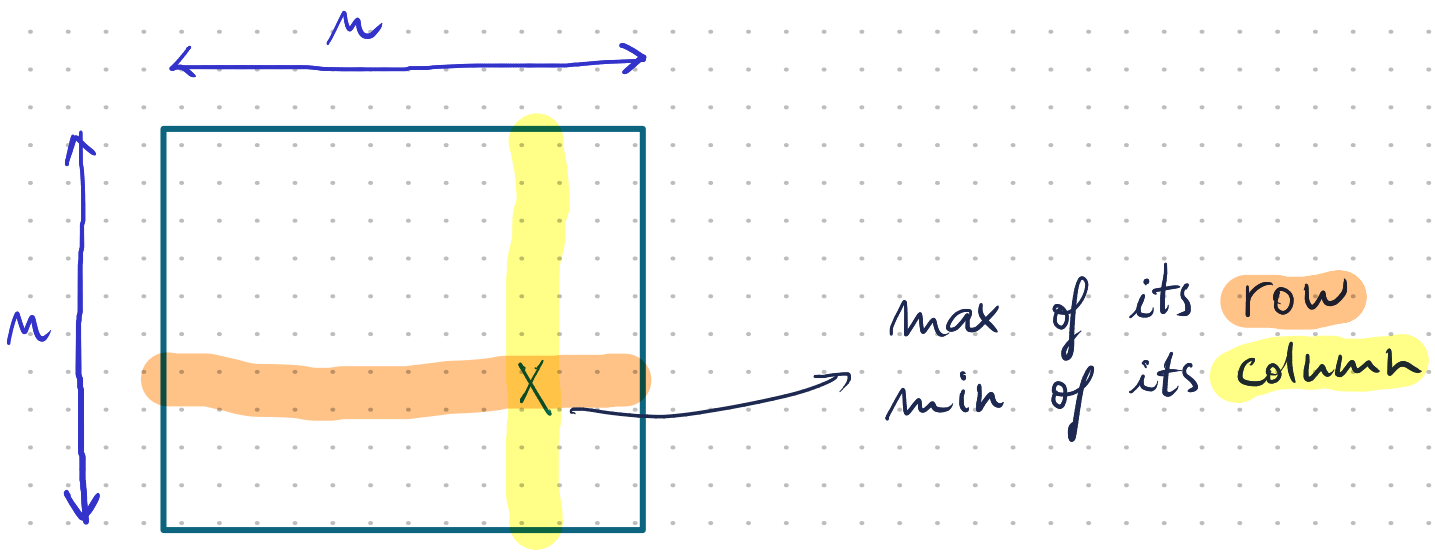
Justin Dallant — UL Bruxelles

Frederik Haagensen, Riko Jakob — ITU Copenhagen

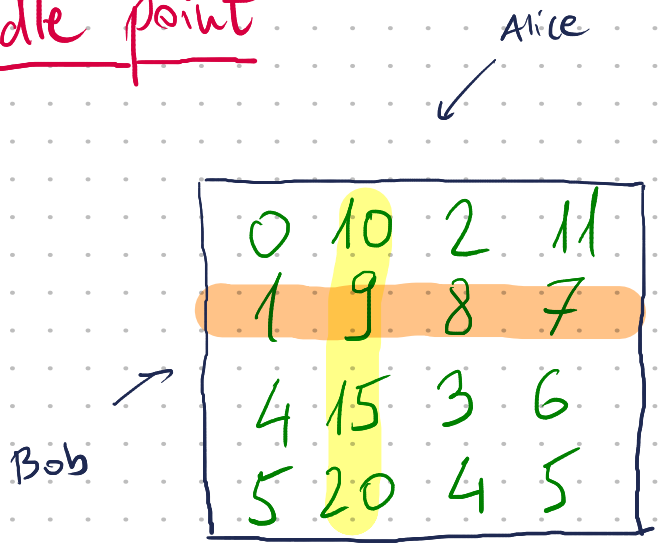
László Kozma — FU Berlin

Sebastian Wild — U Liverpool

Saddle point



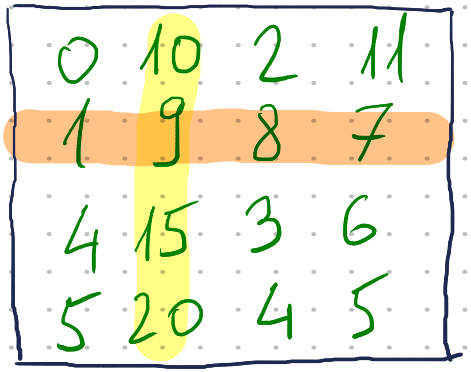
Saddle point



= Pure-strategy Nash-equilibrium

of zero-sum two-player game

Saddle point



[Knuth 1968]

Find a saddlepoint or report "none" in $O(n^2)$ time.

→ comparison/query model

→ this is optimal

Saddle point

0	10	2	11
1	9	8	7
4	15	3	6
5	20	4	5

[Knuth 1968]

Find a saddlepoint or report "none" in $O(n^2)$ time.

Lower bound (adversary)

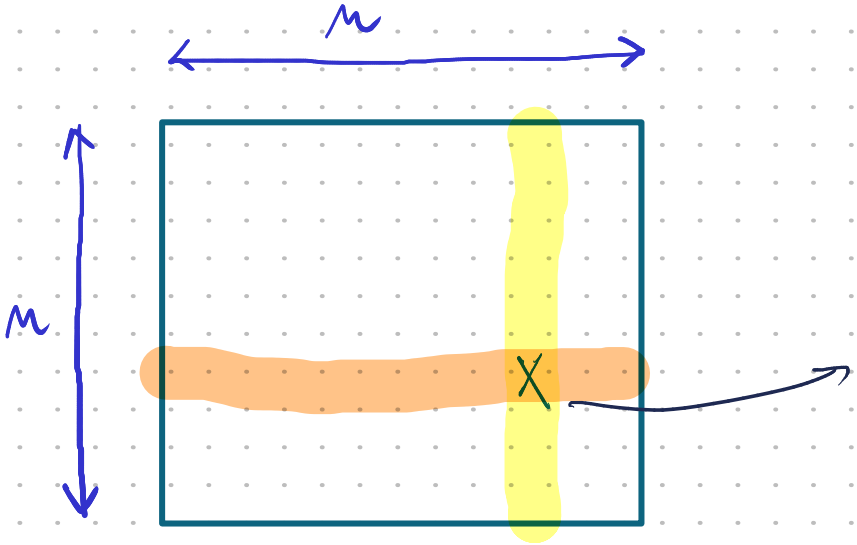
- each queried entry is 0
- last in each row is 2
- very last entry is ± 1

0	0	0	2	0
0	0	0	0	2
0	0	*	0	0
0	2	0	0	0
2	0	0	0	0

* → 1 : it is saddle point or "none"

* → −1 : all zeros in its row are saddlepoints

Strict Saddle point

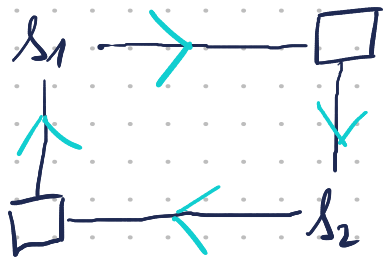


strict max of its row
strict min of its column

(e.g. if all entries distinct)
↳ Can assume w.l.o.g.

Obs. if exists, must be unique.

Prof:



contradiction ☒

Thm. Can find strict saddlepoint (or report "none")
in $O(n \log n)$ time.

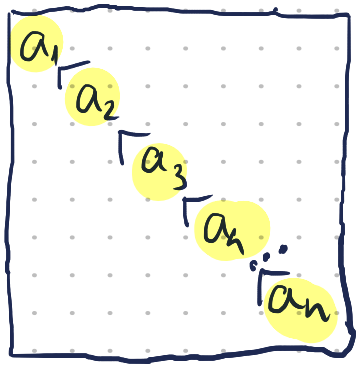
[Bienstock, Chung, Fredman, Schäffer, Shor, Suri '91]
[Byrne; Vaserstein '91]

Thm. Can find strict saddlepoint (or report "none") in $O(n \log n)$ time.

[Bienstock, Chung, Fredman, Schäffer, Shor, Suri '91]
[Byrne, Vaserstein '91]

Sketch:

1. Sort diagonal



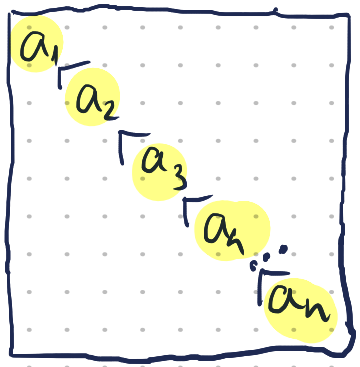
$O(n \log n)$ time

Thm. Can find strict saddlepoint (or report "none") in $O(n \log n)$ time.

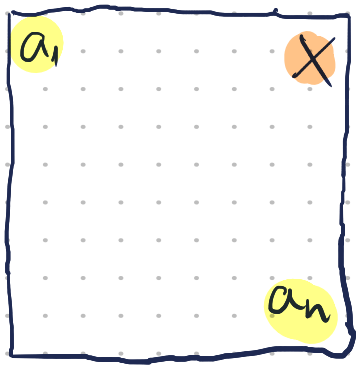
[Bienstock, Chung, Fredman, Schäffer, Shor, Suri '91]
[Byrne, Vaserstein '91]

Sketch:

1. Sort diagonal

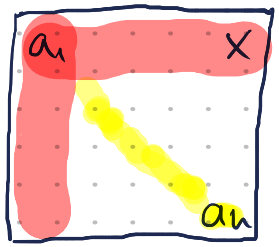


2. Compare a_1, a_n, X



$O(n \log n)$ time

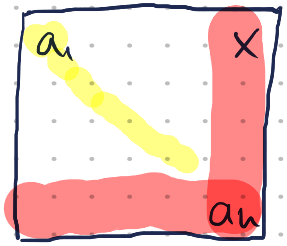
→ Case 2.1. $a_1 < a_n < X$



no saddlepoint here, can delete

$O(1)$ time: $n \to n-1$

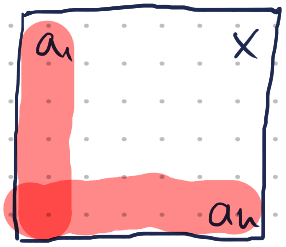
→ Case 2.2. $X < a_1 < a_n$

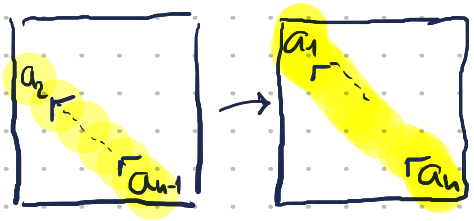


no saddlepoint here, can delete

$O(1)$ time: $n \to n-1$

→ Case 2.3 $a_1 < X < a_n$





no saddlepoint here, can delete

$O(\log n)$ time: $n \to n-1$

→ If one entry remains, go back and check if saddlepoint

Total: $2n-1$ queries

$O(n \log n)$ time

Thm. Can find strict saddlepoint (or report "none")
in $O(n \log n)$ time.

[Bienstock, Chung, Fredman, Schäffer, Shor, Suri '91]
[Byrne, Vaserstein '91]

Is the sorting step unavoidable?

Our results:

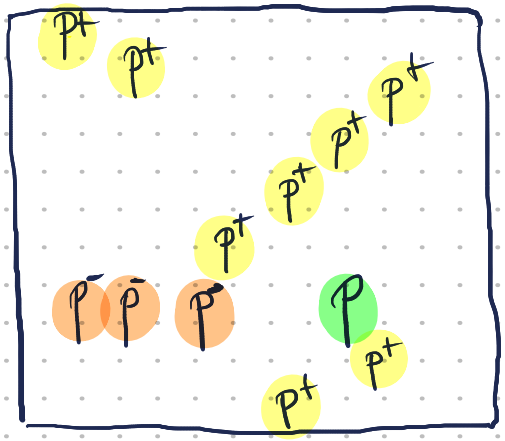
Find the strict saddlepoint (or report "none") in $n \times n$ time:

$O(n)$ randomized or $O(n \log^* n)$ deterministic

LAS VEGAS w.h.p. [ESA'24] [SOSA'24]

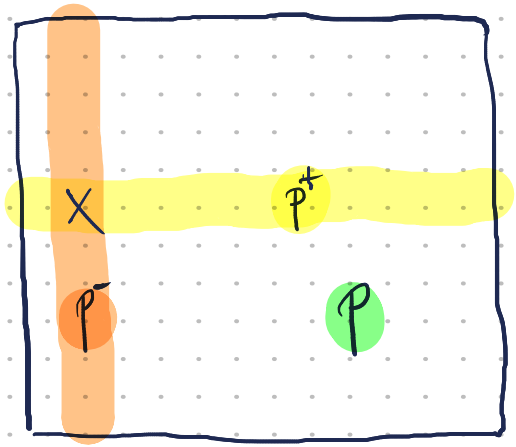
can also find value of non-strict saddlepoint but not certify its existence

Randomized O(n) time algorithm

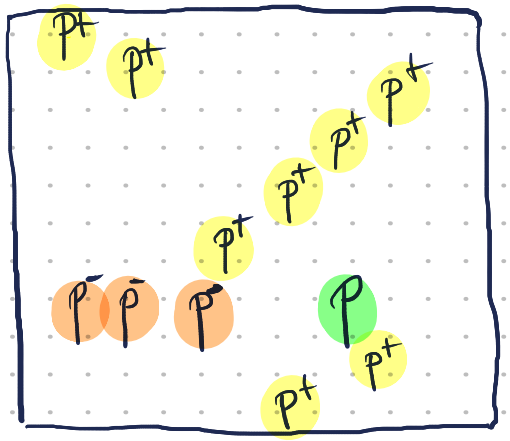


pivot p :- each row has a larger entry (p^+)
- row of p has many smaller entries (p^-)

$\Rightarrow$ Columns of p^- can be deleted



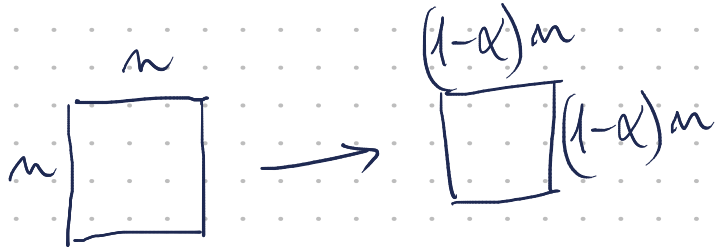
Randomized $O(n)$ time algorithm



pivot p: - each row has a larger entry (p^+)
- row of p has many smaller entries (p^-)

$\Rightarrow$ Columns of p^- can be deleted

Symmetric for horizontal / vertical pivot.



Suppose: - pivot found in $O(n)$ time
- remove α-fraction of rows/cols ($0<\alpha<1$)

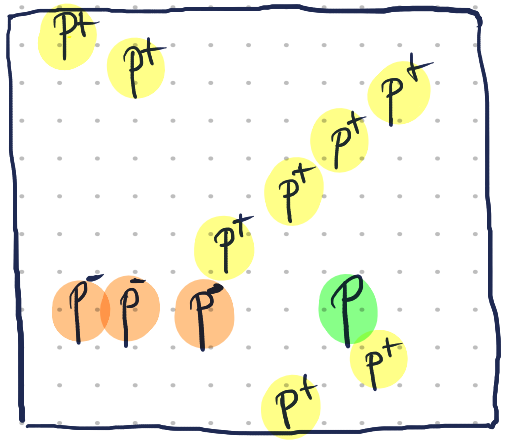
then time: $n + (1-\alpha)n + (1-\alpha)^2 n + \dots$

$\in O(n)$

How to find pivot in O(n) time?

Recall, pivot p: (1) each row has p^+

(2) α-fraction of p's row is $\bar{p}$



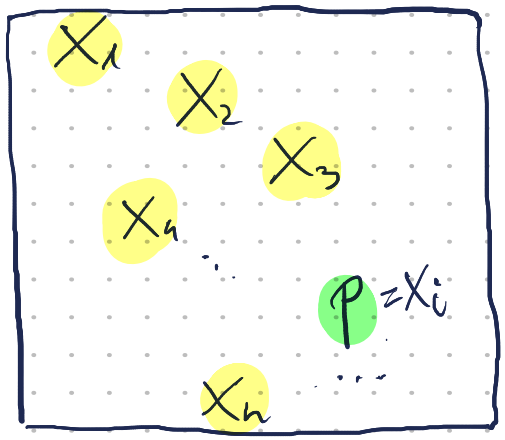
How to find pivot in $O(n)$ time?

Recall, pivot p: (1) each row has p^t

(2) α-fraction of p's row is $\bar{p}$

Idea 1

- in each row i pick random x_i
- $p = \min_i \{x_i\}$



satisfies (1)

Doesn't satisfy (2)

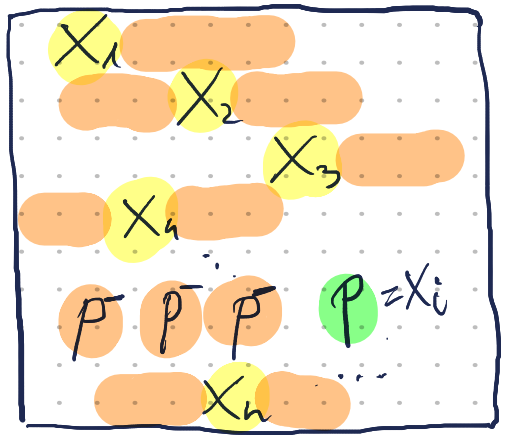
How to find pivot in $O(n)$ time?

Recall, pivot p: (1) each row has p^t

(2) α-fraction of p's row is $\bar{p}$

Idea 2:

- in each row i sample t entries, let X_i be max.
- $p = \min_i \{X_i\}$



Satisfies (1) and (2)

but $t \in \omega(1)$

$\Rightarrow$ not linear time

Idea 1: pick X_i randomly in row i

Idea 2: sample t from row i, let X_i be max

} Combine

Idea 1: pick x_i randomly in row i

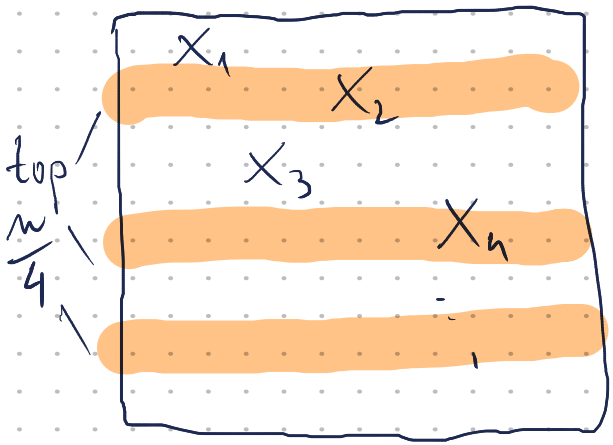
Idea 2: sample t from row i, let x_i be max

} Combine

Algorithm

phase 1: → until $n^{19/20}$ rows remain

- in each row i pick random x_i
- delete top quarter of rows by x_i → deleted rows likely to have large entry pt



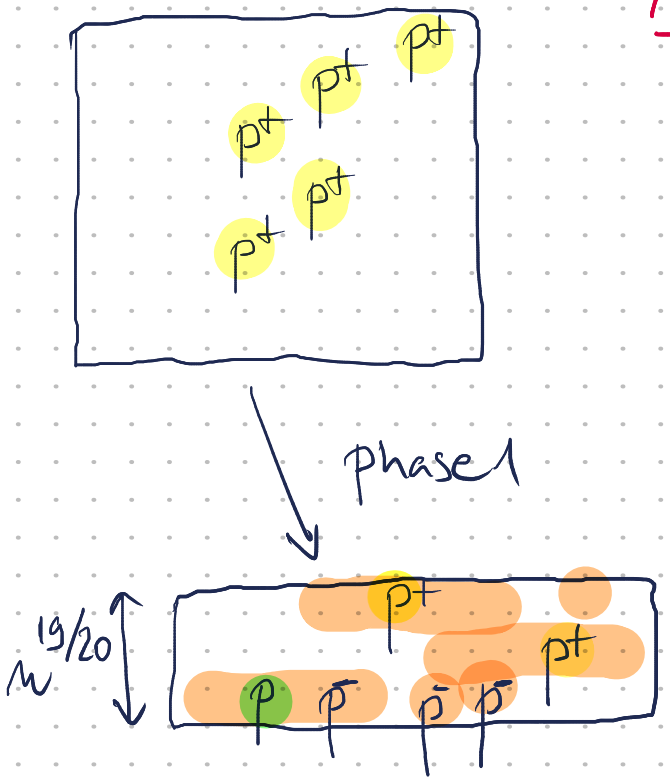
time: $n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \ldots$

$\in O(n)$

Idea 1: pick x_i randomly in row i

Idea 2: sample t from row i, let x_i be max

} Combine



Algorithm

phase 1: (delete rows with large sample) → until $n^{19/20}$ rows remain

phase 2:

- Sample $t = n^{1/20}$ from each row i
- x_i = entry of rank 40% of sample
- return pivot $p = \min_i \{x_i\}$ → (1) w.h.p. smaller than all removed row samples
  (2) w.h.p. larger than $\alpha = 30\%$ of its row

(Chernoff)

Algorithm (pivot selection)

phase 1: (until $n^{19/20}$ rows remain)

- in each row i pick random x_i
- delete top quarter of rows by x_i

time: $n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \dots$

$\in O(n)$

phase 2:

- Sample $t = n^{1/20}$ from each row i
- x_i = entry of rank 40% of sample
- return pivot $p = \min_i \{x_i\}$

time: $n^{19/20} \cdot n^{1/20}$

$\in O(n)$

Total: $O(n)$

(if fails, repeat)

$pr \leq e^{-n^{1/20}}$

Deterministic algorithm

DO BETTER?

DO BETTER?

ANNOYING: SADDLEPOINTS DO NOT ALWAYS EXIST...

DO BETTER?

ANNOYING: SADDLEPOINTS DO NOT ALWAYS EXIST...

SOLUTION: PSEUDO-SADDLEPOINT (PSP)

PSP: EVERY COLUMN HAS AN ENTRY SMALLER OR EQUAL
EVERY ROW HAS AN ENTRY LARGER OR EQUAL

DO BETTER?

ANNOYING: SADDLEPOINTS DO NOT ALWAYS EXIST...

SOLUTION: PSEUDO-SADDLEPOINT (PSP)

PSP: EVERY COLUMN HAS AN ENTRY SMALLER OR EQUAL
EVERY ROW HAS AN ENTRY LARGER OR EQUAL

$\Longleftrightarrow$

MAX OF COLUMN MINIMA $\leq$ • $\leq$ MIN OF ROW MAXIMA

DO BETTER?

ANNOYING: SADDLEPOINTS DO NOT ALWAYS EXIST...

SOLUTION: PSEUDO-SADDLEPOINT (PSP)

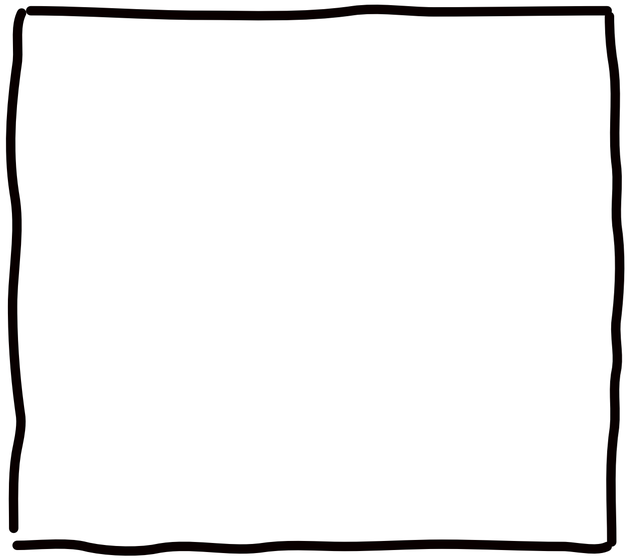
PSP: EVERY COLUMN HAS AN ENTRY SMALLER OR EQUAL
EVERY ROW HAS AN ENTRY LARGER OR EQUAL

$\Longleftrightarrow$

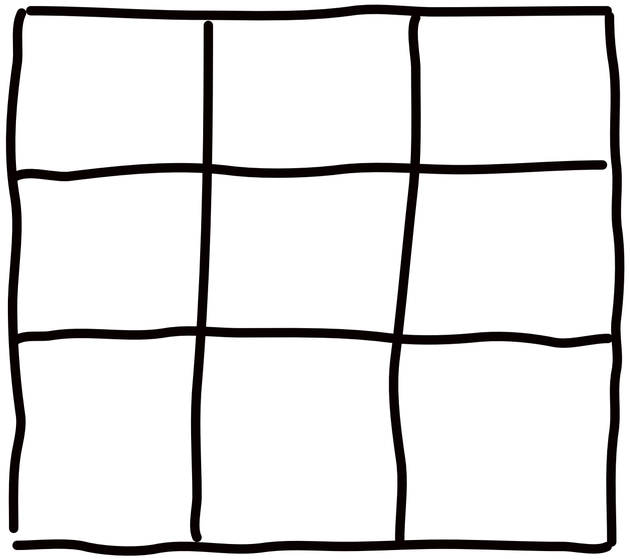
MAX OF COLUMN MINIMA $\leq$ • $\leq$ MIN OF ROW MAXIMA

PSP = STRICT SADDLEPOINT, WHEN IT EXISTS.

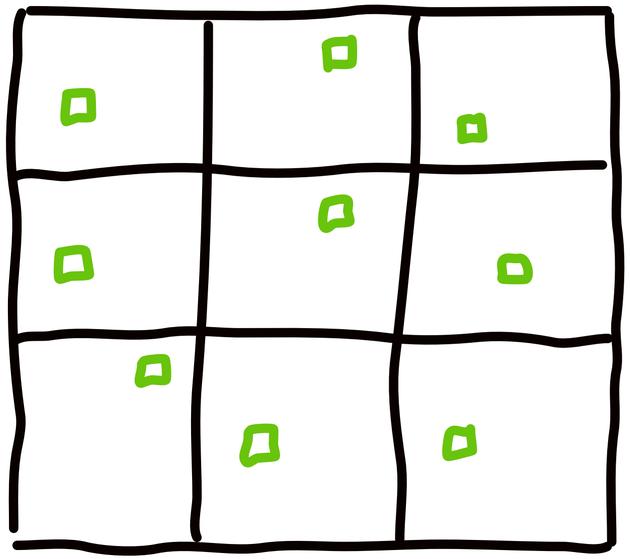
NICE FACT ABOUT PSPs :



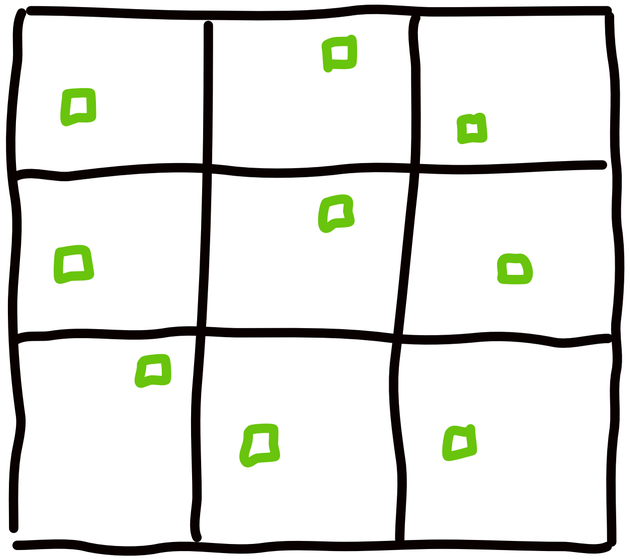
NICE FACT ABOUT PSPs :



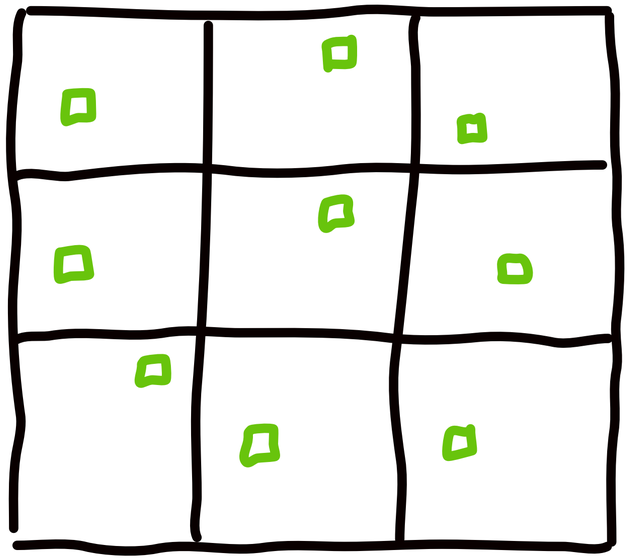
NICE FACT ABOUT PSPs :



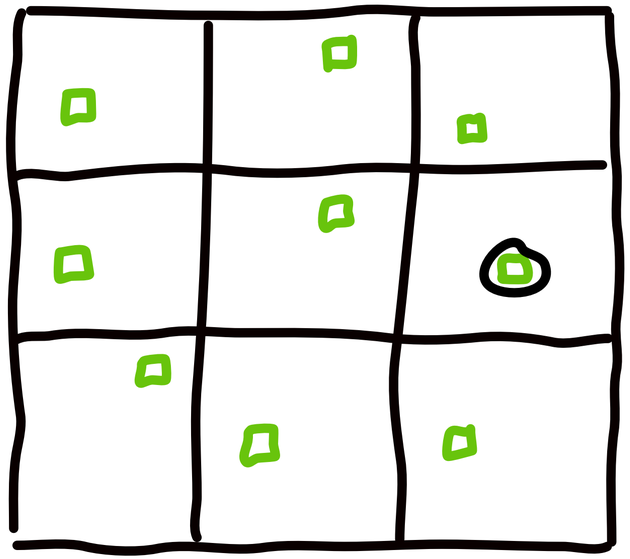
NICE FACT ABOUT PSPs :



NICE FACT ABOUT PSPs :

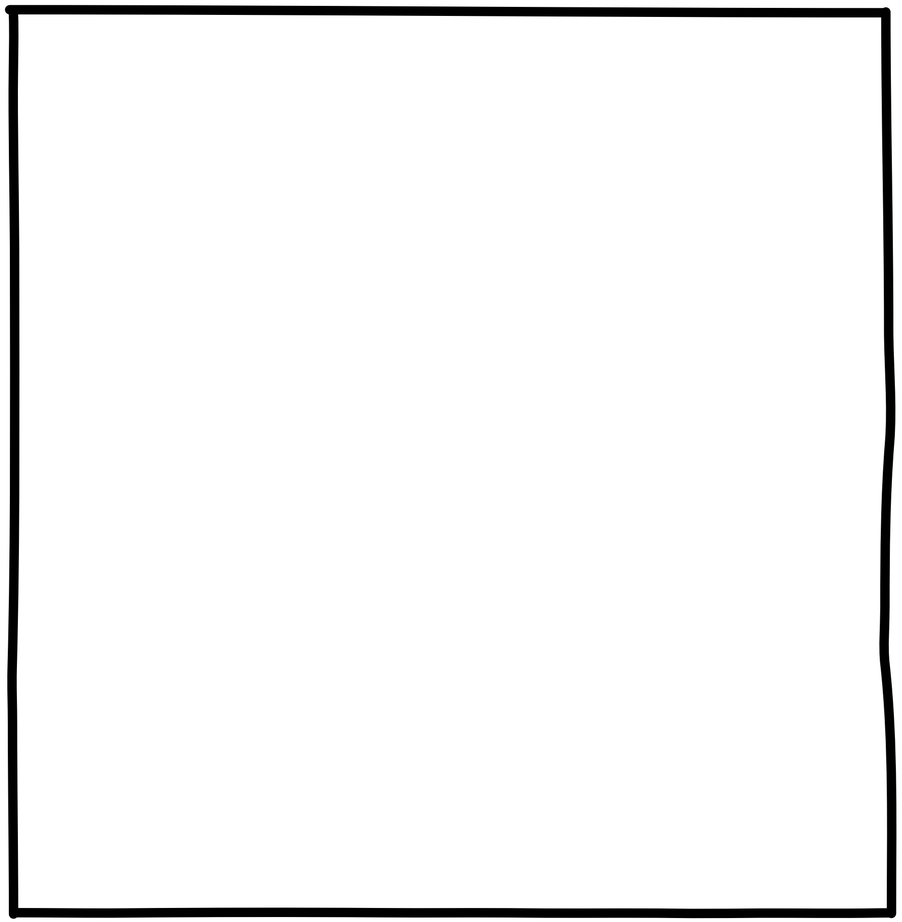


NICE FACT ABOUT PSPs :

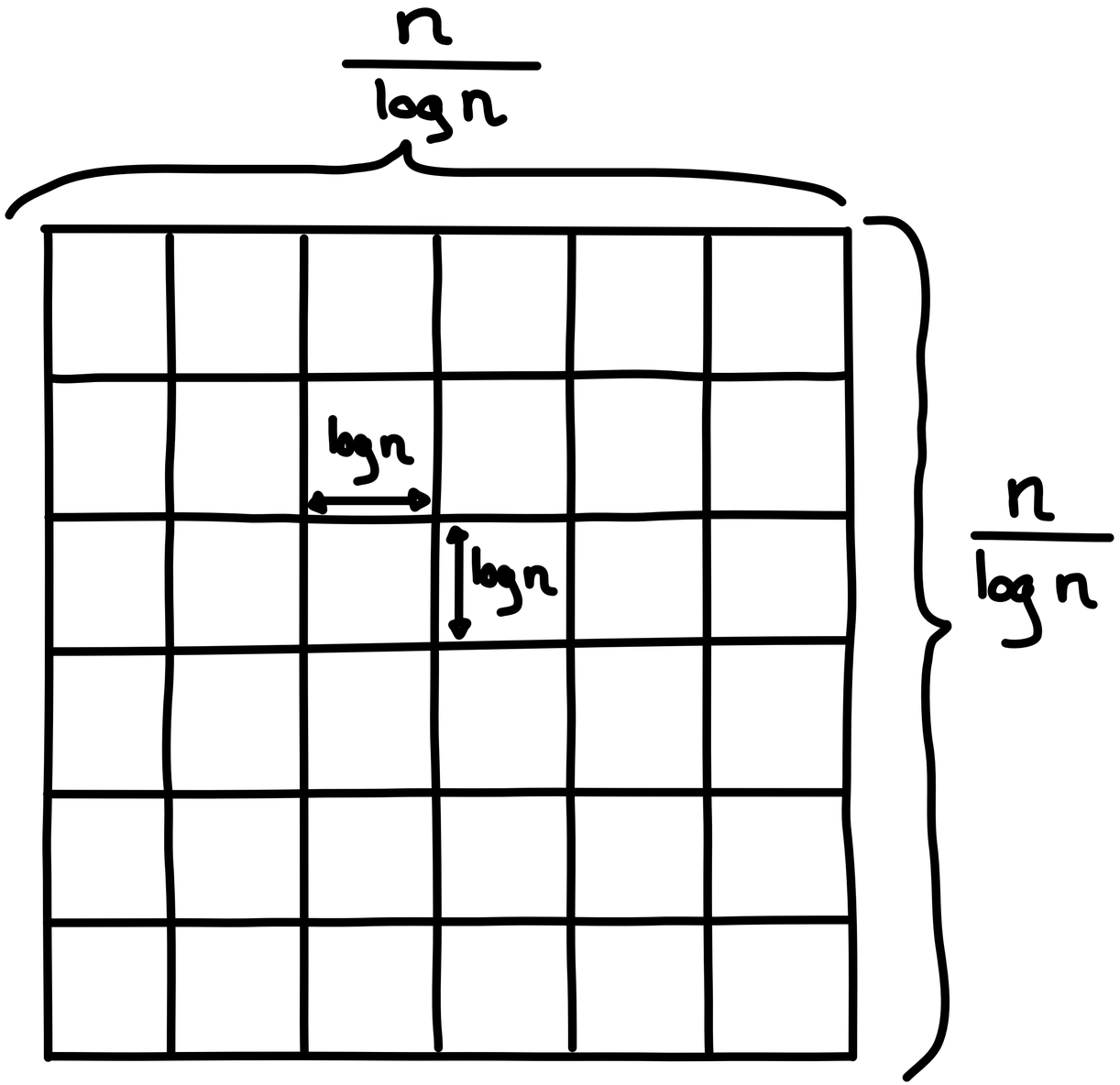


○ IS PSP IN ORIGINAL MATRIX.

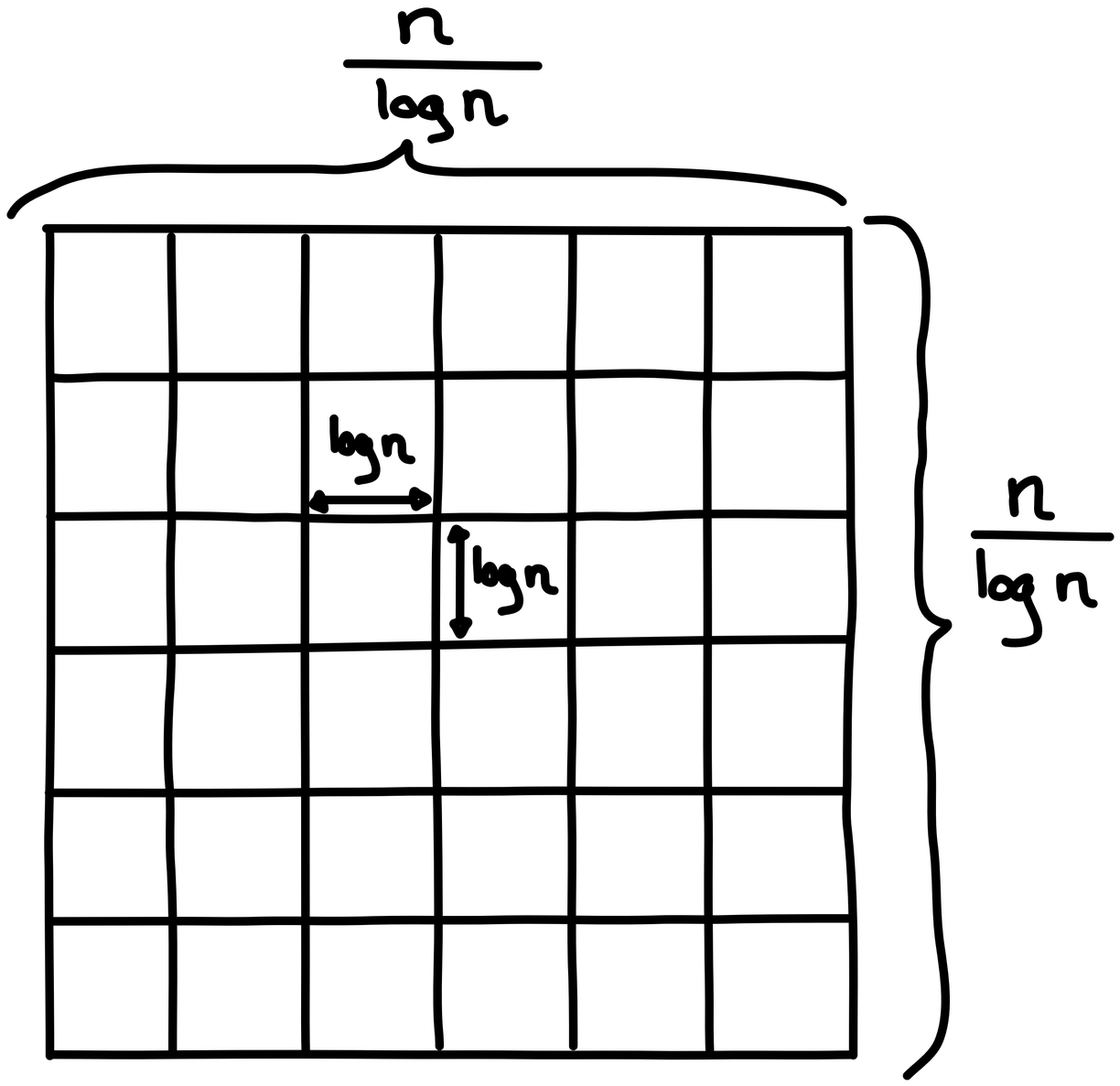
THE ALGORITHM:



THE ALGORITHM:

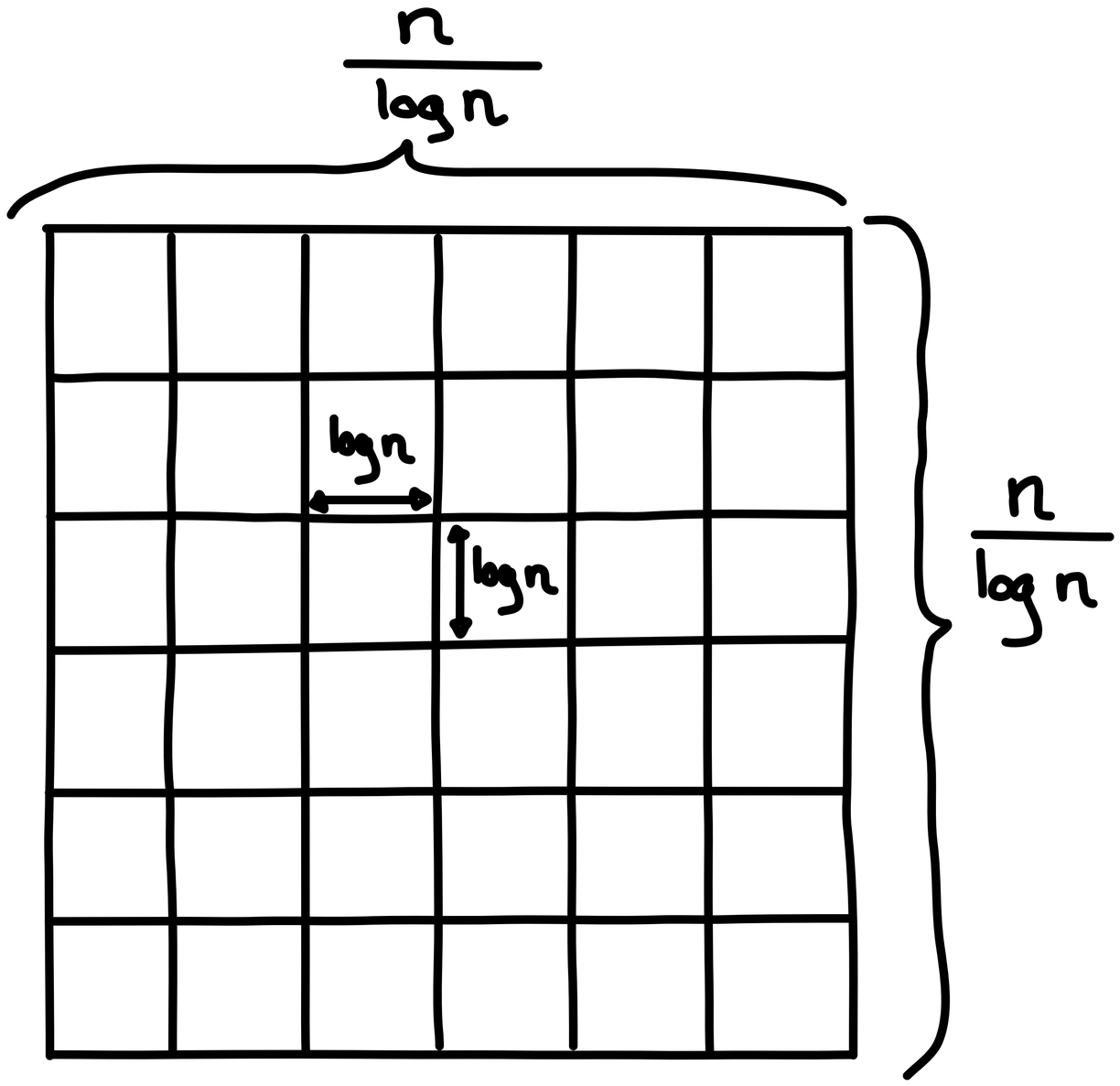


THE ALGORITHM:



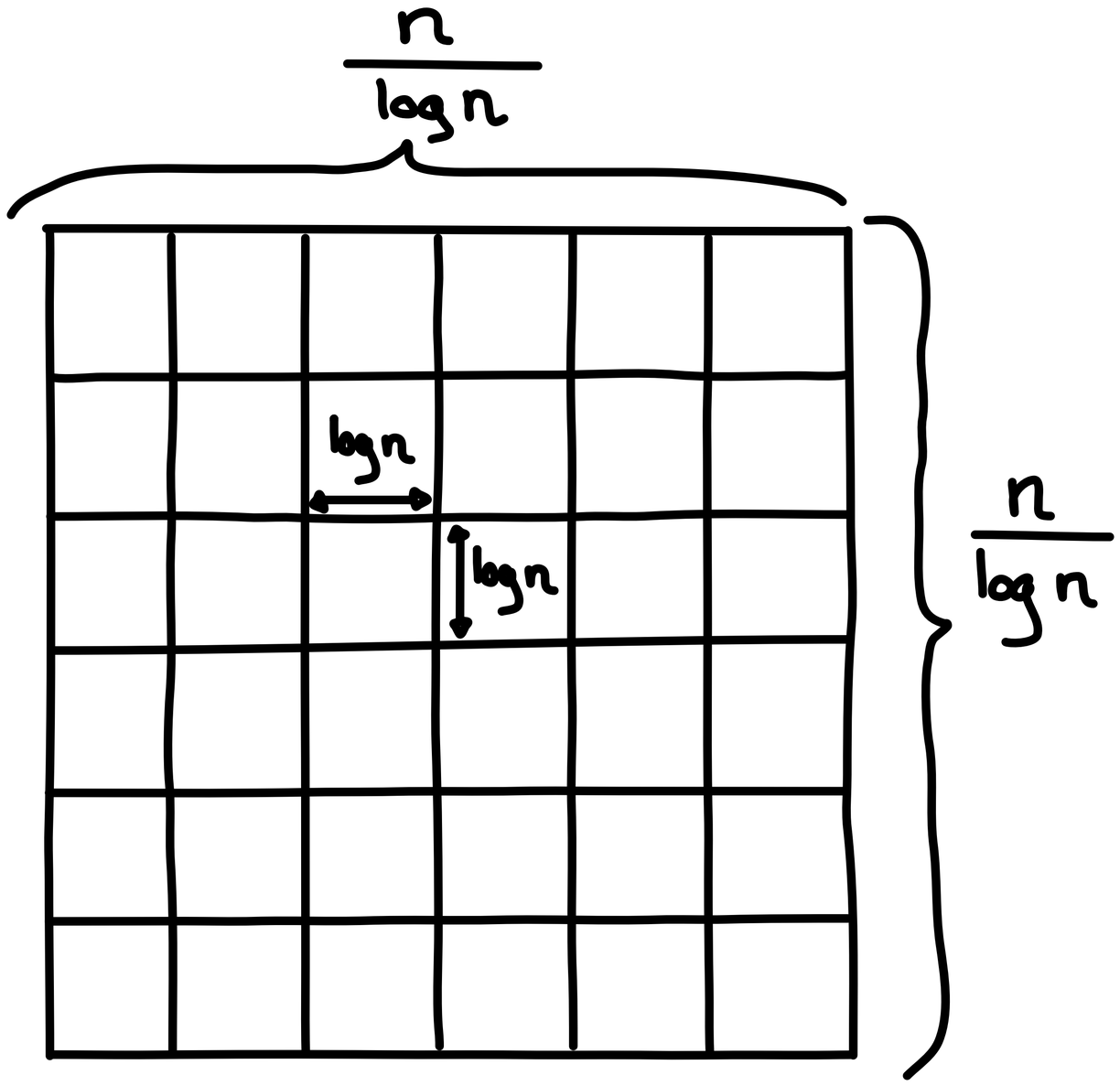
- APPLY THE BCFSSS ALGORITHM.

THE ALGORITHM:



- APPLY THE BCFSSS ALGORITHM.
- ON EACH OF THE $2\frac{n}{\log n}$ QUERIED SUBMATRICES, CALL THE ALGO RECURSIVELY.

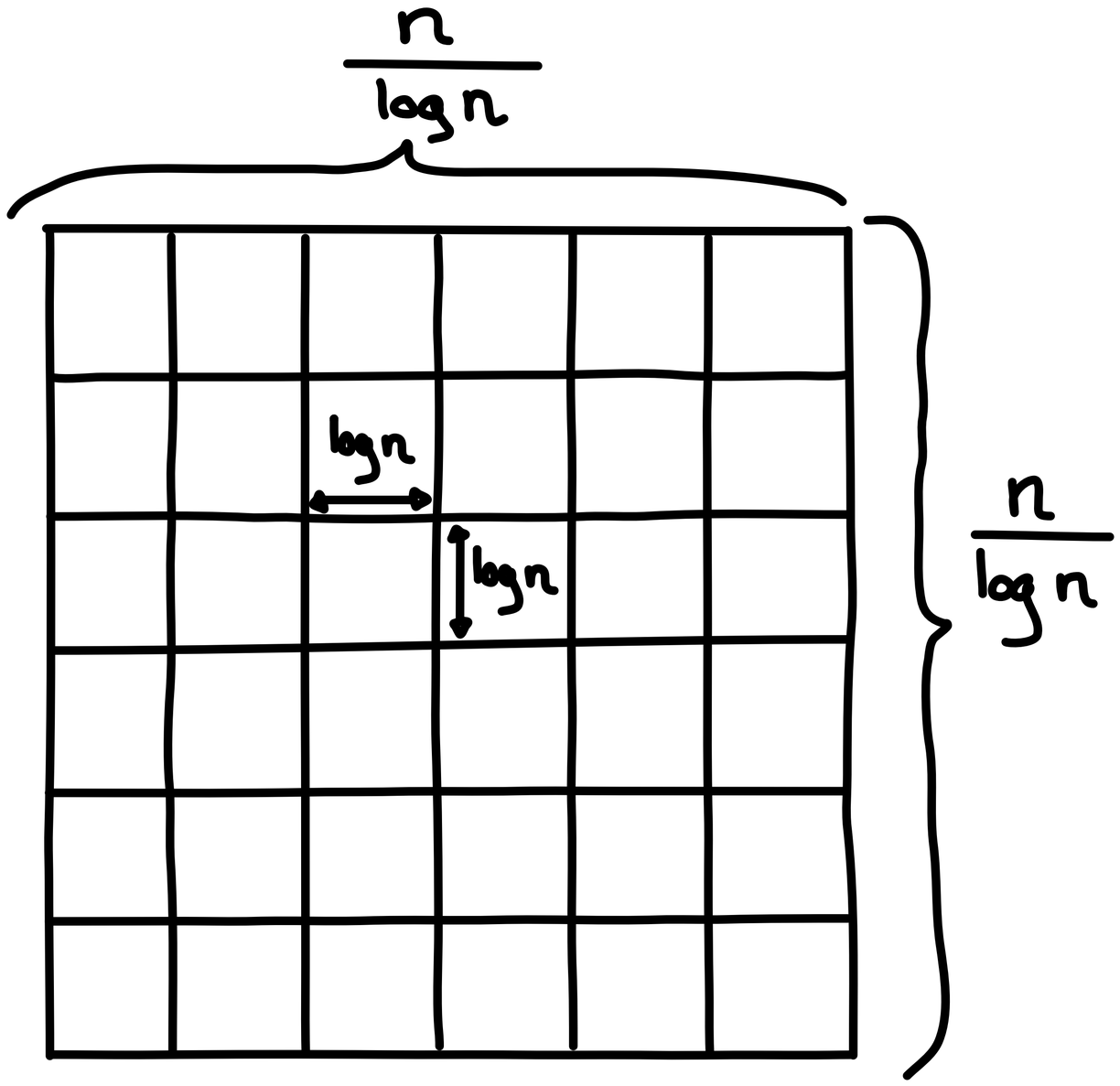
THE ALGORITHM:



- APPLY THE BCFSSS ALGORITHM.
- ON EACH OF THE $2\frac{n}{\log n}$ QUERIED SUBMATRICES, CALL THE ALGO RECURSIVELY.

$$T(n) \leq O(n) + 2\frac{n}{\log n} T(\log n)$$

THE ALGORITHM:



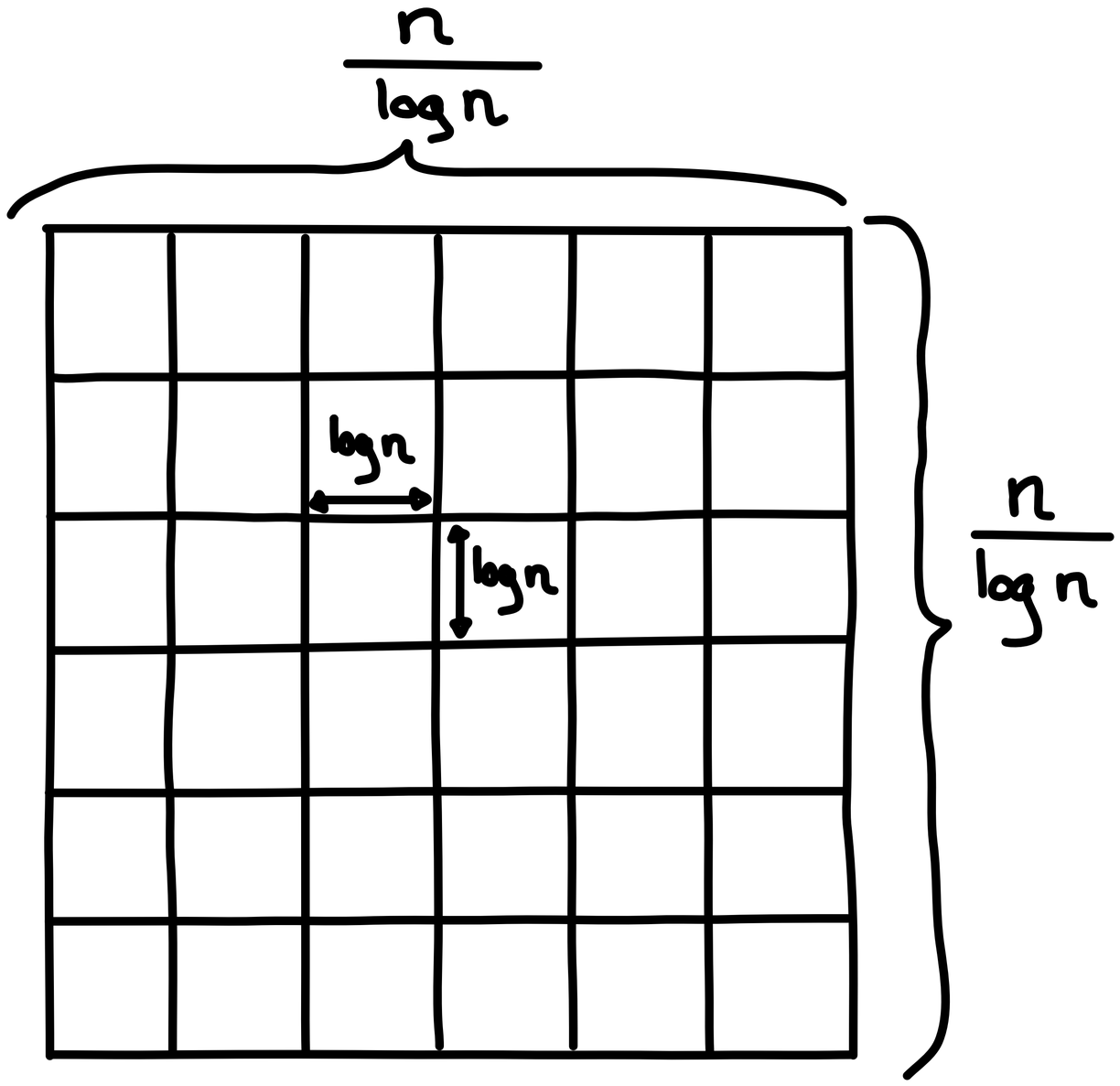
- APPLY THE BCFSSS ALGORITHM.
- ON EACH OF THE $2\frac{n}{\log n}$ QUERIED SUBMATRICES, CALL THE ALGO RECURSIVELY.

$$T(n) \leq O(n) + 2\frac{n}{\log n} T(\log n)$$

$$\Rightarrow$$

$$T(n) = O(n * 2^{\log^* n})$$

THE ALGORITHM:



- APPLY THE BCFSSS ALGORITHM.
- ON EACH OF THE $2\frac{n}{\log n}$ QUERIED SUBMATRICES, CALL THE ALGO RECURSIVELY.

$$T(n) \leq O(n) + 2\frac{n}{\log n} T(\log n)$$

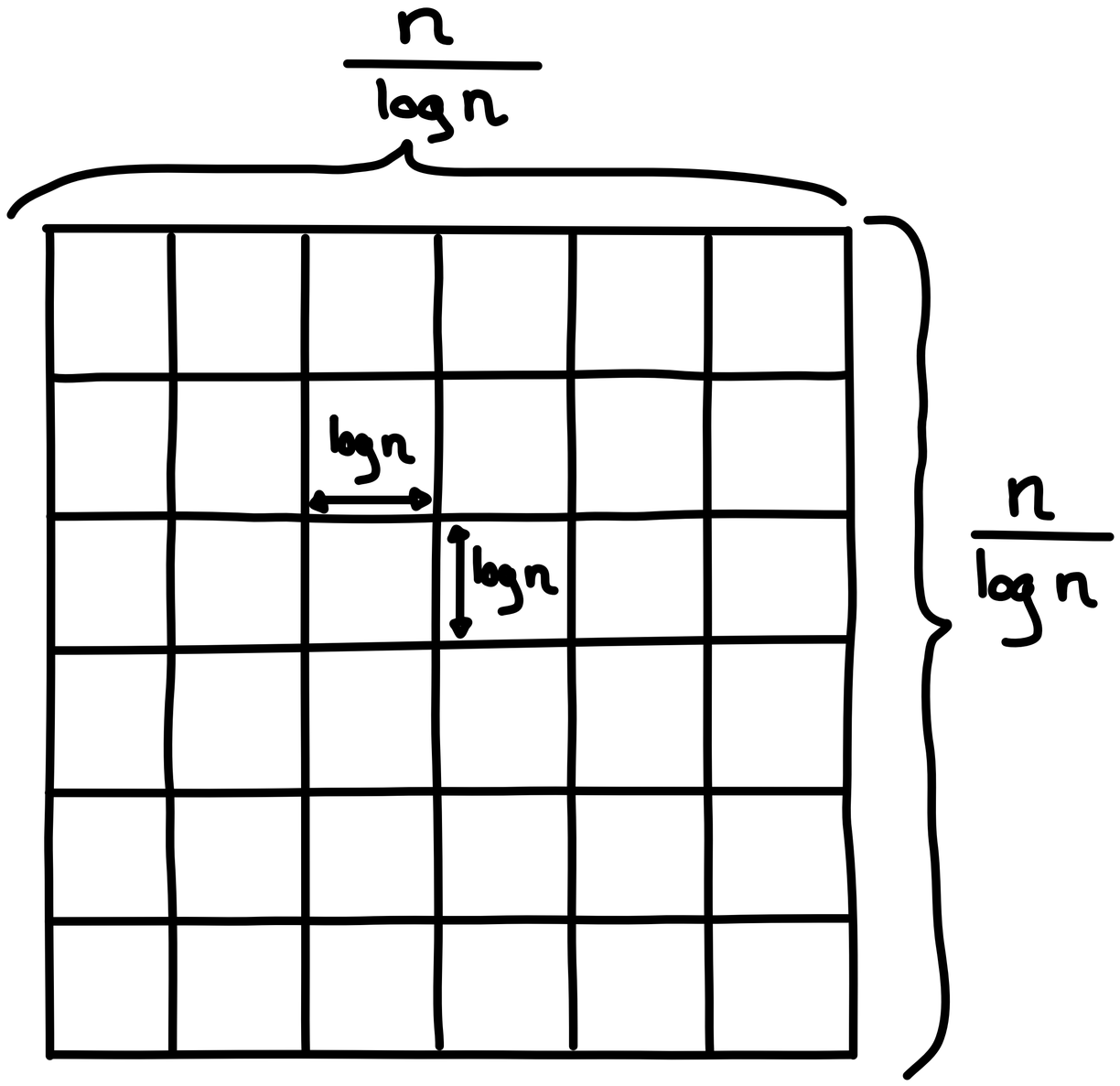
$$\Rightarrow$$

$$T(n) = O(n \times 2^{\log^* n})$$

$$x = 2^{2^{2^{\cdot^{\cdot^{2}}}}} \Big\} k$$

$$\log^*(x) = k.$$

THE ALGORITHM:



- APPLY THE BCFSSS ALGORITHM.
- ON EACH OF THE $2\frac{n}{\log n}$ QUERIED SUBMATRICES, CALL THE ALGO RECURSIVELY.

$$T(n) \leq O(n) + \cancel{2}^{1} \frac{n}{\log n} T(\log n)$$

$$\Rightarrow$$

$$T(n) = \cancel{O(n \cdot 2^{\log^* n})}\ O(n \log^* n)$$

$$x = 2^{2^{2^{\cdot^{\cdot^{2}}}}} \Big\}k$$

$$\log^*(x) = k.$$

- Can you do deterministic $O(n)$?

- Other game solution concepts, e.g. Shapley's (weak) saddle...

Sometimes

failed lower bound = surprising new algorithm(s)

- Can you do deterministic $O(n)$?

- Other game solution concepts, e.g. Shapley's (weak) saddle...

Sometimes

failed lower bound = surprising new algorithm(s)

Thanks