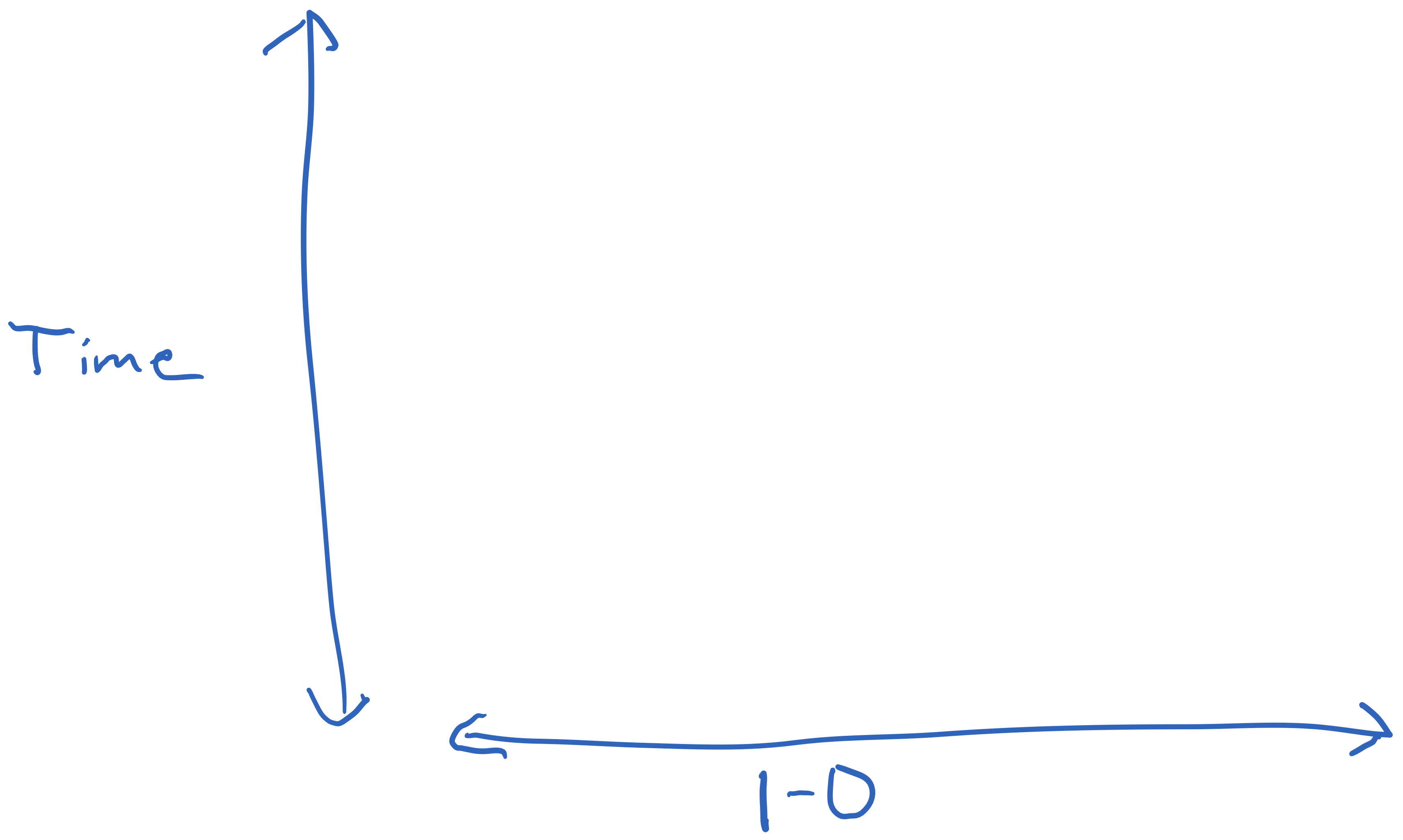
The Geometry of Data Structures

John Iacono

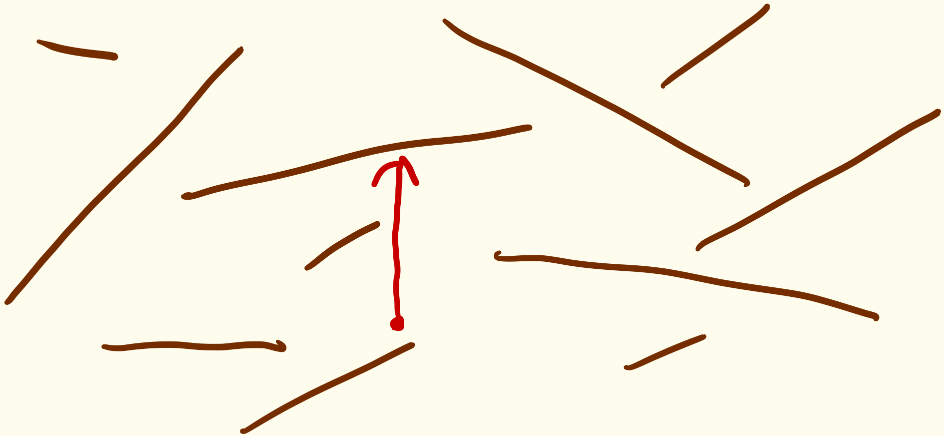
Université Libre de Bruxelles

1-D Data structure + Time = 2D

1-D Data structure + Time = 2D

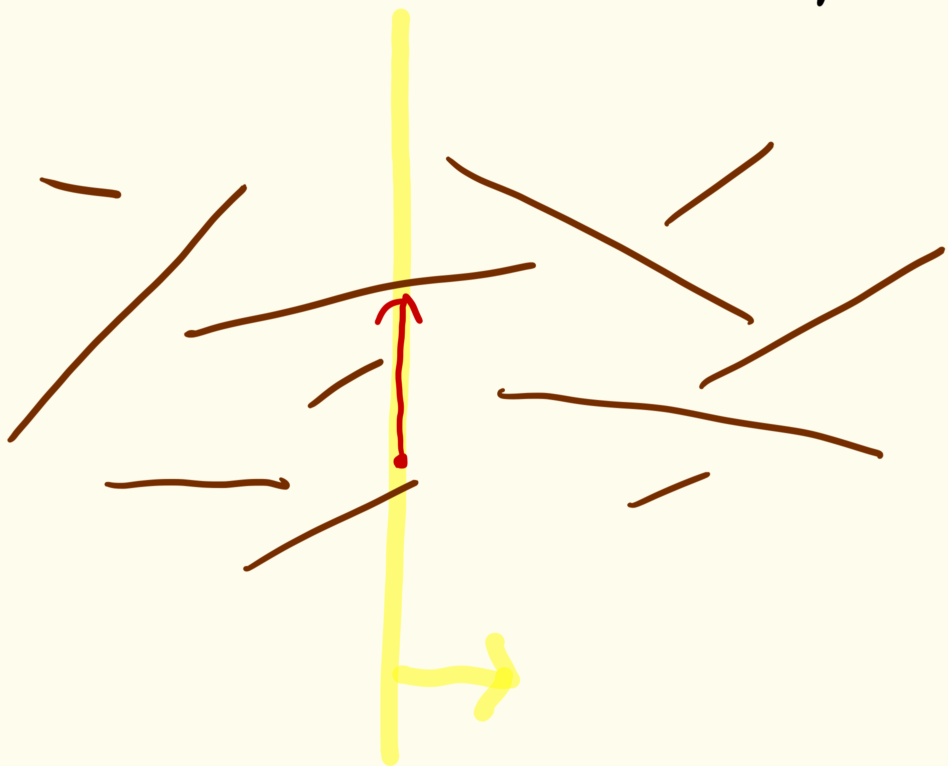


Classic example: Line Sweep



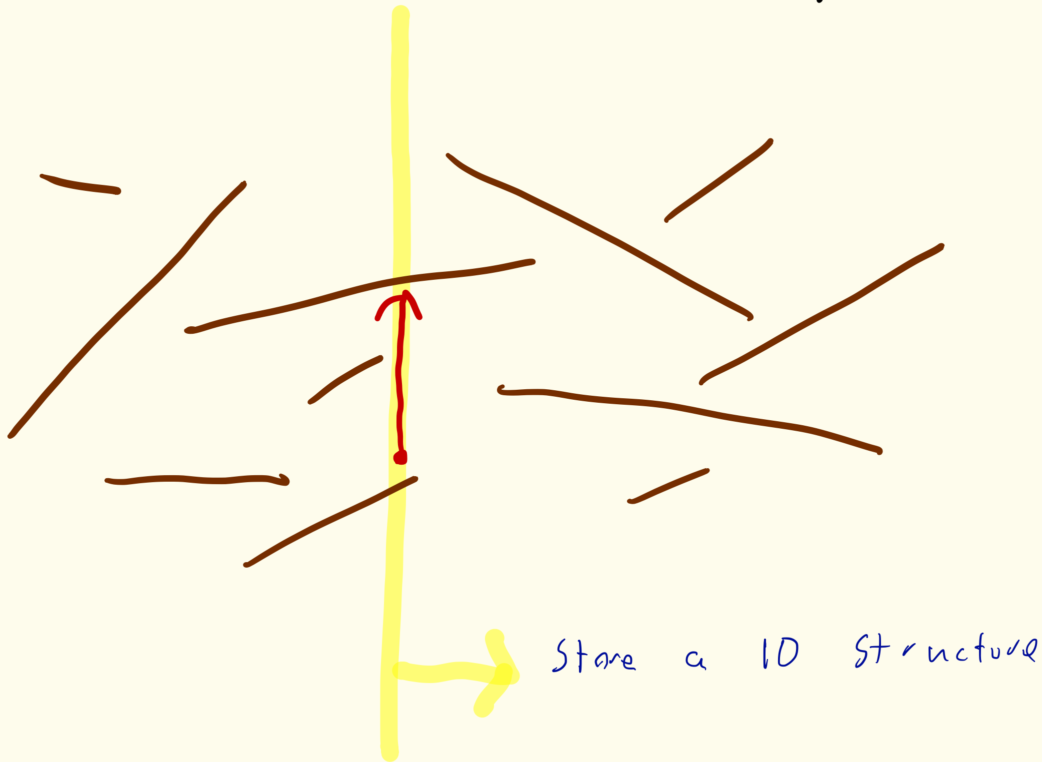
Persistent search Trees [Sarnack + Tarjan 86]

Classic example: Line Sweep



Persistent search Trees [Sarnak + Tarjan 86]

Classic example: Line Sweep



Persistent search Trees [Sarnack + Tarjan 86]

Line Sweep: Use 1-D "standard" Data Structures to solve geometric problems

Here: Use 2D geometry to solve "standard" data structure problems

Why?

Why?

- Can simplyfy
- Can see things that would be hard to see otherwise
- Can use standard geometric tools and intuition

OUTLINE

- Binary Search Trees

= Persistent Cache-Oblivious

≡ Forbidden Submatricies and friends

Part — Binary Search Trees

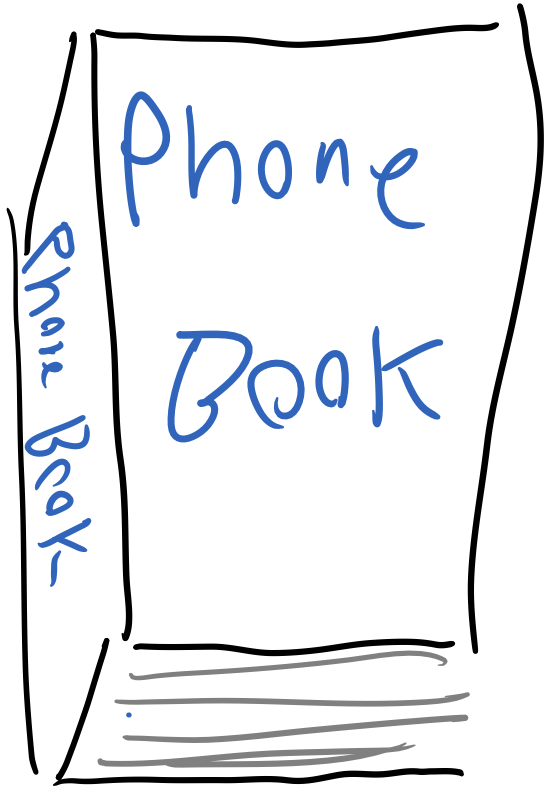
Part — Binary Search Trees

Many collaborators

P. J. Bose
M. L. Fredman
E. D. Demaine
S. Langerman
M. Patrascu
D. Harmon

1 8 7 8

1878



How To Find a Name in a Phone Book

How To Find a Name in a Phone Book

- Method 1: Read it like a book

How To Find a Name in a Phone Book

- Method 1: Read it like a book

- Method 2: Look in the middle, Throw away half the book

How To Find a Name in a Phone Book

- Method 1: Read it like a book

  Phone Book Doubles in size: Twice as much time

- Method 2: Look in the middle, Throw away half the book

  Phone Book Doubles in size: One more lookup

How To Find a Name in a Phone Book

- Method 1: Read it like a book

  Phone Book Doubles in size: Twice as much time

  1000000 names → 1000000 lookups

- Method 2: Look in the middle, Throw away half the book

  Phone Book Doubles in size: One more lookup

  1000000 names → 20 lookups

How To Find a Name in a Phone Book

- Method 1: Read it like a book

  Phone Book Doubles in size: Twice as much time

  1000000 names → 1000000 lookups

- Method 2: Look in the middle, Throw away half the book

  phone Book Doubles in size: One more lookup

  1000000 names → 20 lookups

BETTER

How about a Flow Chart?

Search in: A B C D E F G

How about a Flow Chart?

Search in: A B C D E F G

D

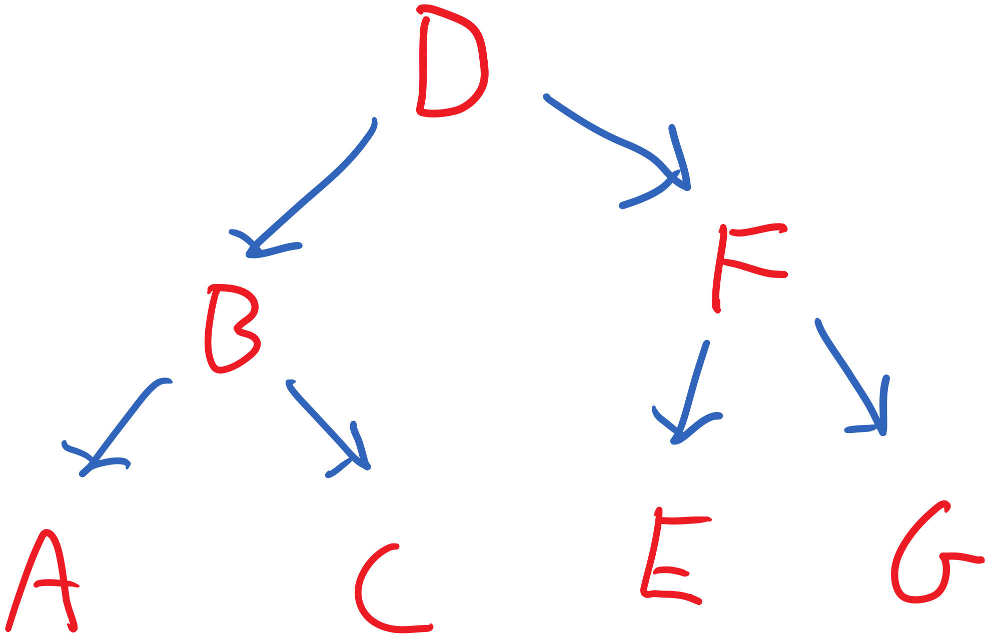
How about a Flow Chart?

Search in: $\underbrace{A \quad B \quad C}_{\text{Smaller}} \quad \underset{\uparrow}{D} \quad \underbrace{E \quad F \quad G}_{\text{Bigger}}$

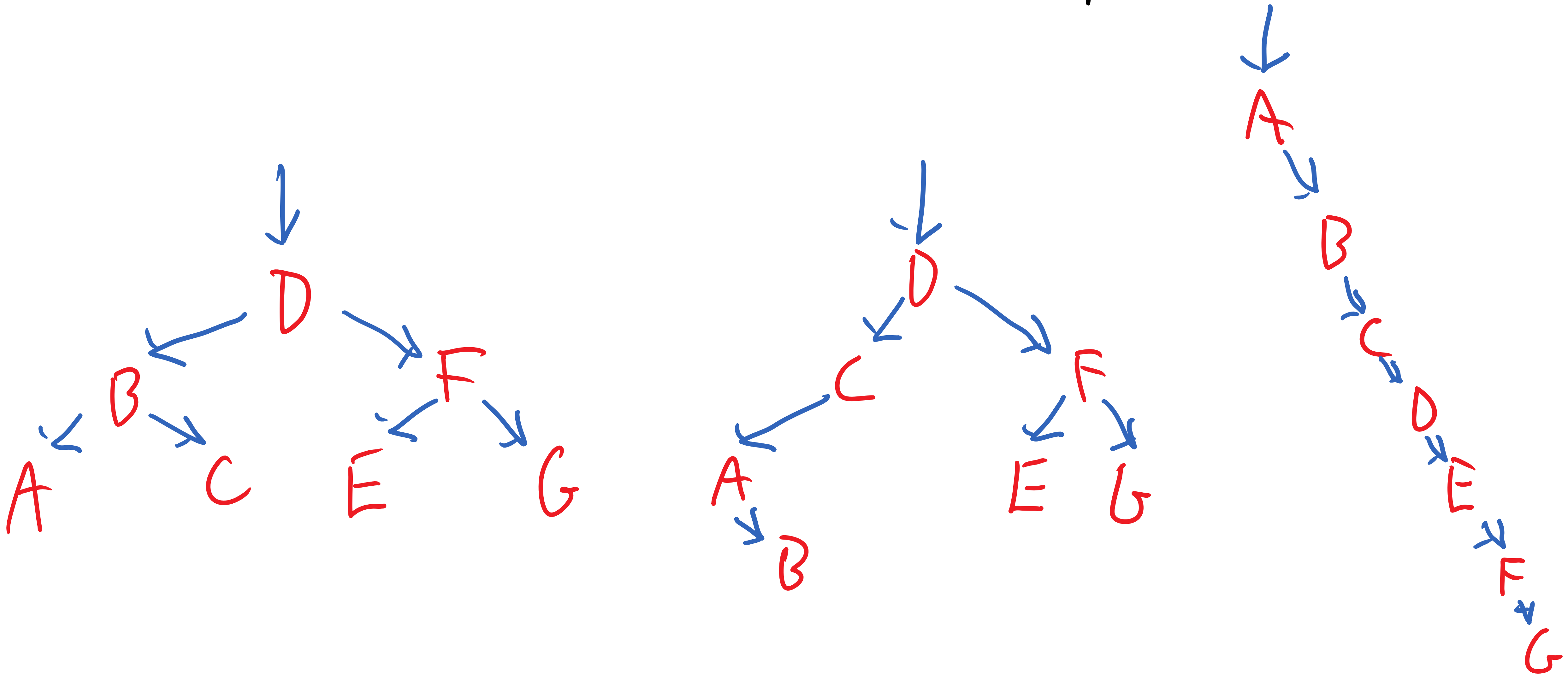
D

How about a Flow Chart?

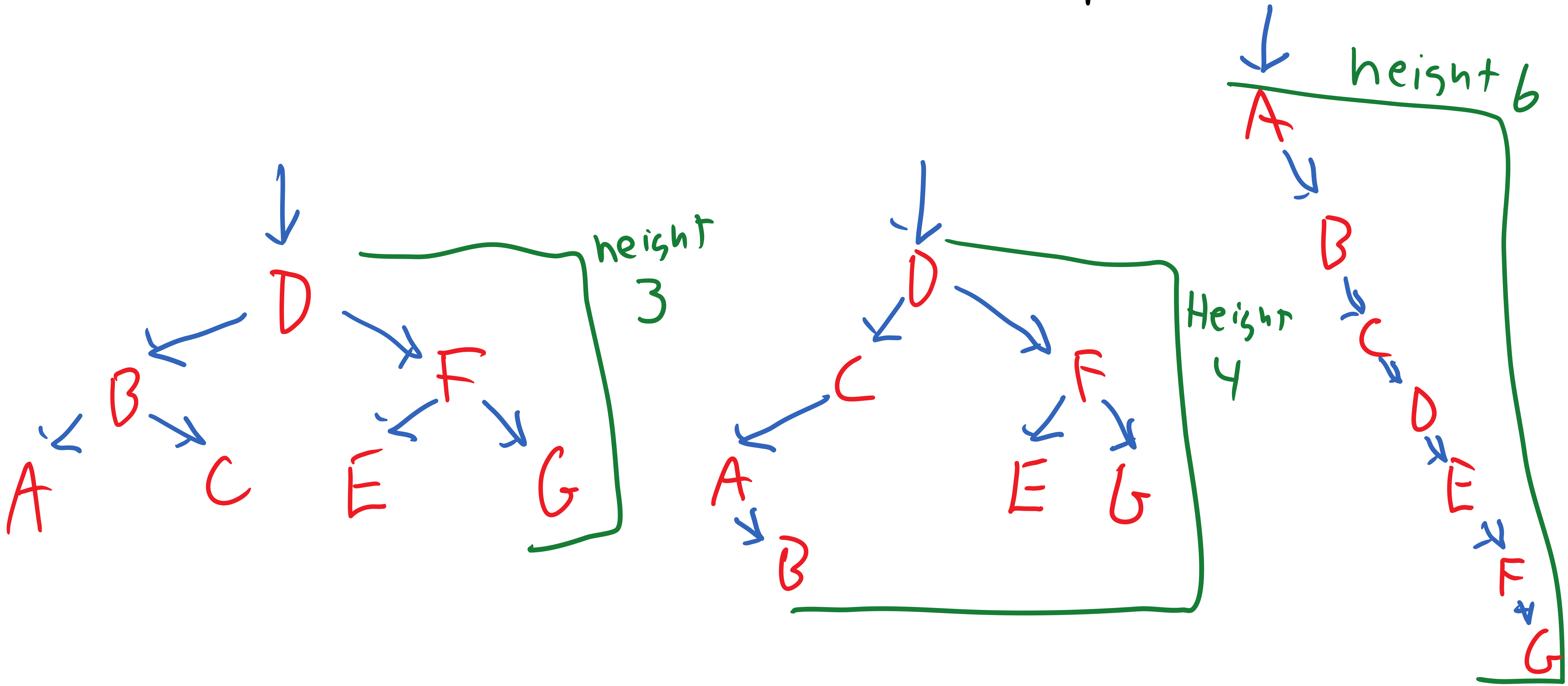
Search in: $\underbrace{A \quad B \quad C}_{\text{smaller}} \quad D \quad \underbrace{E \quad F \quad G}_{\text{bigger}}$



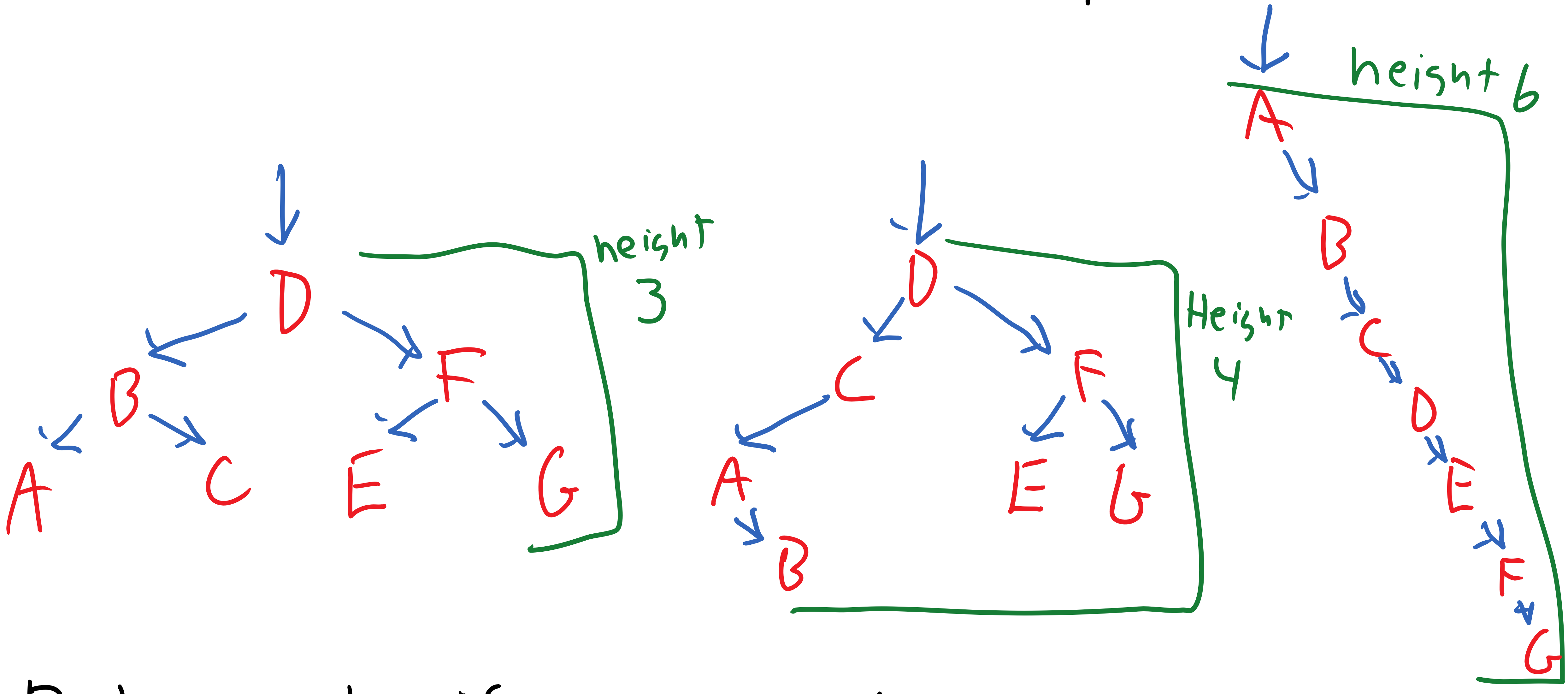
Which is better?



Which is better?

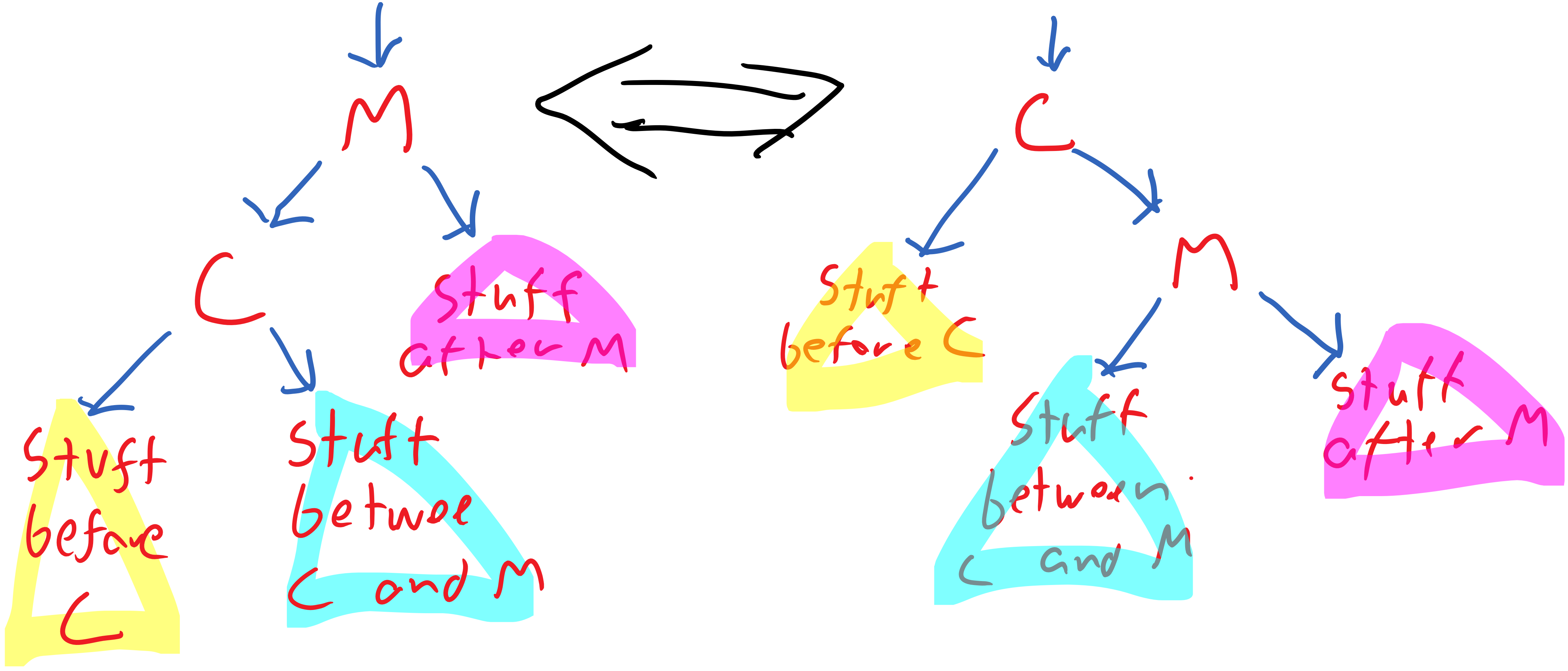


Which is better?

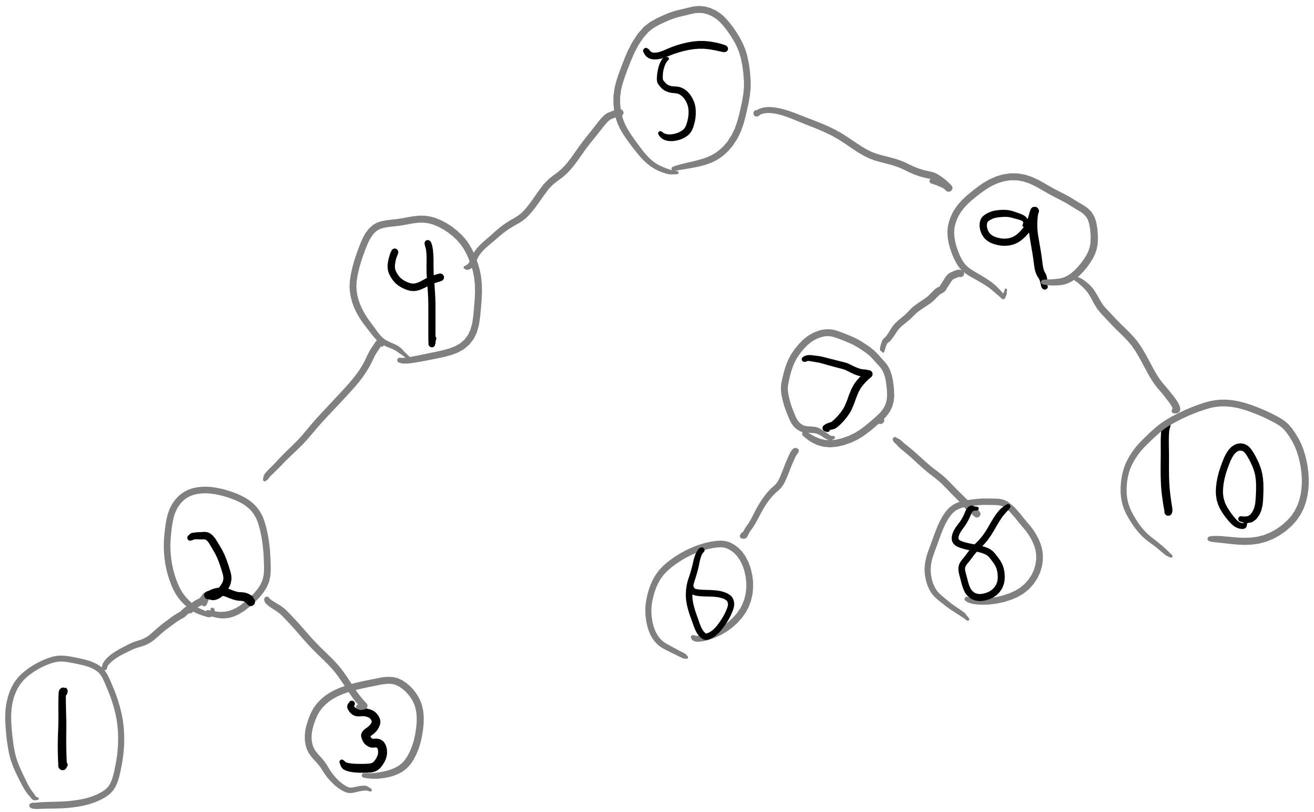


But what if we only search for A?

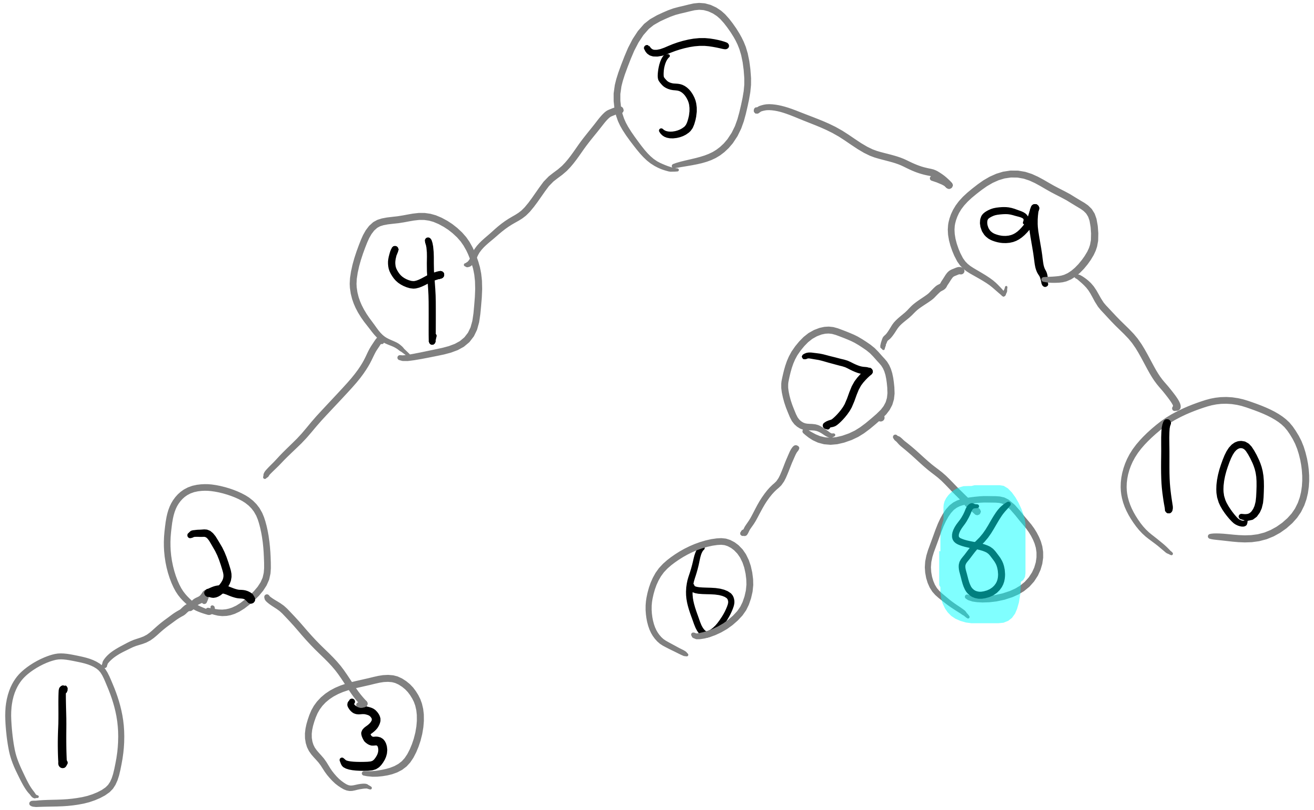
Changing Your Mind on How to Search



Binary Search Tree Model



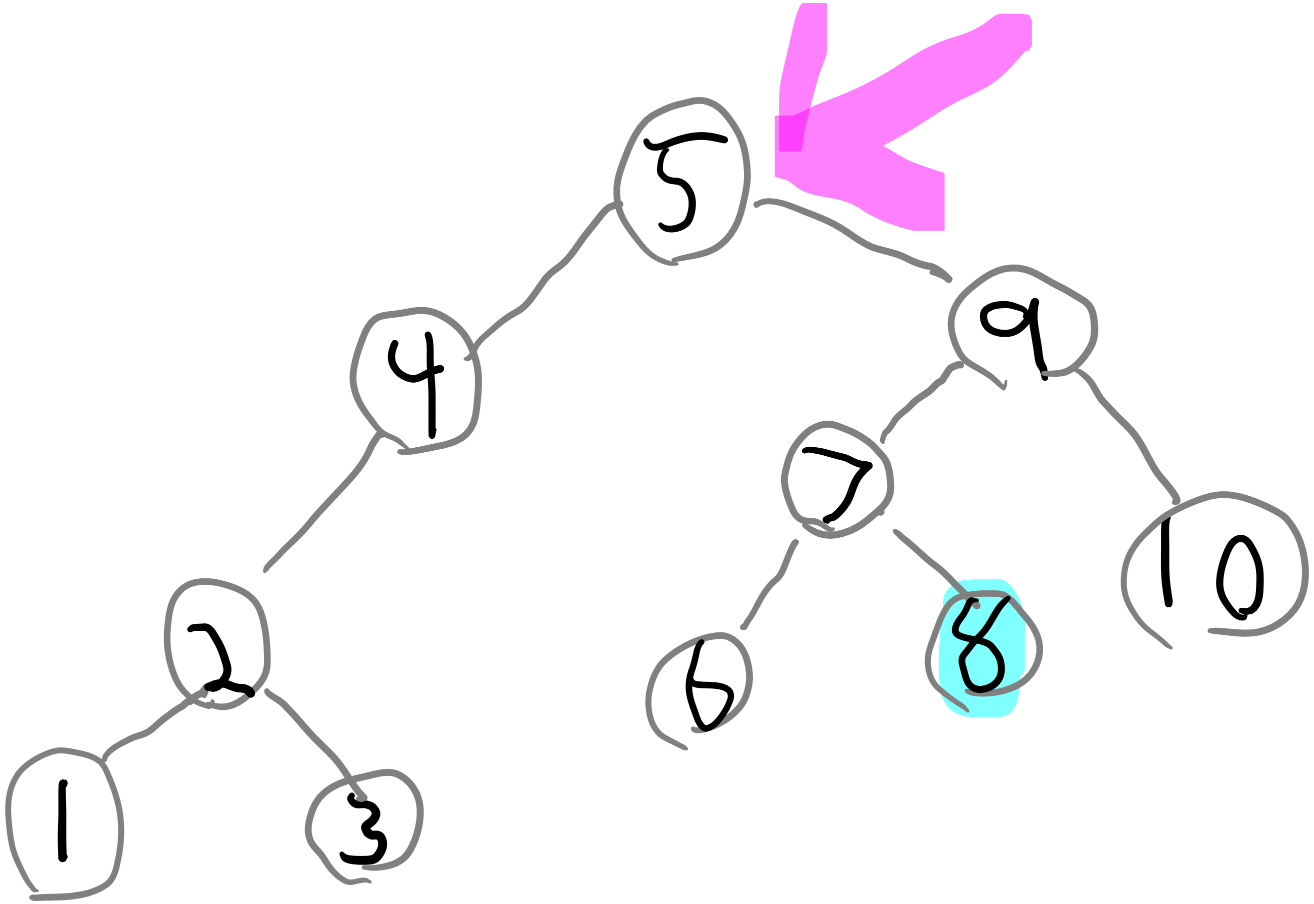
Binary Search Tree Model



Executes Searches

EG search for 8

Binary Search Tree Model

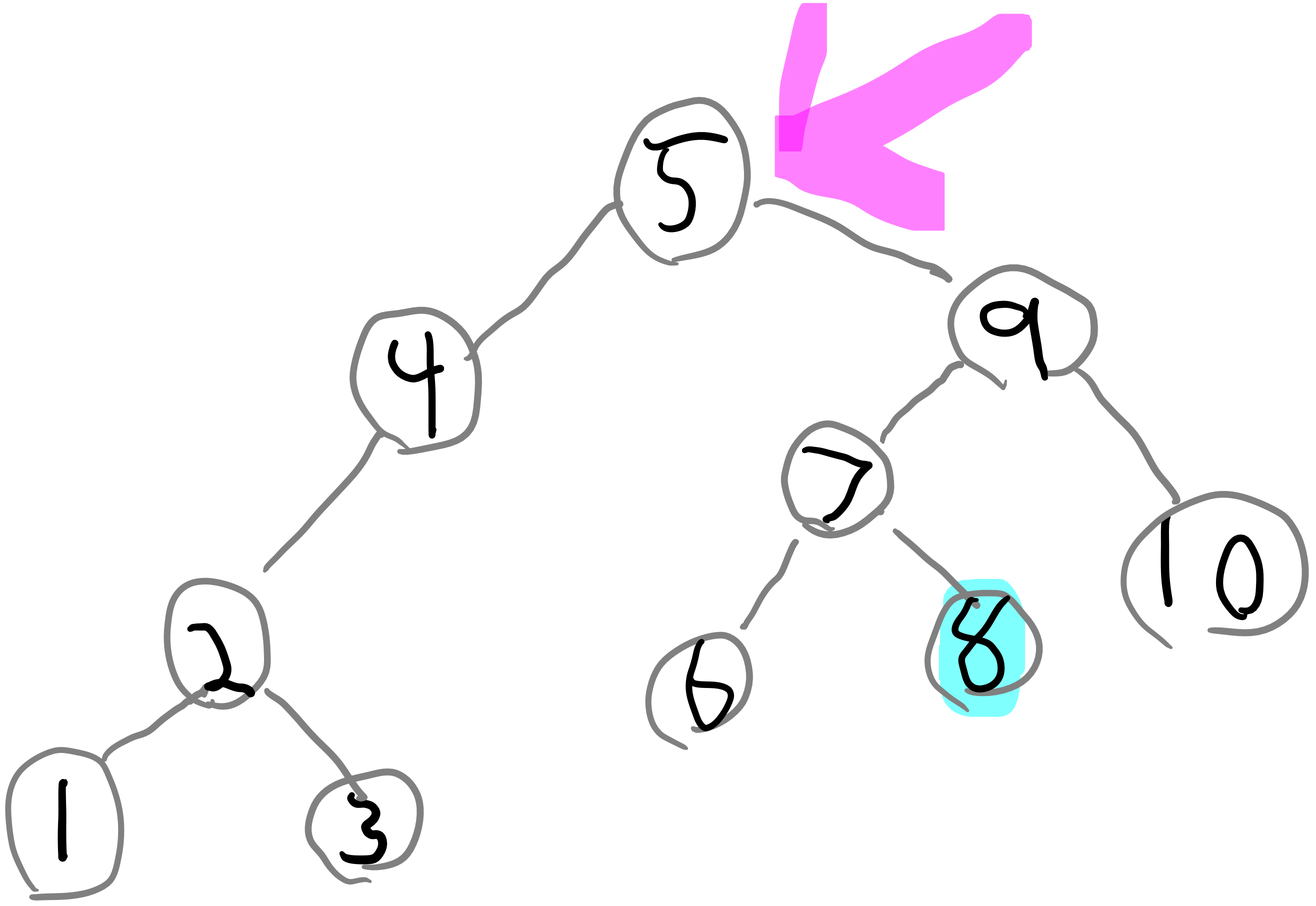


Executes Searches

EG search for 8

Single pointer, starts each search at root

Binary Search Tree Model



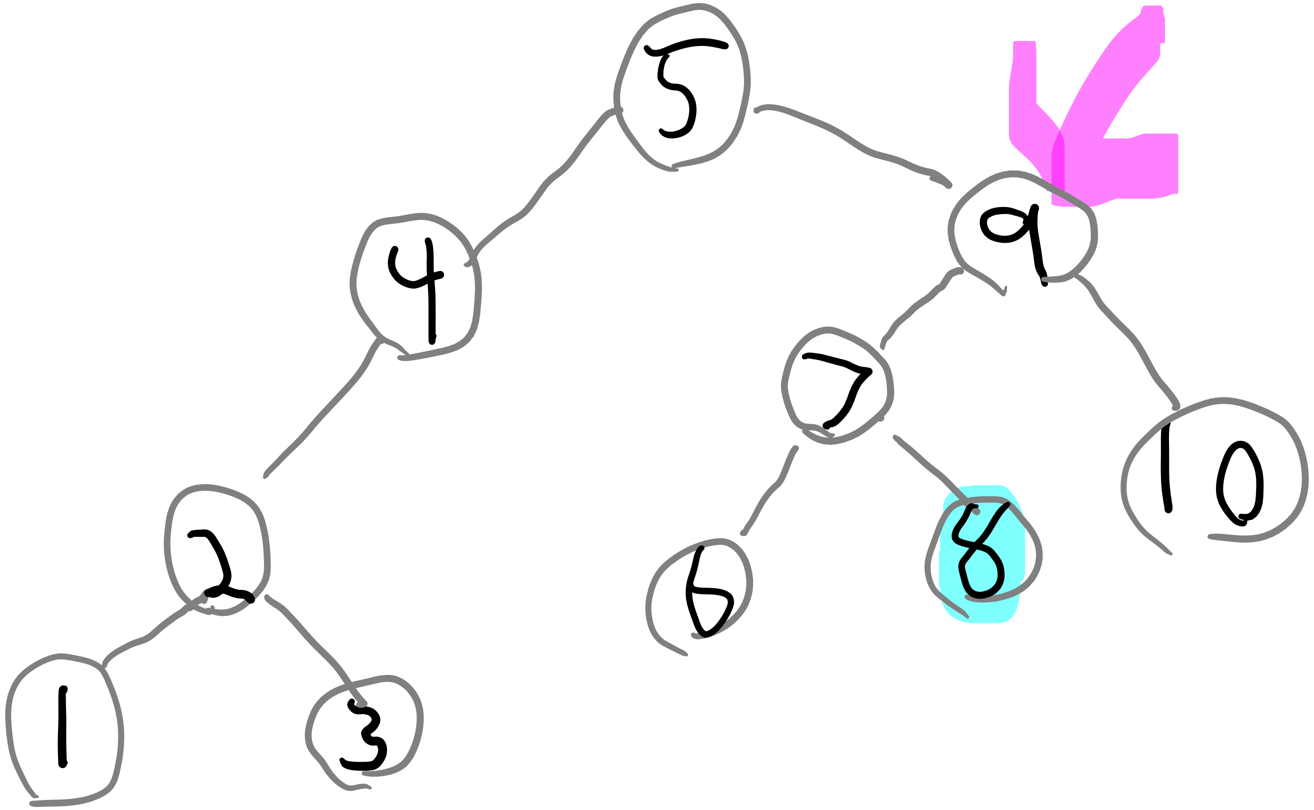
Executes Searches

EG search for 8

Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit Cost

Binary Search Tree Model



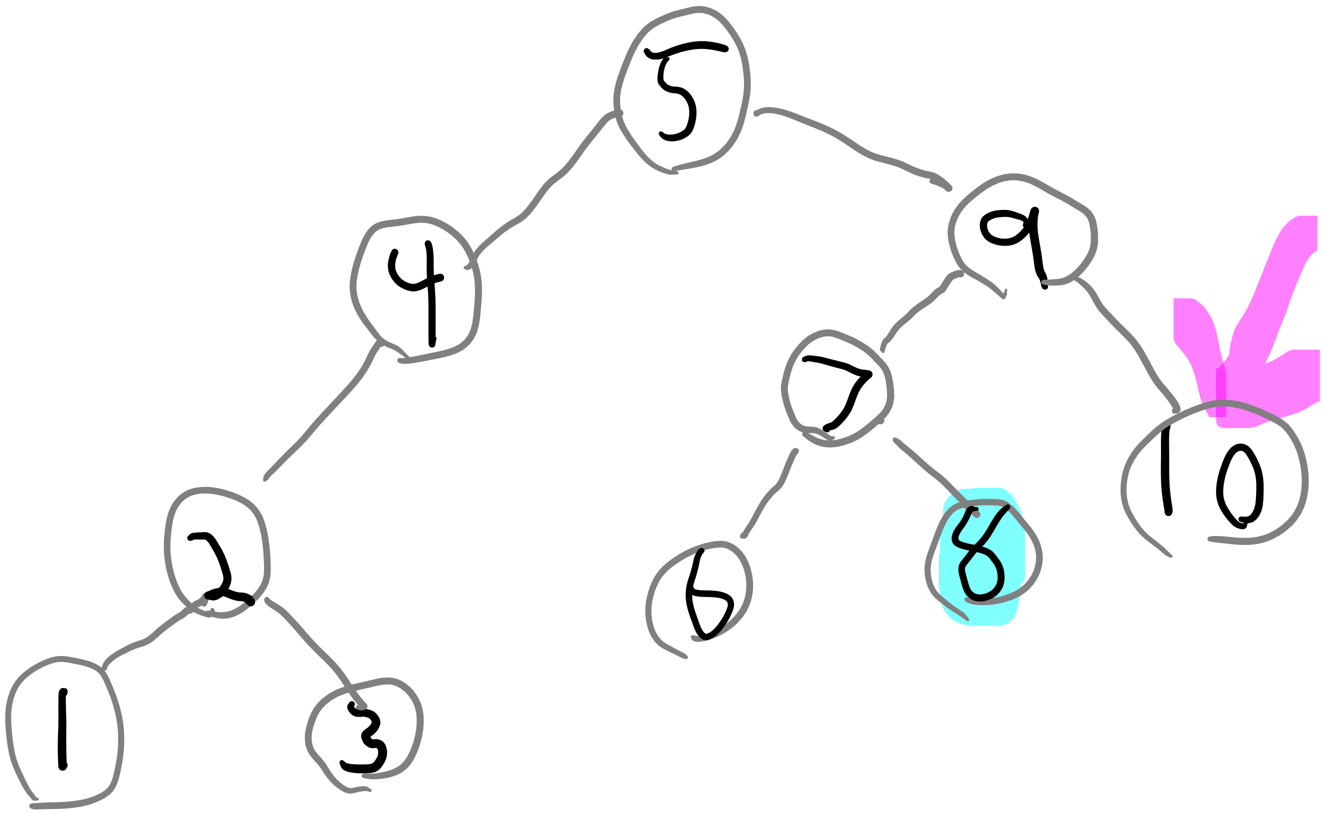
Executes Searches

EG search for 8

Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit Cost

Binary Search Tree Model



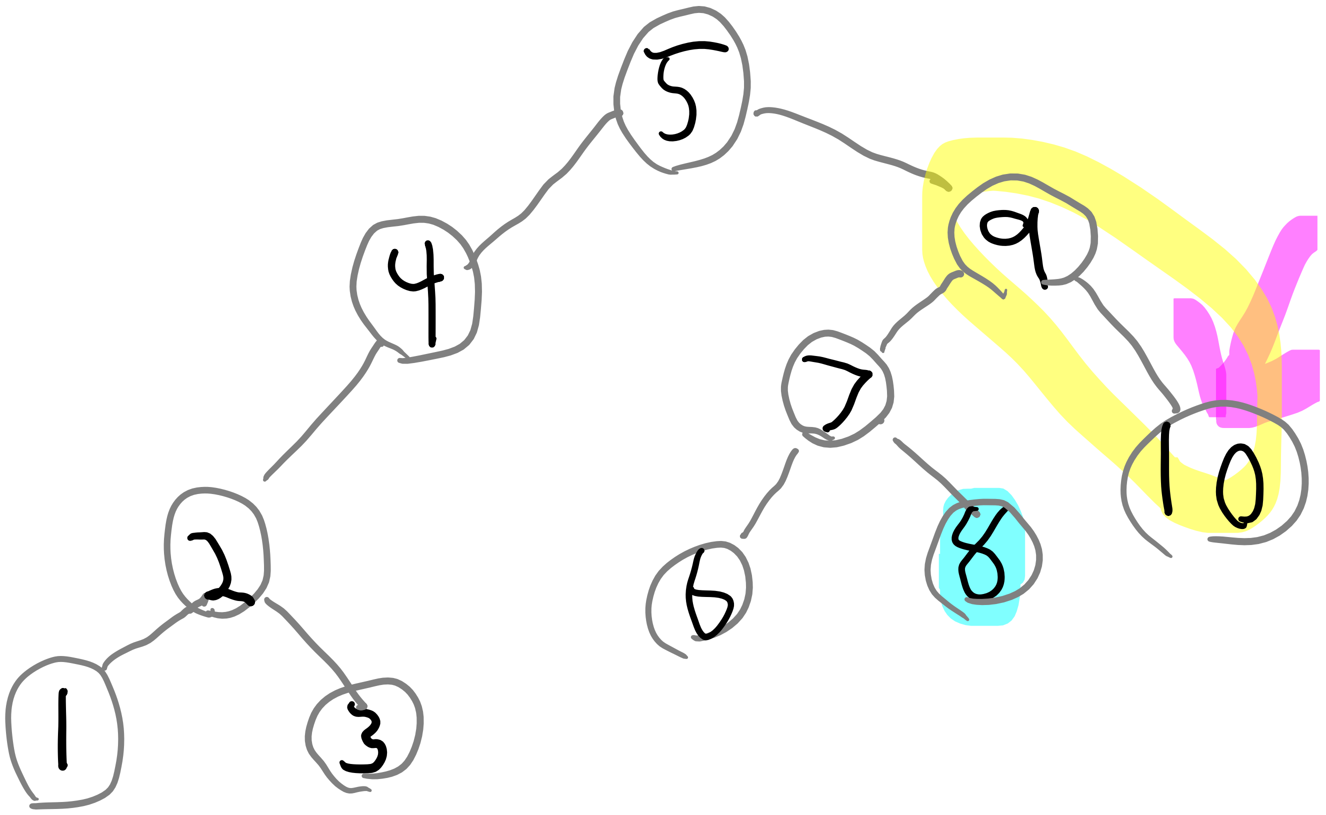
Executes Searches

EG search for 8

Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit Cost

Binary Search Tree Model



Executes Searches

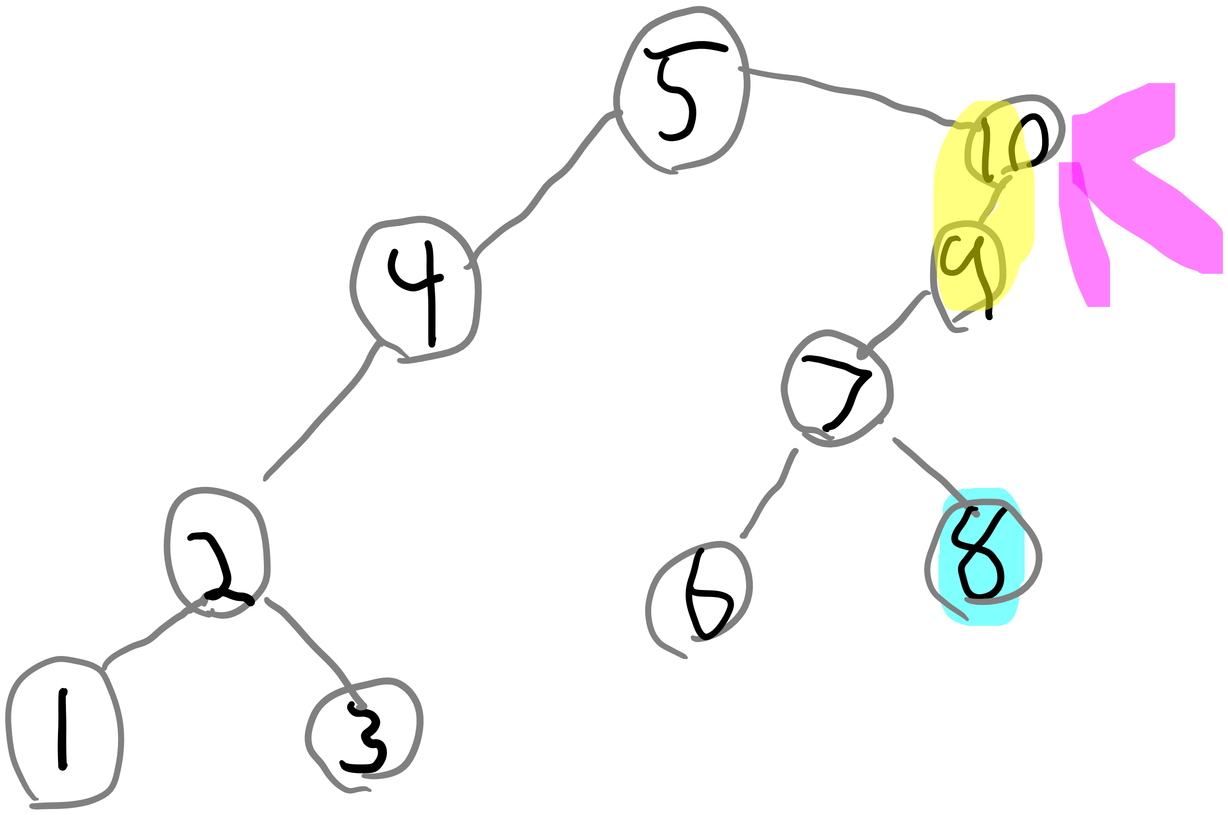
EG search for 8

Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit Cost

Can rotate with parent at unit cost

Binary Search Tree Model



Executes Searches

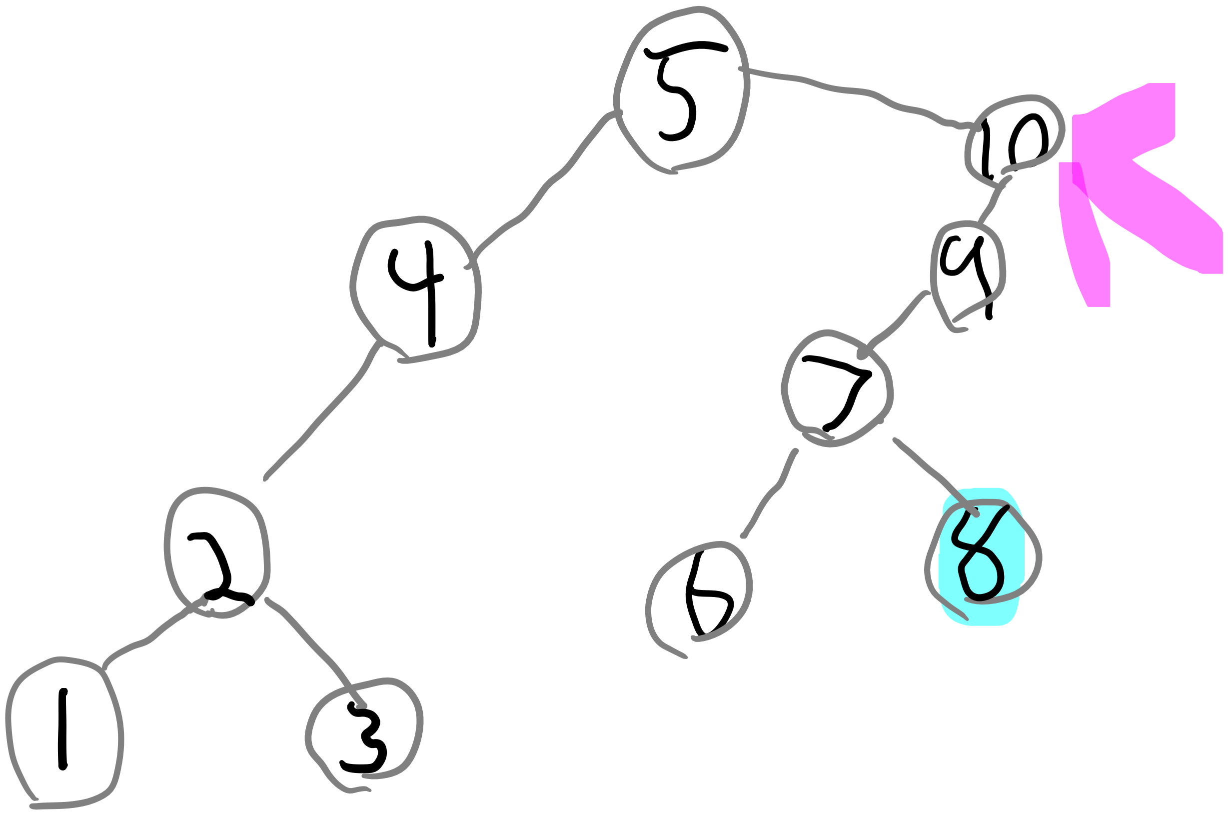
EG search for 8

Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit Cost

Can rotate with parent at unit cost

Binary Search Tree Model



Executes Searches

EG search for 8

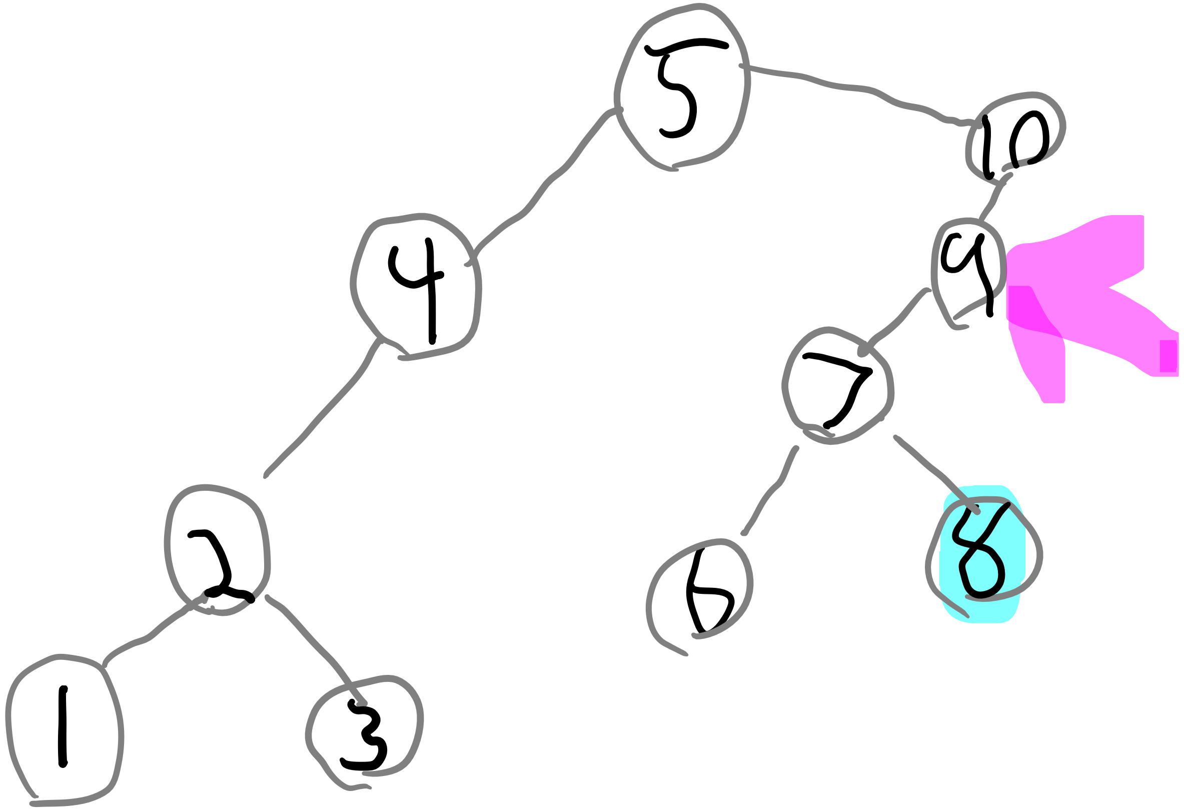
Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit cost

Can rotate with parent at unit cost

Must go to the searched item

Binary Search Tree Model



Executes Searches

EG search for 8

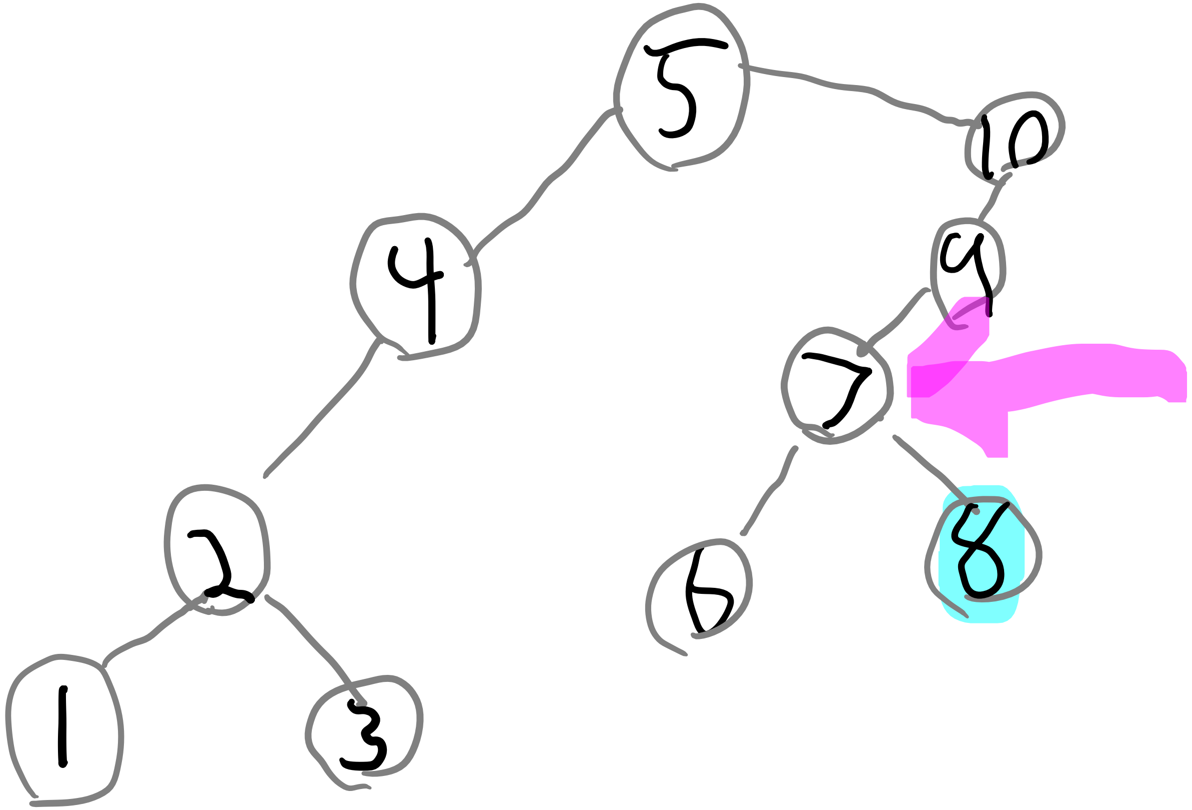
Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit cost

Can rotate with parent at unit cost

Must go to the searched item

Binary Search Tree Model



Executes Searches

EG search for 8

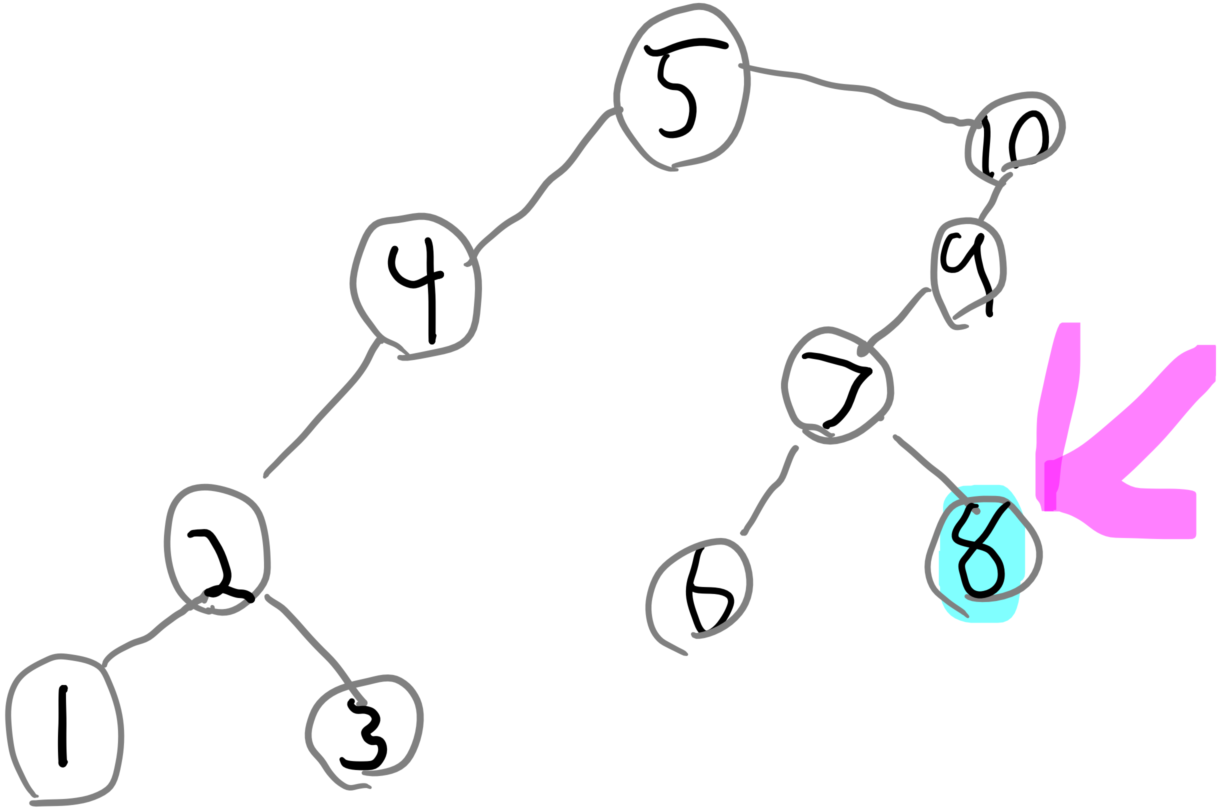
Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit cost

Can rotate with parent at unit cost

Must go to the searched item

Binary Search Tree Model



Executes Searches

EG search for 8

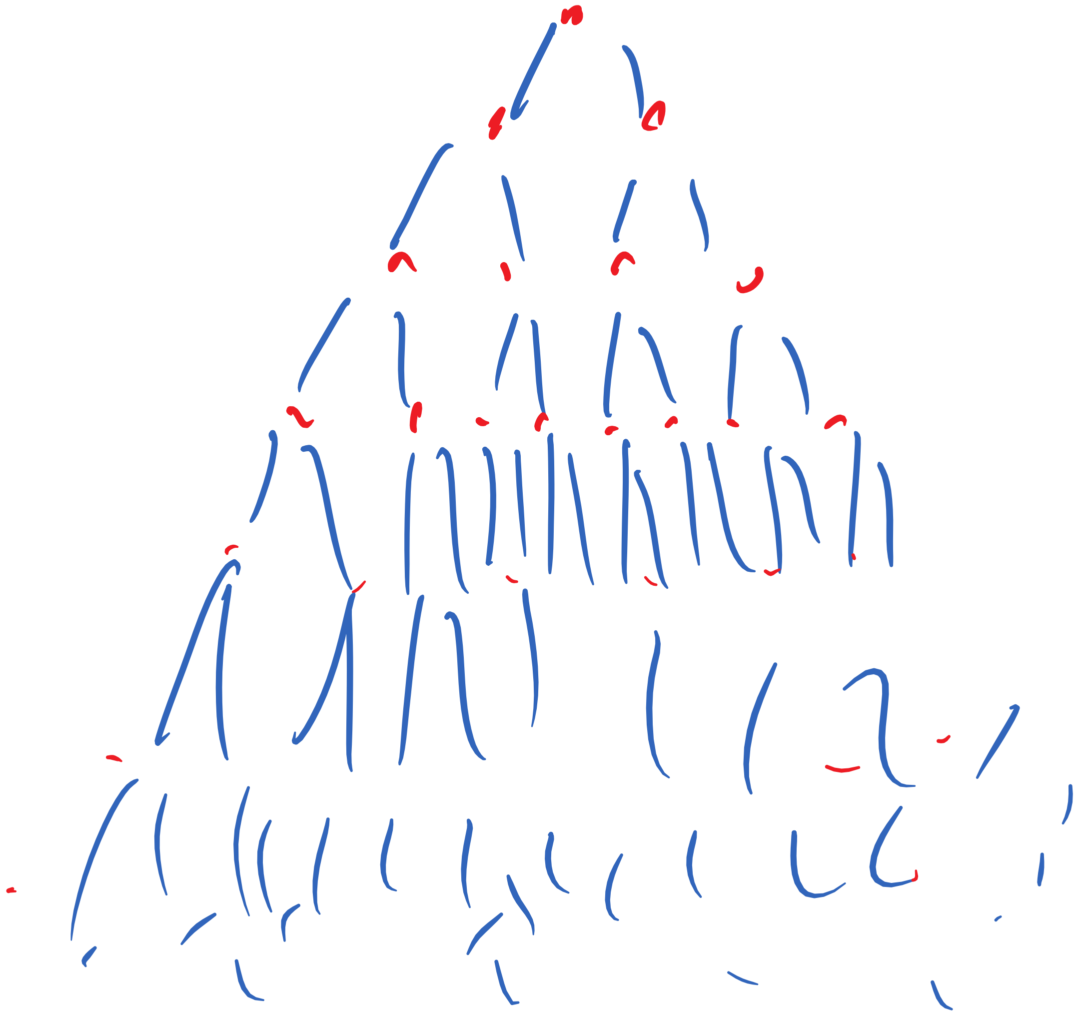
Single pointer, starts each search at root

Can move pointer Left, Right, Up at Unit cost

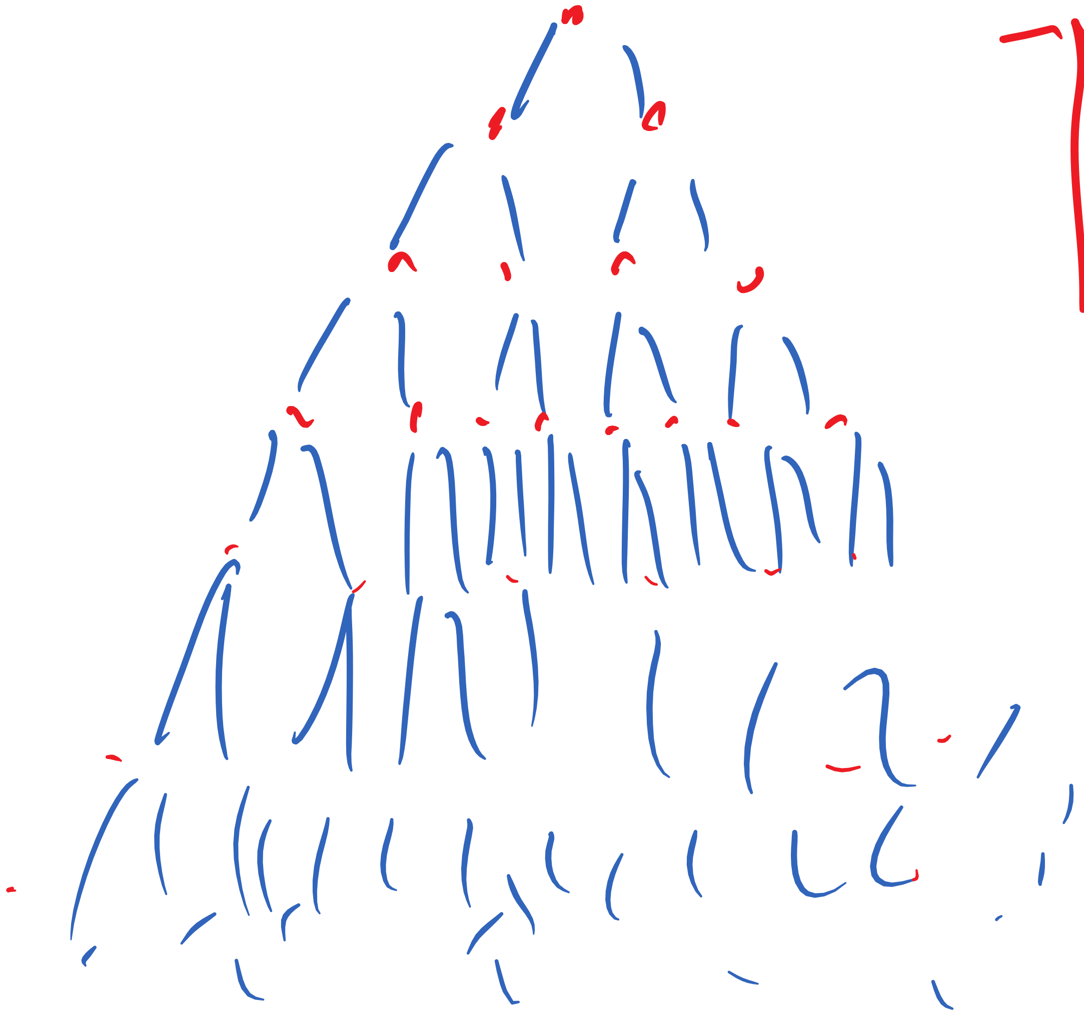
Can rotate with parent at unit cost

Must go to the searched item

How would you arrange your stuff



How would you arrange your stuff

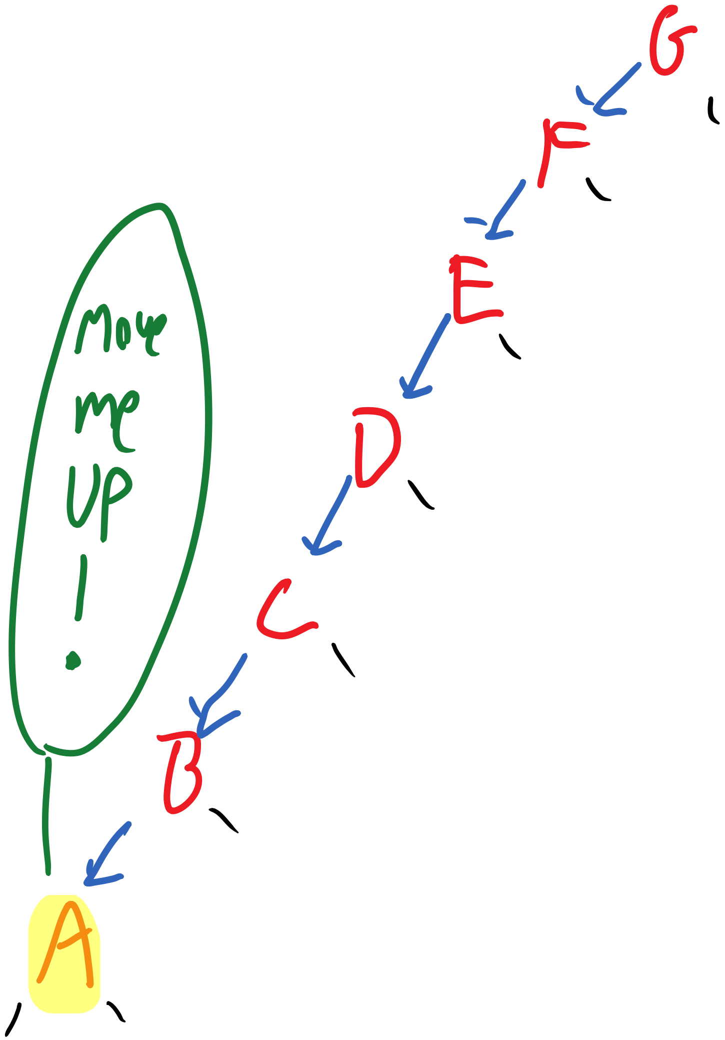


↑ stuff you acess frequently

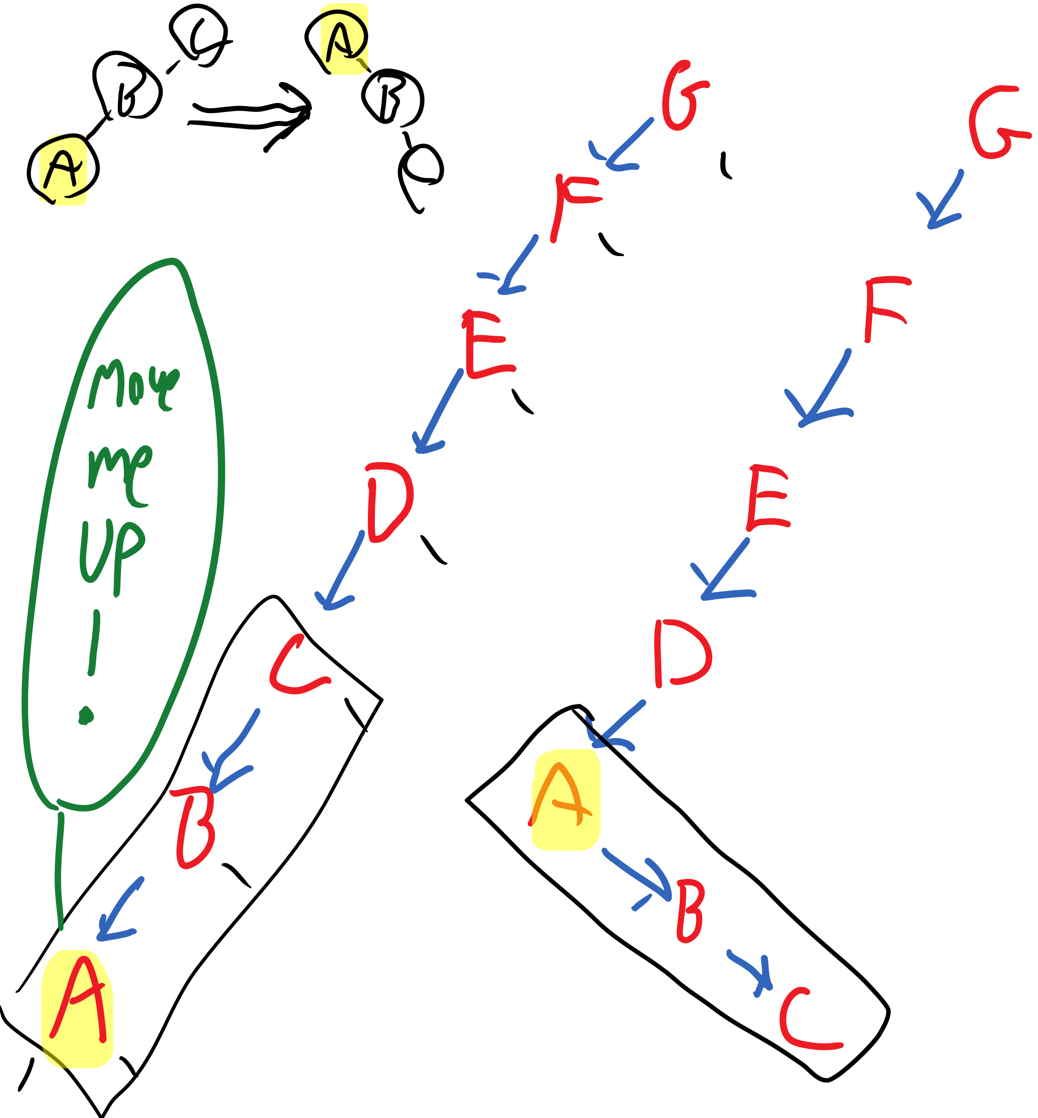
↓ stuff you never acess

Idea: When you find something
move it to the top

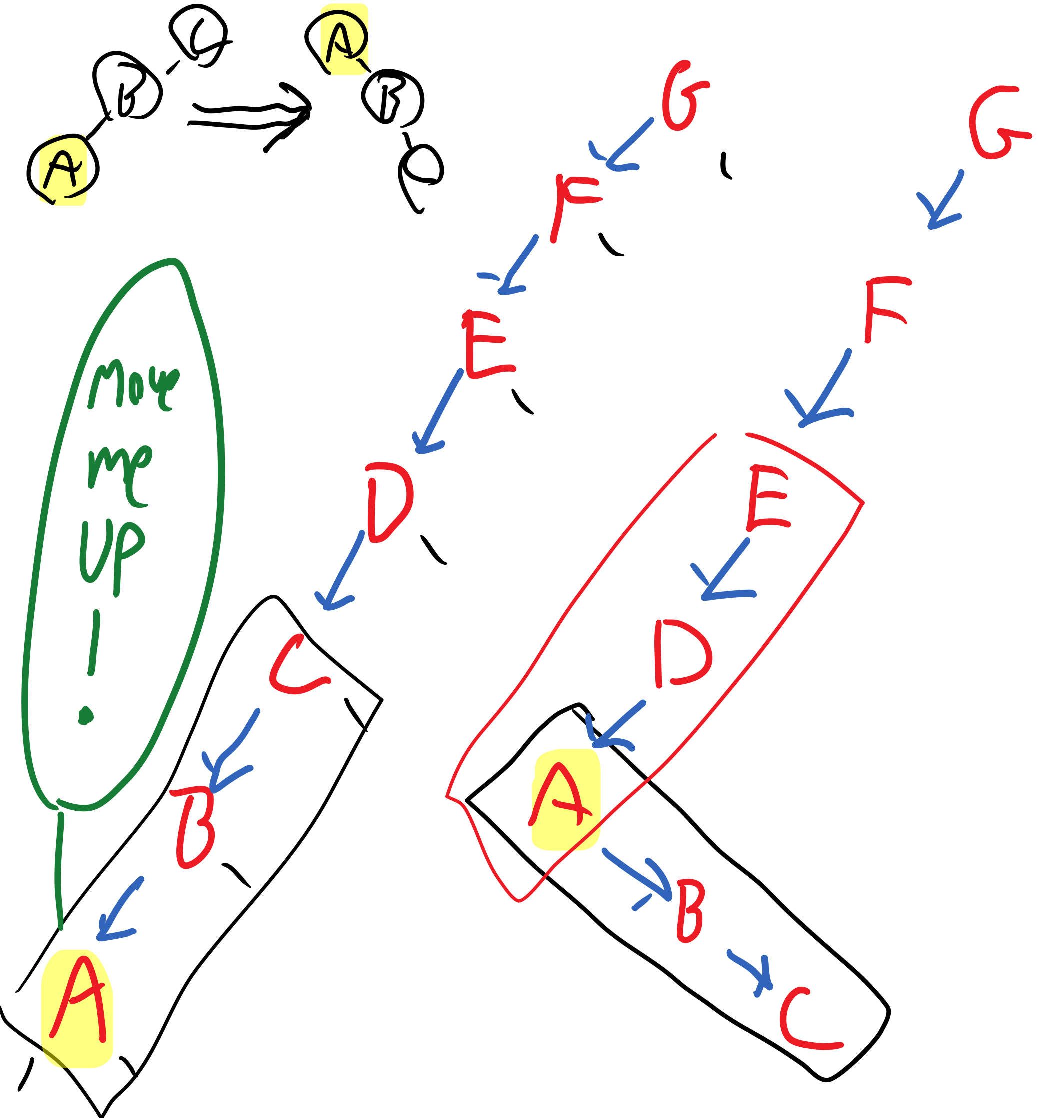
Idea: When you find something move it to the top



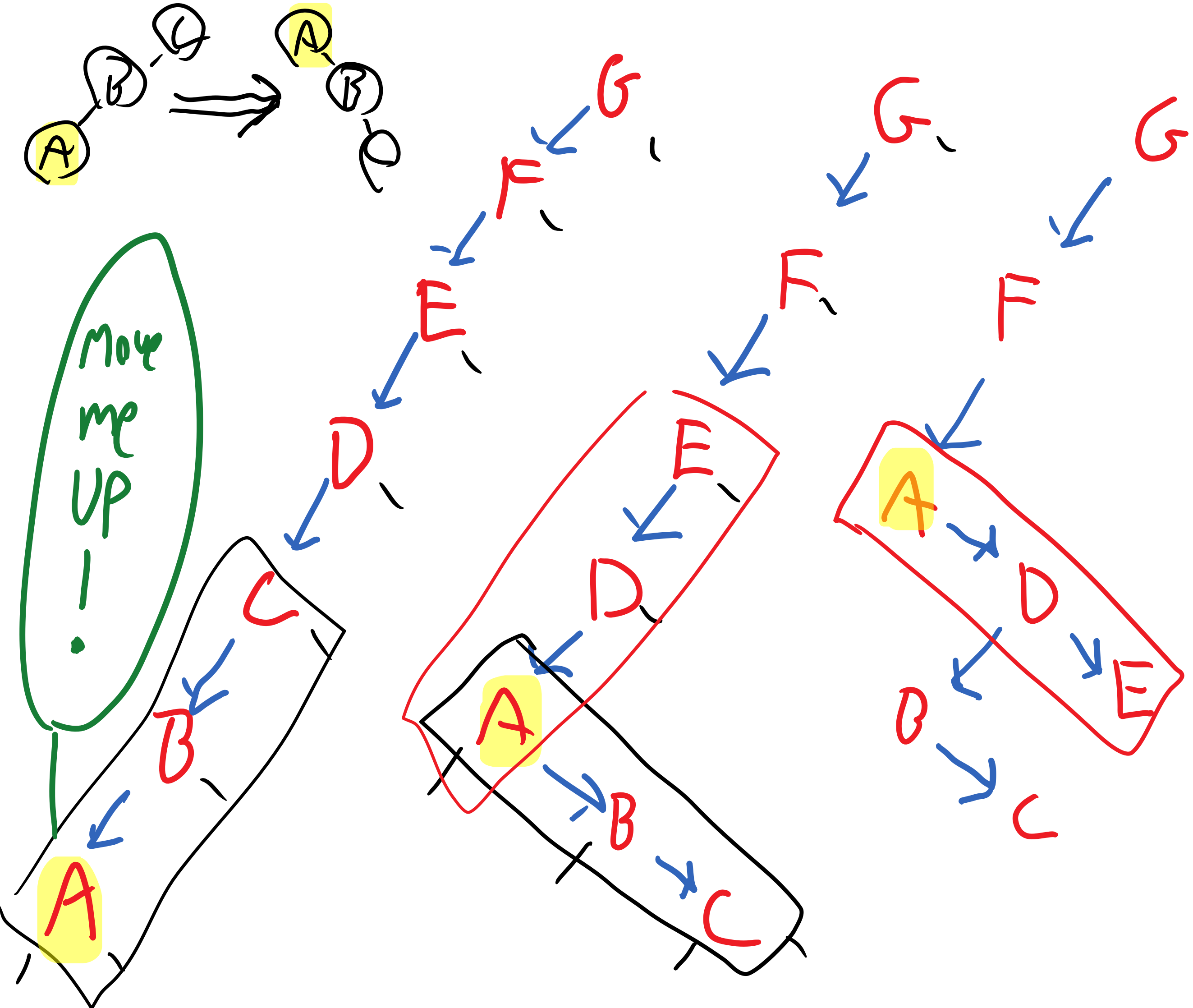
Idea: When you find something move it to the top



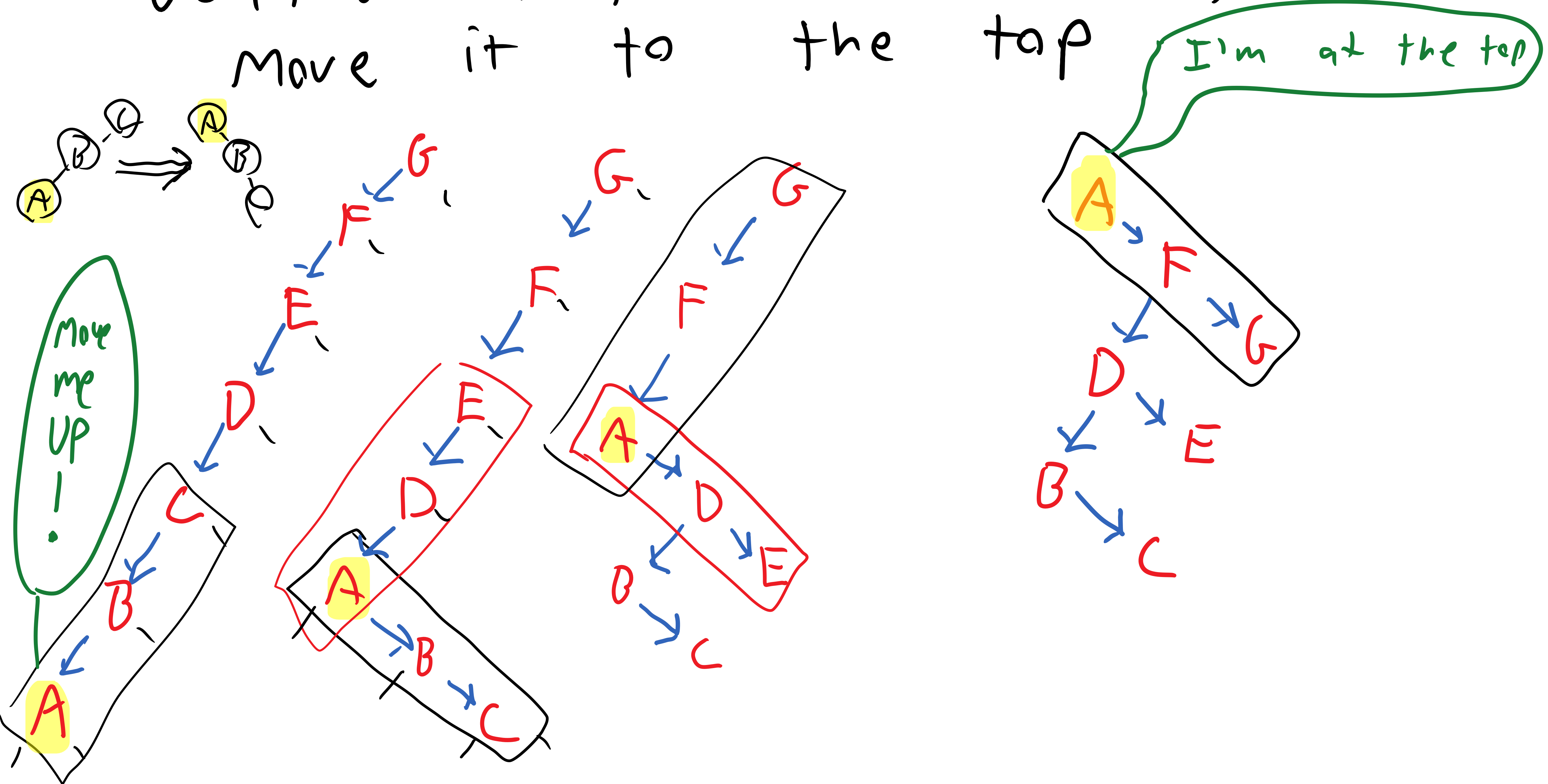
Idea: When you find something move it to the top



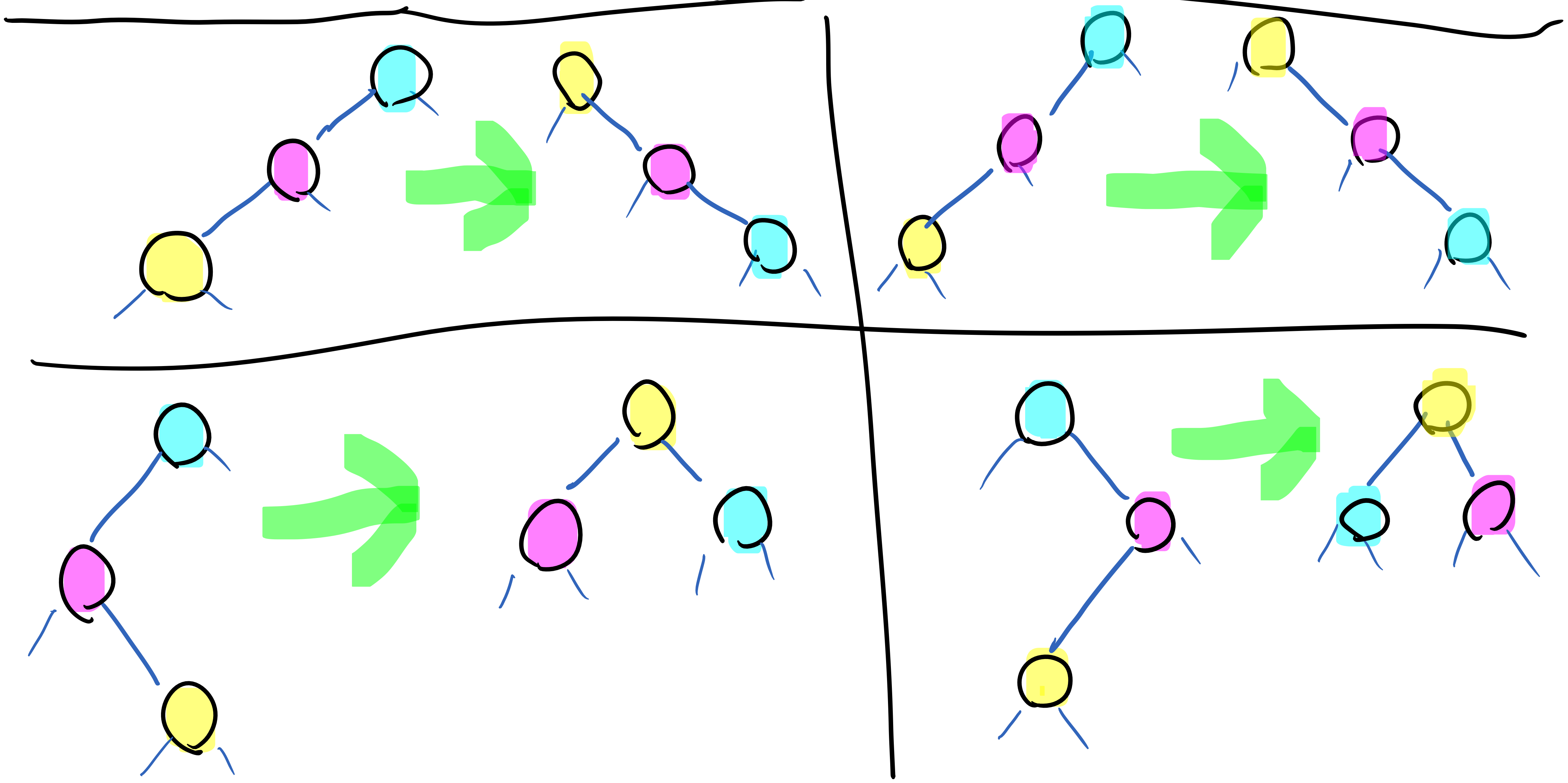
Idea: When you find something move it to the top



Idea: When you find something move it to the top



Details



- Moving what you are looking for to the top by moving up two at a time works pretty well.

- Name of this idea:

  <u>Splay trees</u>

  Sleator + Tarjan 84

What if you knew the future?

What if you knew the future?

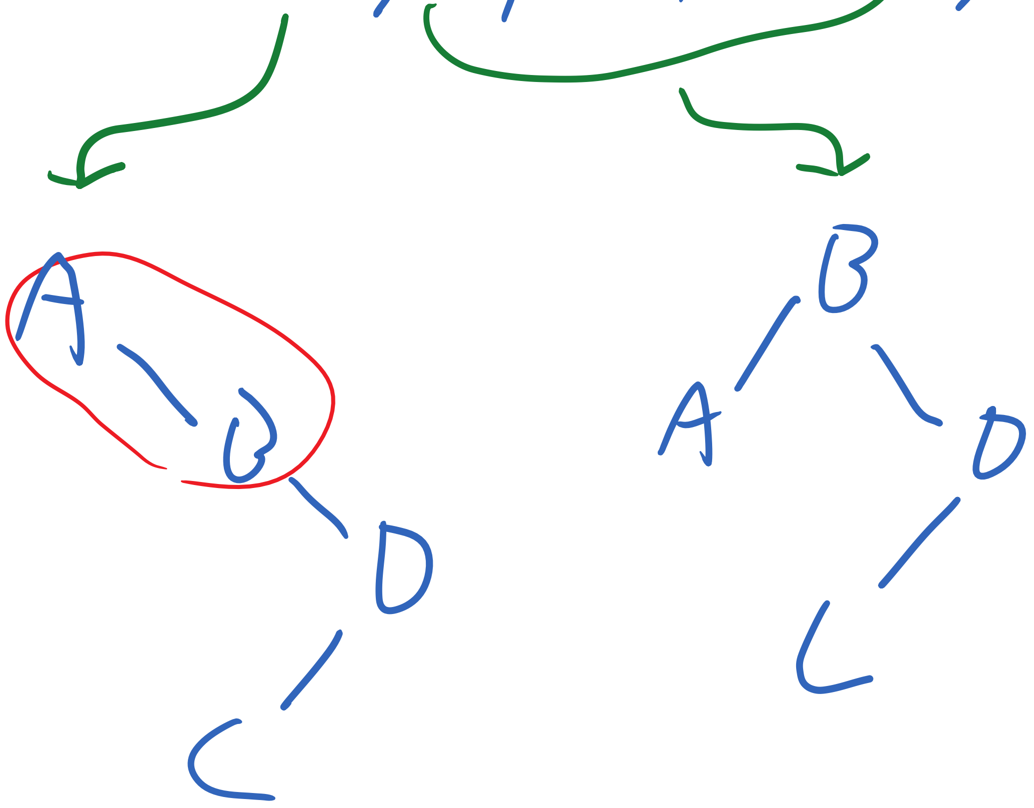
Searches: A, B, D, B, D, C, C, C, C, C

What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C

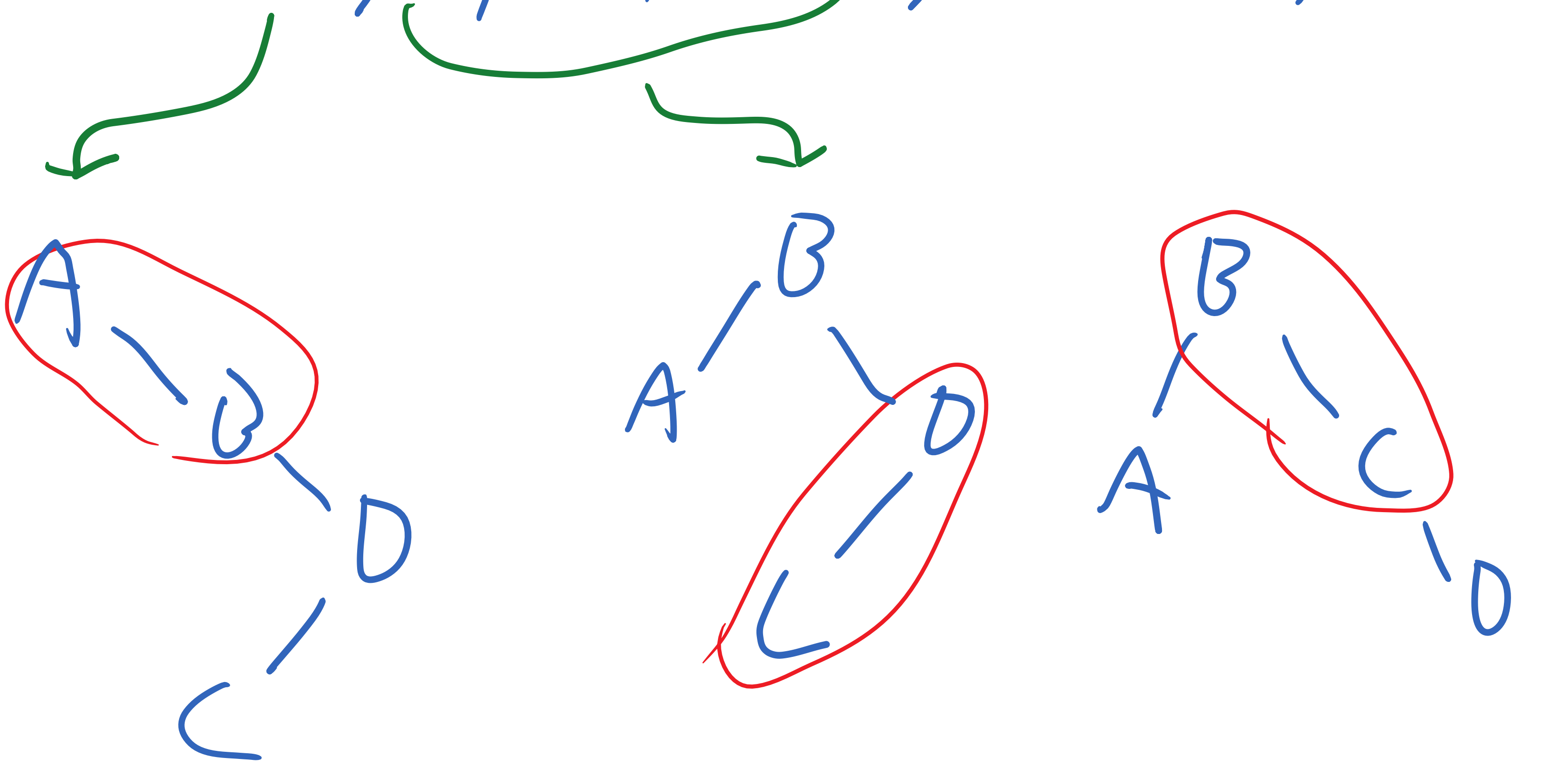
What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C



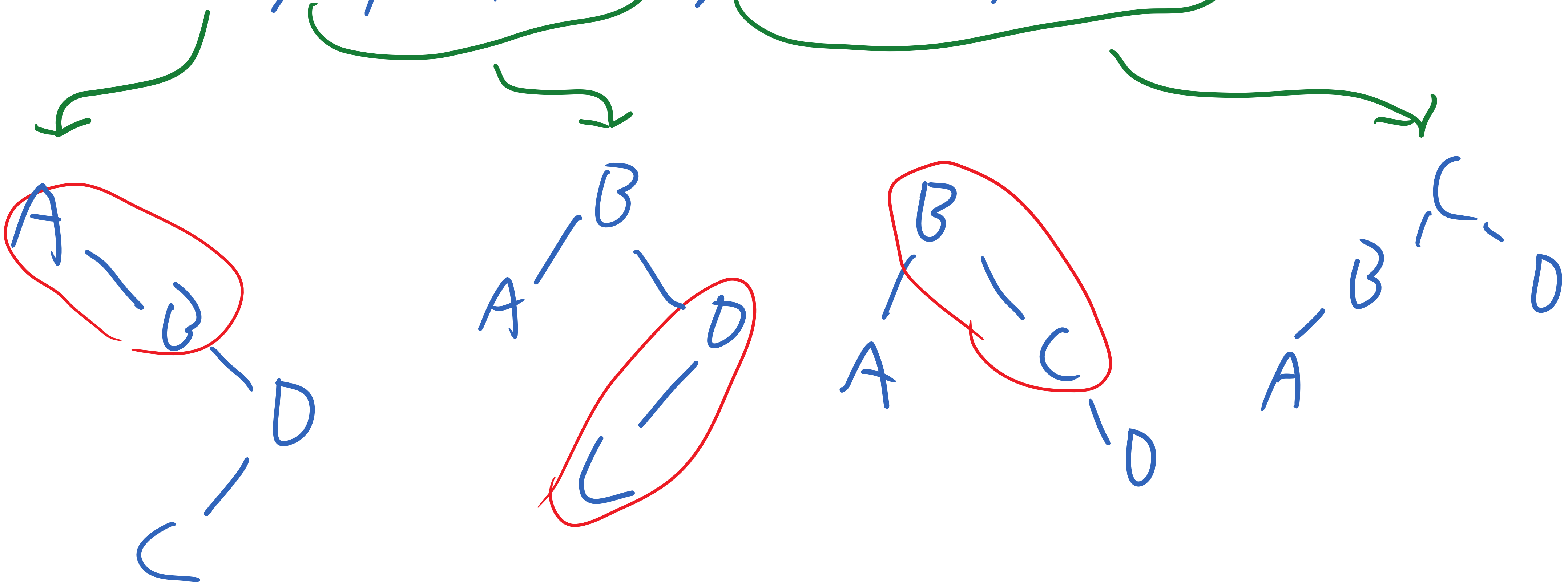
What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C



What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C



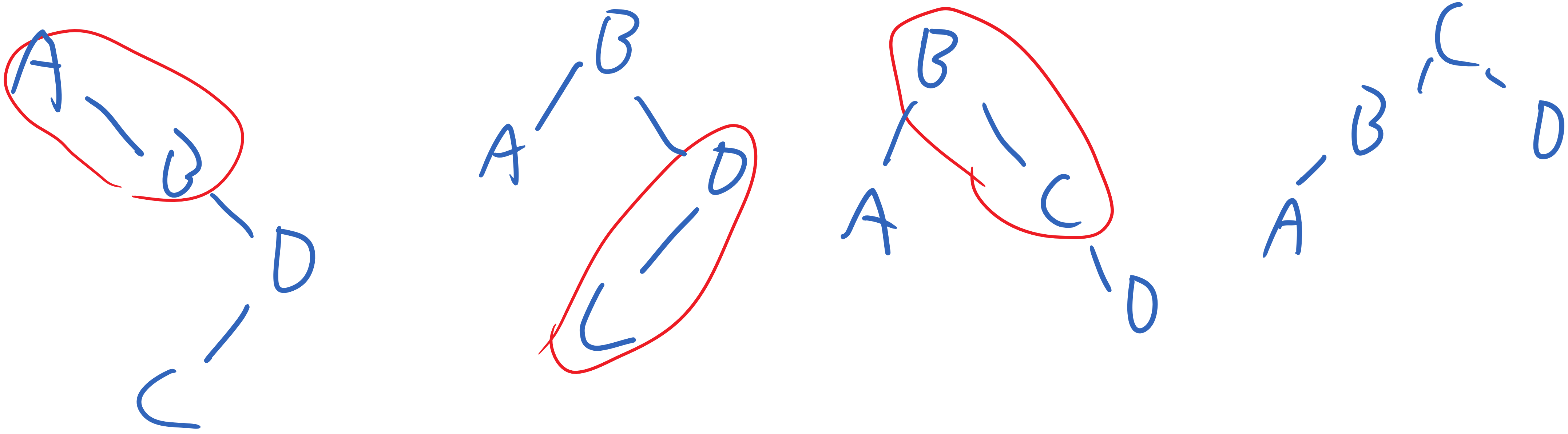
What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C

1 1 2 1 2 1 1 1 1 1 = 12 search

+ 3 changes

15 COST



Dynamic Optimality Conjecture

Cost to run splay trees
on a sequence

is at most
something like double the
smallest cost possible
for that sequence even knowing
the future

Notions of Optimality

N = # of items in tree

$X = x_1, x_2, x_3 \ldots x_m$ Searches

$R_A(X)$ = Time to execute X using Alg A

Notions of Optimality

N = # of items in tree

$X = x_1, x_2, x_3 \ldots x_m$ Searches

$R_A(X)$ = Time to execute X using Alg A

Amortized worst-case

$$WC(N) = \min_{A} \lim_{m \to \infty} \frac{\max_{X, |X| = m} R_A(X)}{m}$$

Notions of Optimality

N = # of items in tree

$X = x_1, x_2, x_3 \ldots x_m$ Searches

$R_A(X)$ = Time to execute X using Alg A

Amortized worst-case

$$WC(N) = \min_{A} \lim_{m \to \infty} \frac{\max_{X, |X|=m} R_A(X)}{m} = O(m \log n)$$

For BST: $WC(N) = \lceil \log_2(N+1) \rceil$

Notions of Optimality

N = # of items in tree

$X = x_1, x_2, x_3 \dots x_m$ Searches

$R_A(X)$ = Time to execute X using Alg A

Amortized worst-case $WC(N) = \min\limits_{A} \lim\limits_{m \to \infty} \dfrac{\max\limits_{X, |X|=m} R_A(X)}{m}$

Instance-Based Optimality $OPT(X) = \min\limits_{A} R_A(X)$

Online vs Offline

$X = x_1, x_2, x_3 \cdots x_m$ Sequence of operations

Alg A is Online: Executes x_i based on $x_1, x_2 \ldots x_i$ only.

Alg A is Offline: Executes x_i based on all of X.

Dynamic Optimality

$$OPT(X) = \min_{A} R_A(X)$$

Algorithm A is <u>Dynamically Optimal</u> if

$$\forall X \quad R_A(X) = O(OPT(X))$$

Dynamic Optimality

$$OPT(X) = \min_{A} R_A(X)$$

Different A for each X

Algorithm A is <u>Dynamically Optimal</u> if

One A for all X

$$\forall x \quad R_A(X) = O(OPT(X))$$

Dynamic Optimality

$$OPT(X) = \min_A R_A(X)$$

Different A for each X

Algorithm A is <u>Dynamically Optimal</u> if

One A for all X

$$\forall X \quad R_A(X) = O(OPT(X))$$

Interesting even if A is offline

... more interesting if A is Online ....

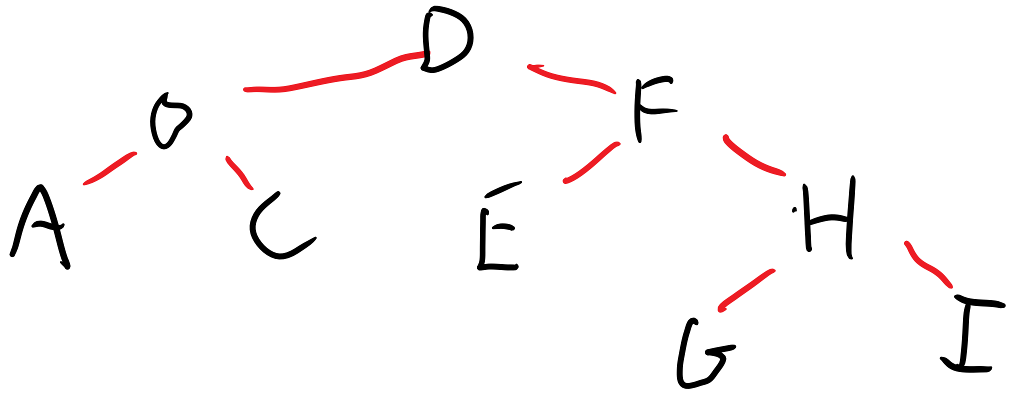
If this were true:

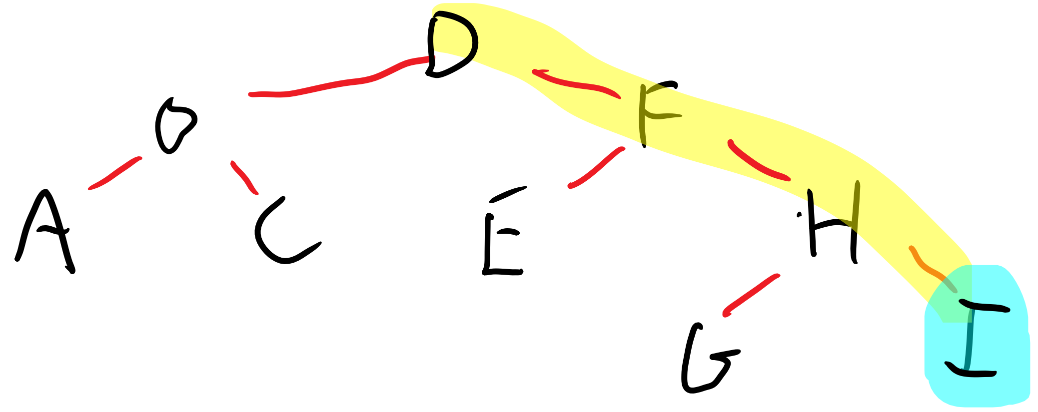
Knowing the future is useless,

If at first you don't understand...

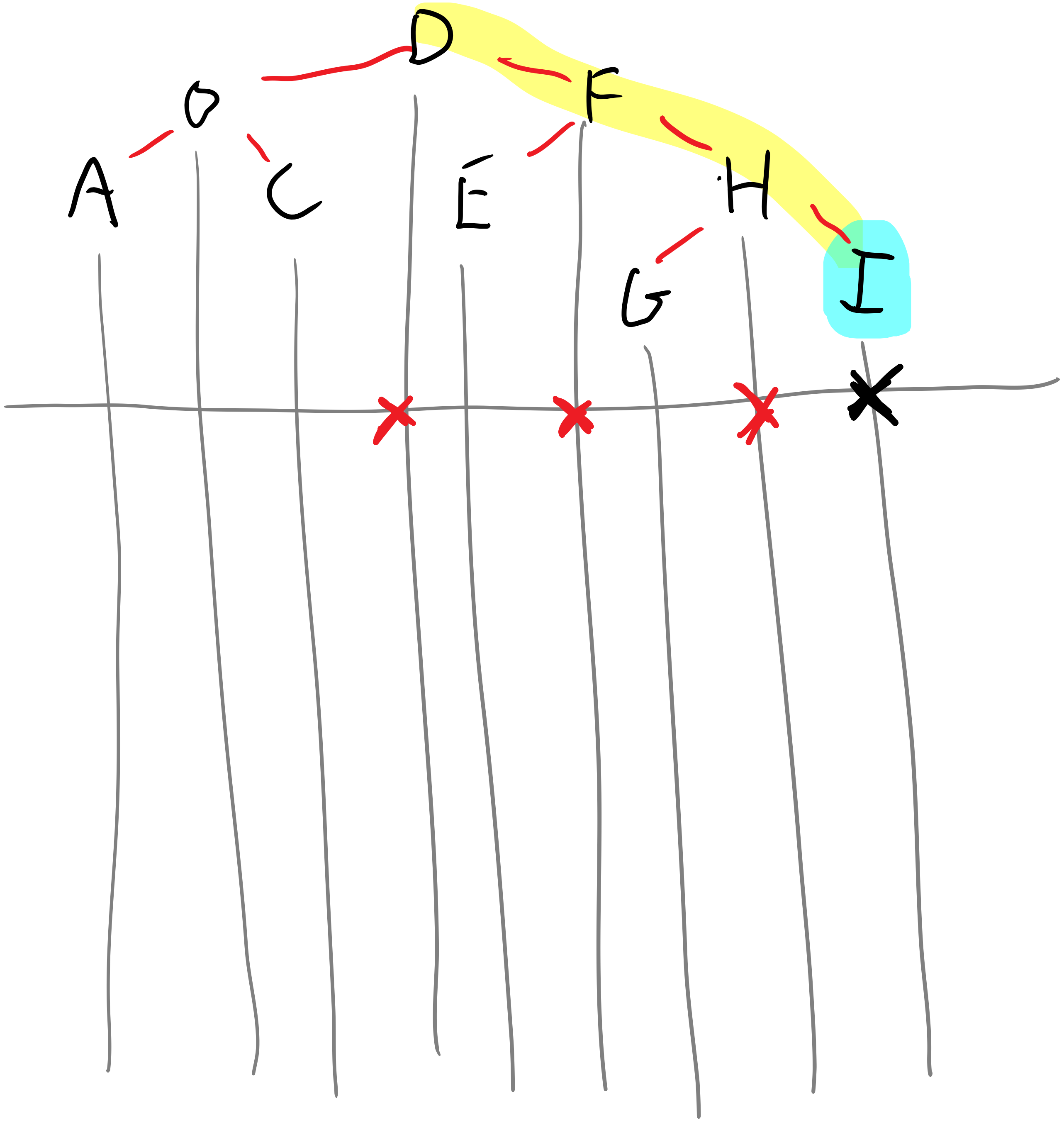
If at first you don't understand...

Draw a Different Picture

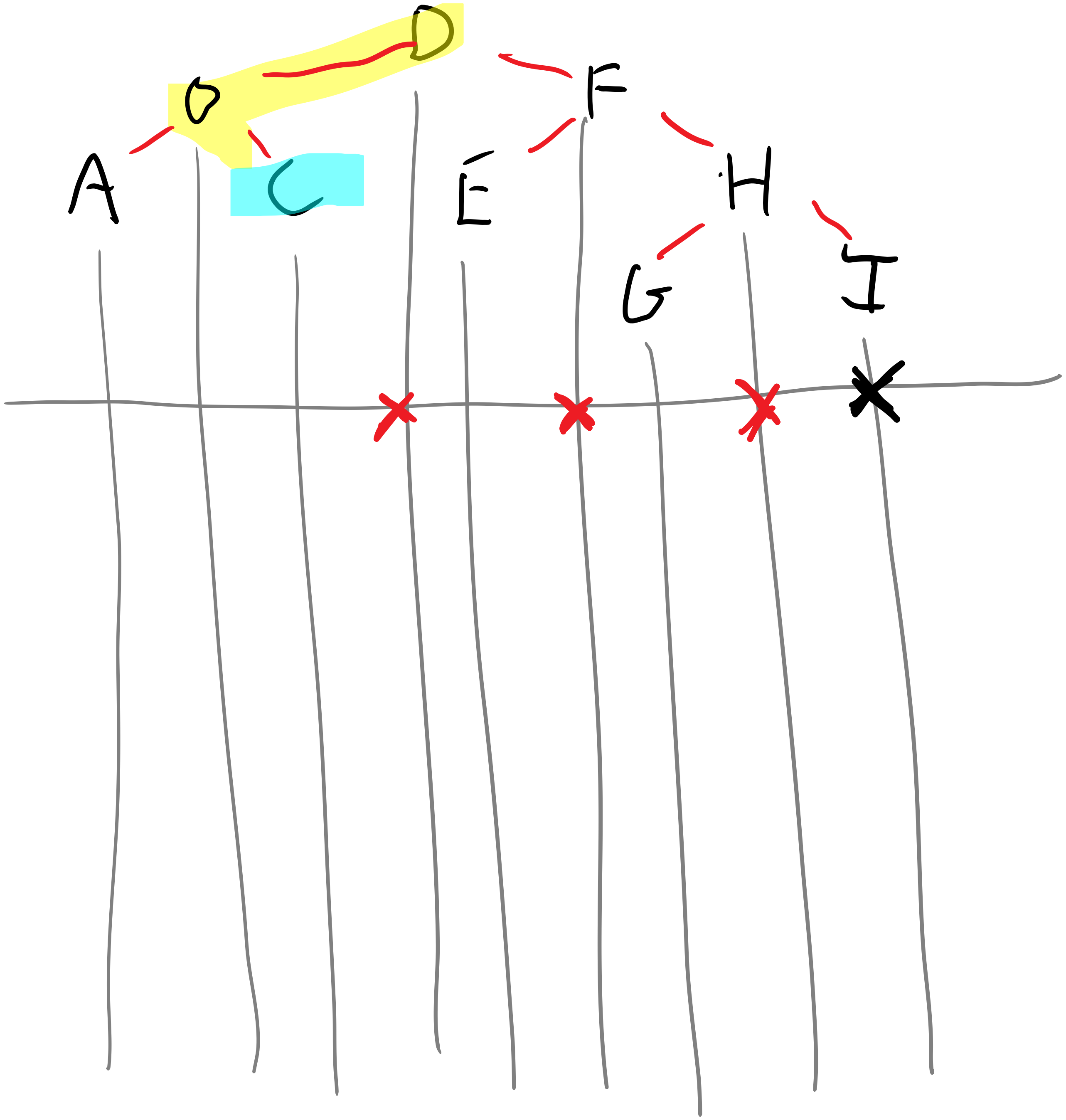

D
B
F
A
C
E
H
G
I



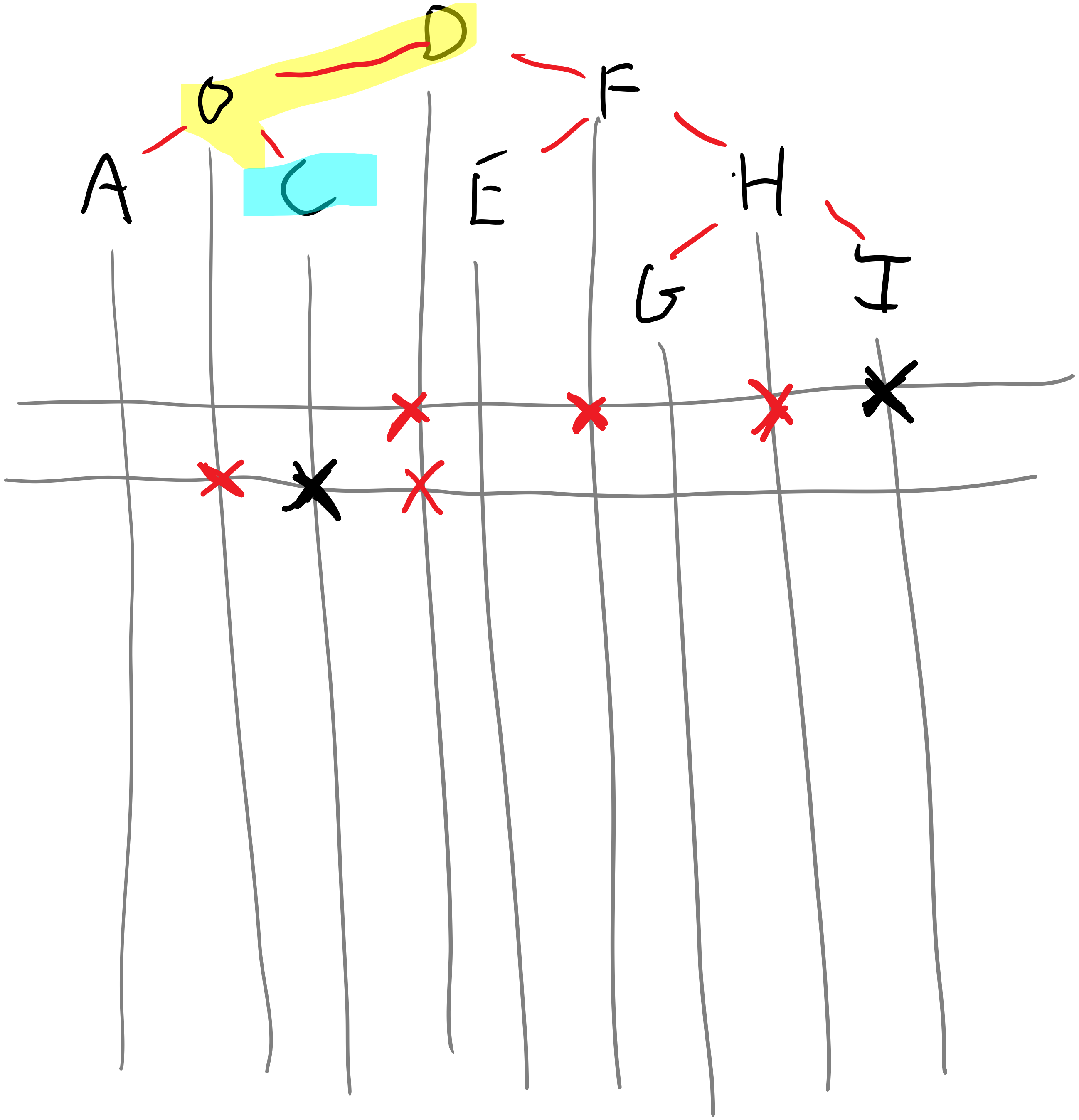
Search for I



Search for I

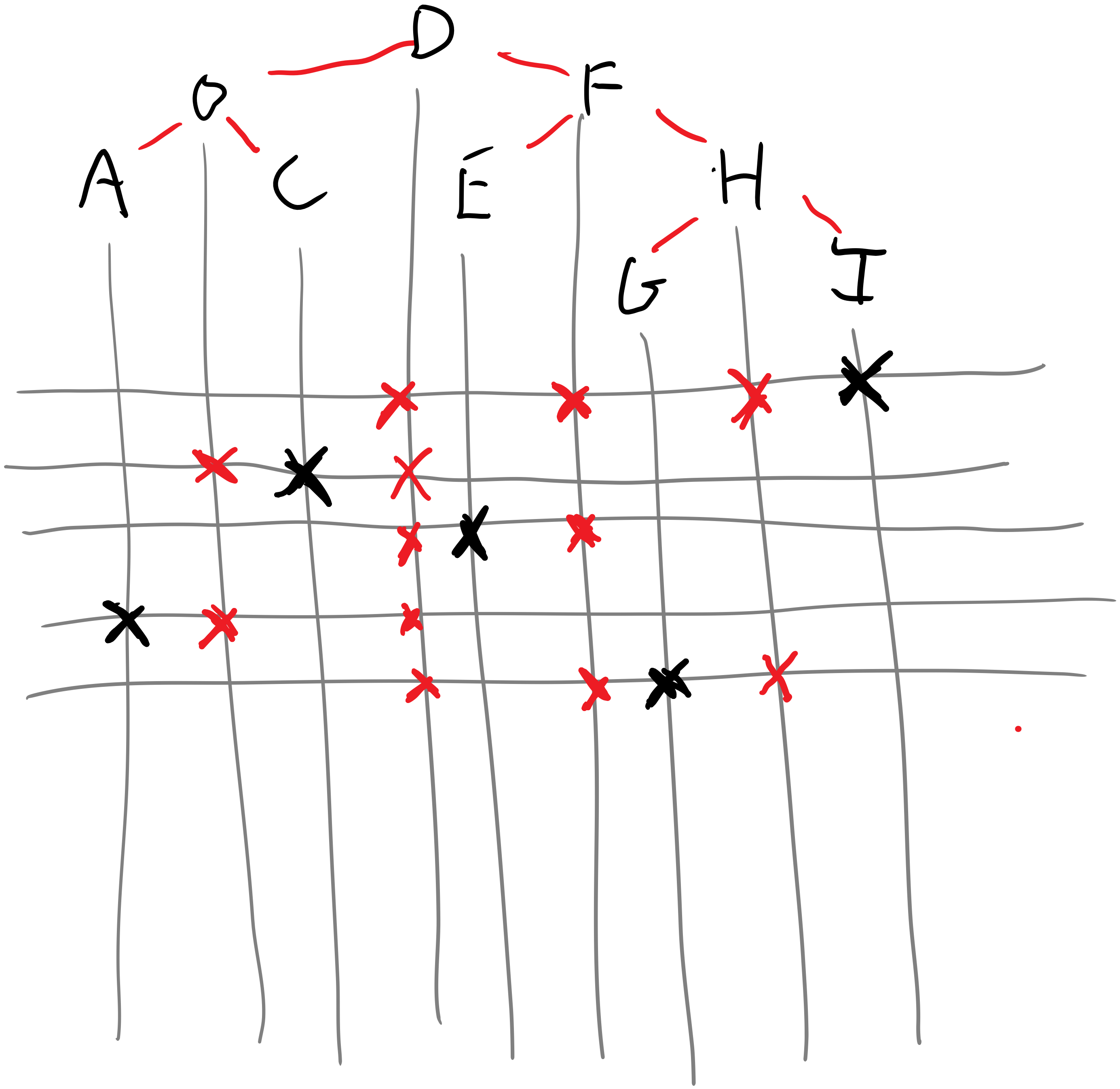


Search for
C



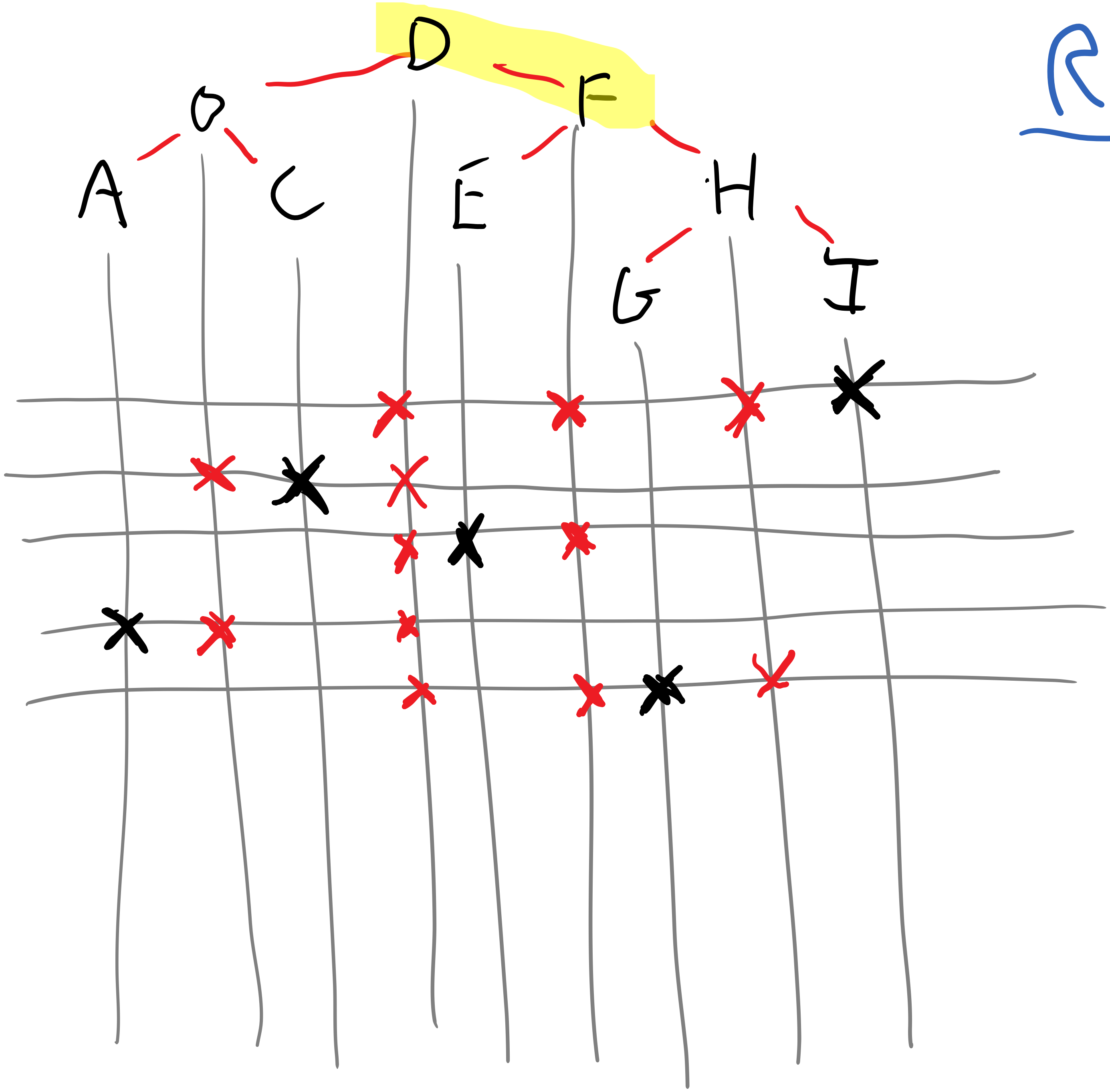
Search for

C

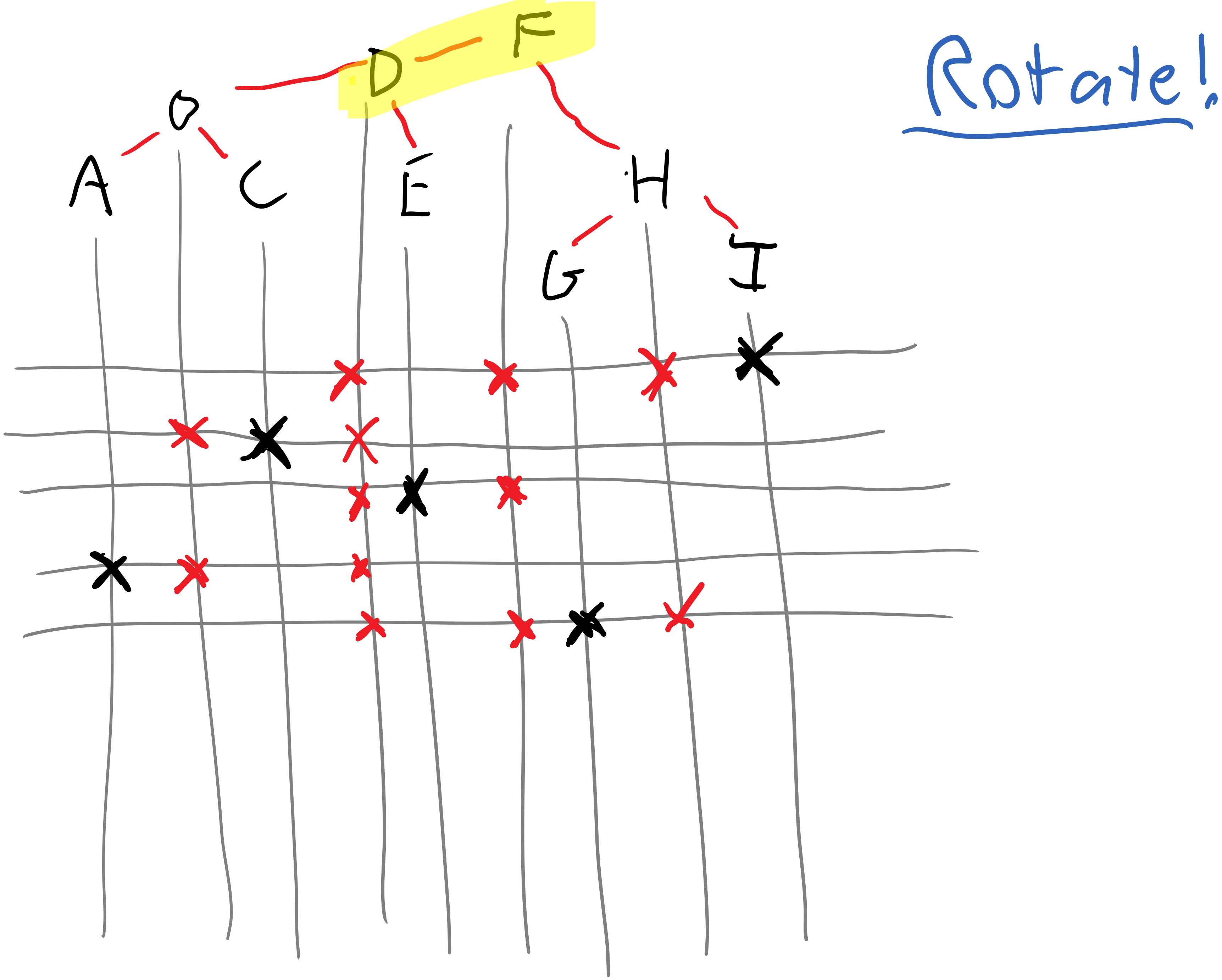


Search for

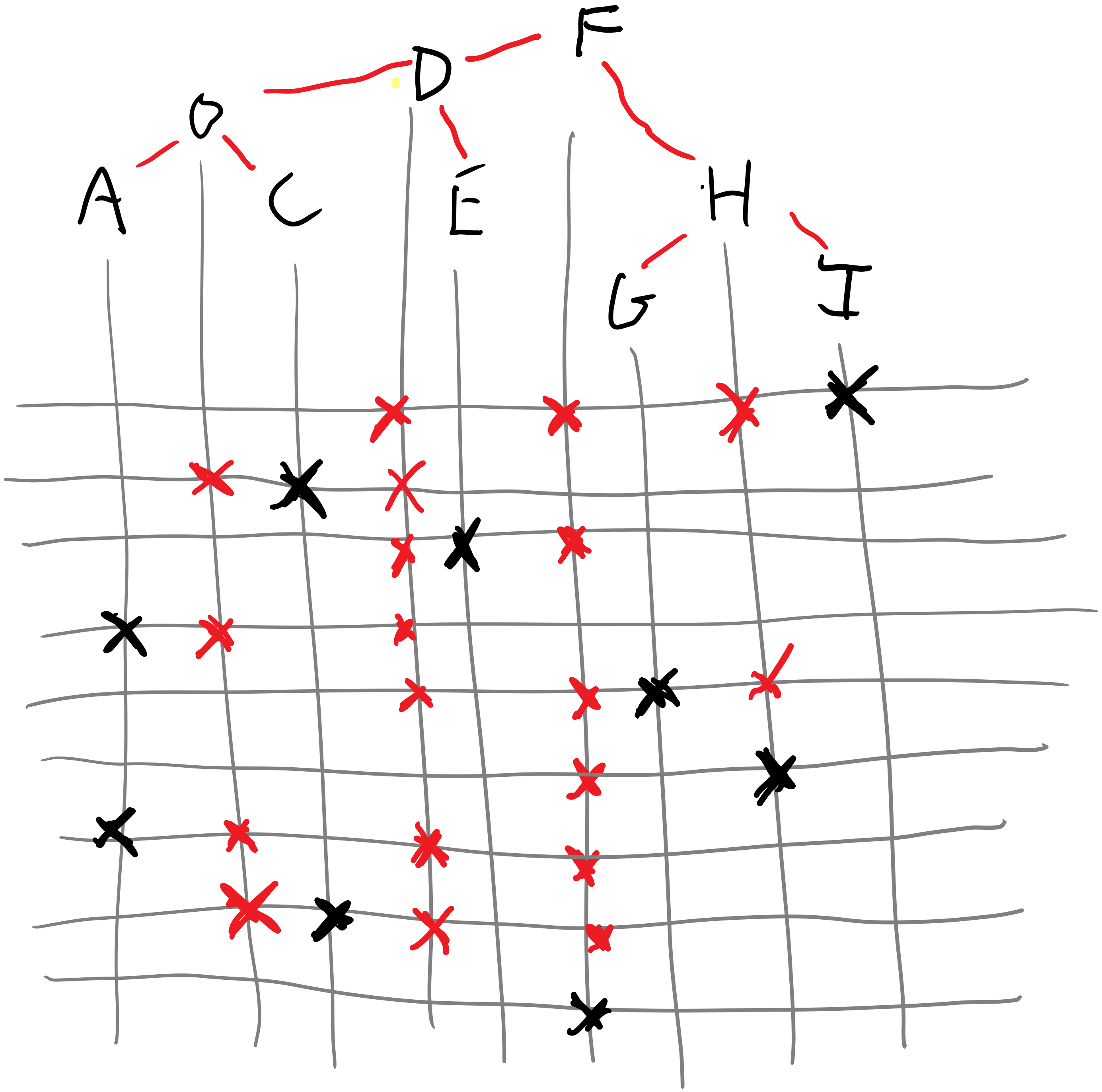
E
A
G



Rotate!

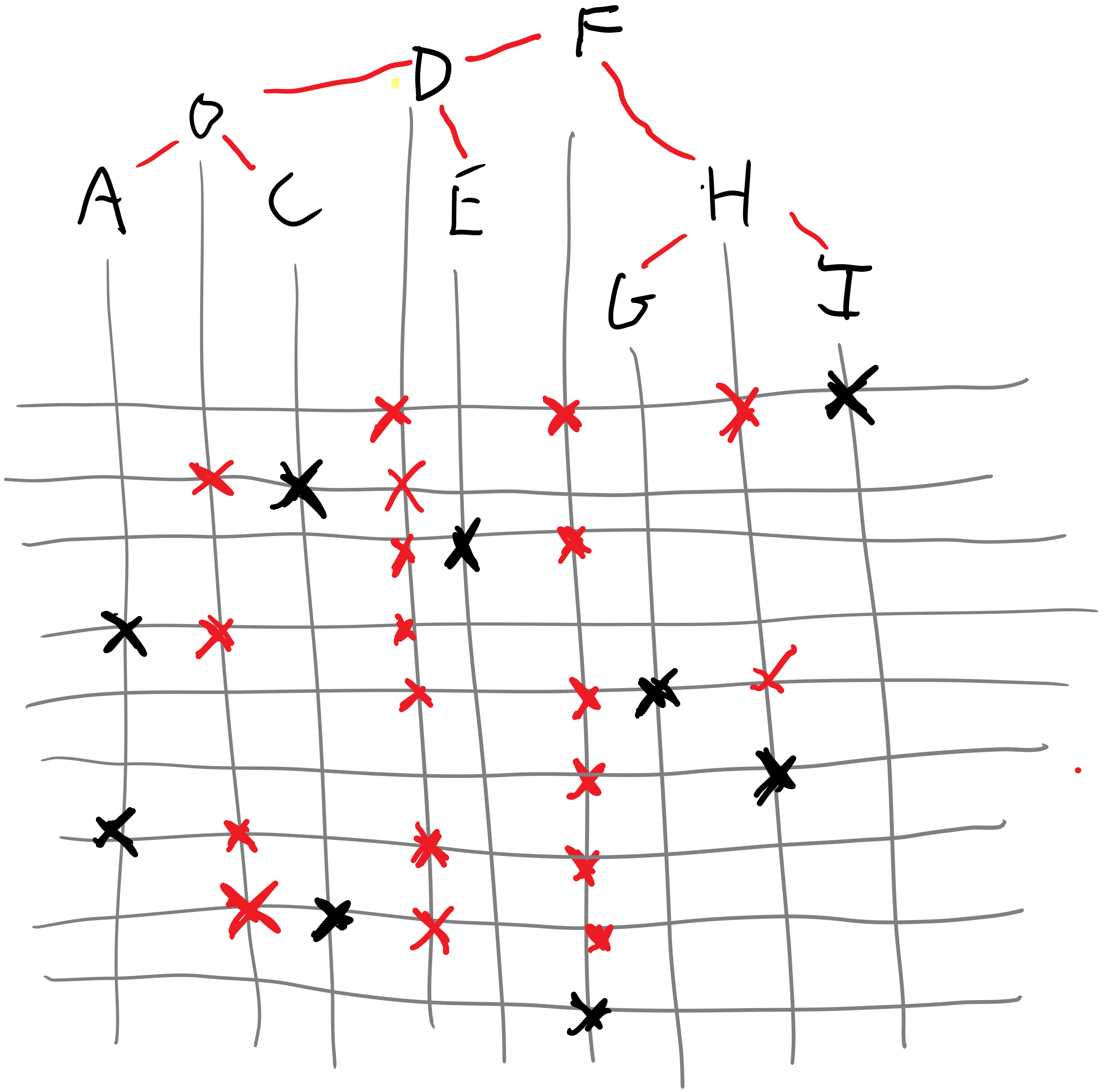


Rotate!
A
O
C
D
E
F
H
G
I

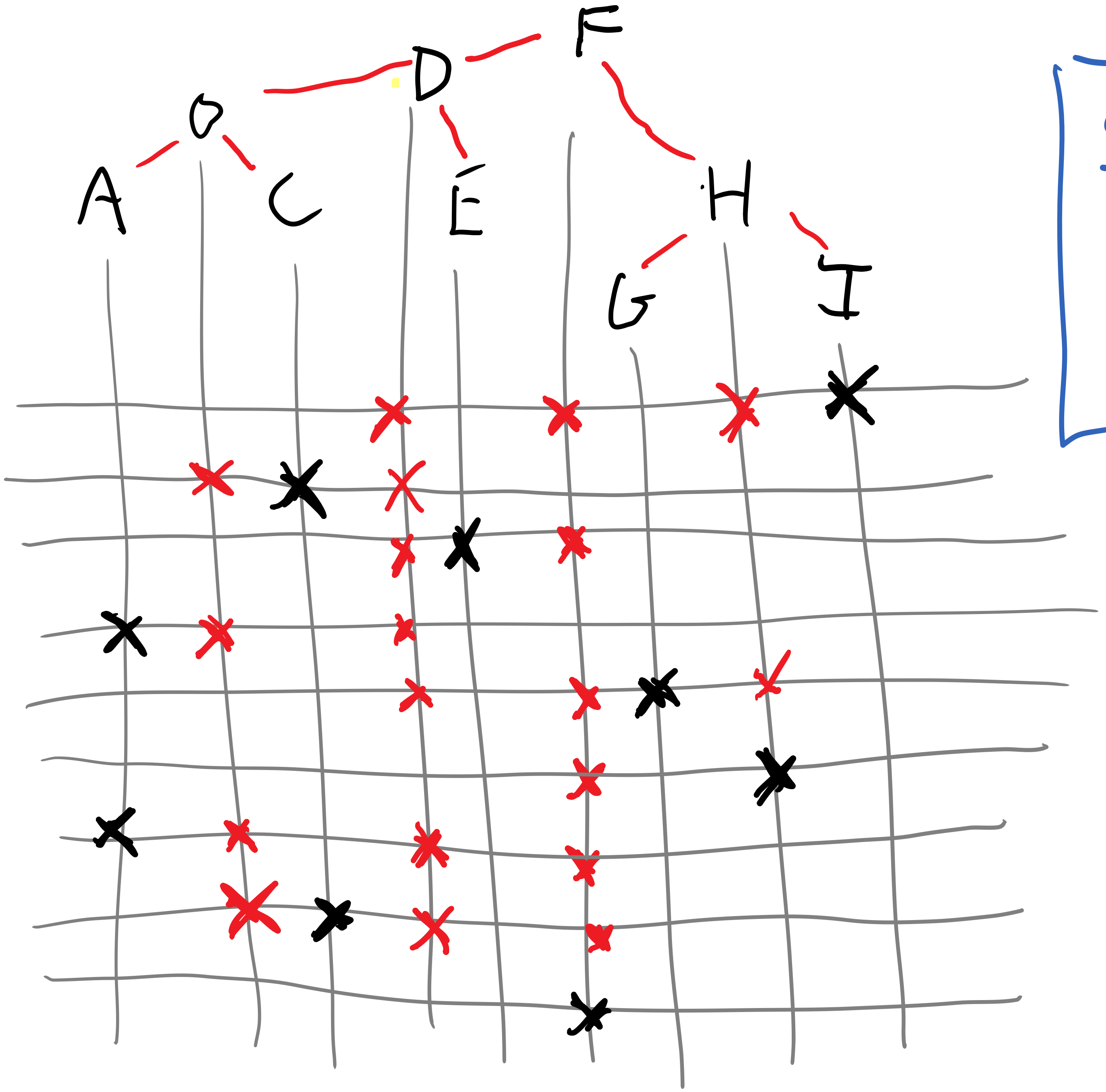


search for

H
A
C
F

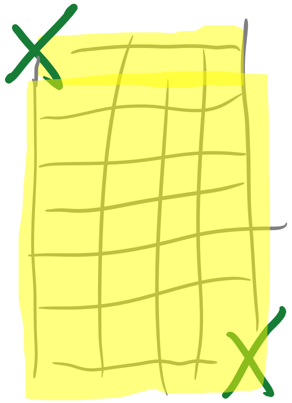


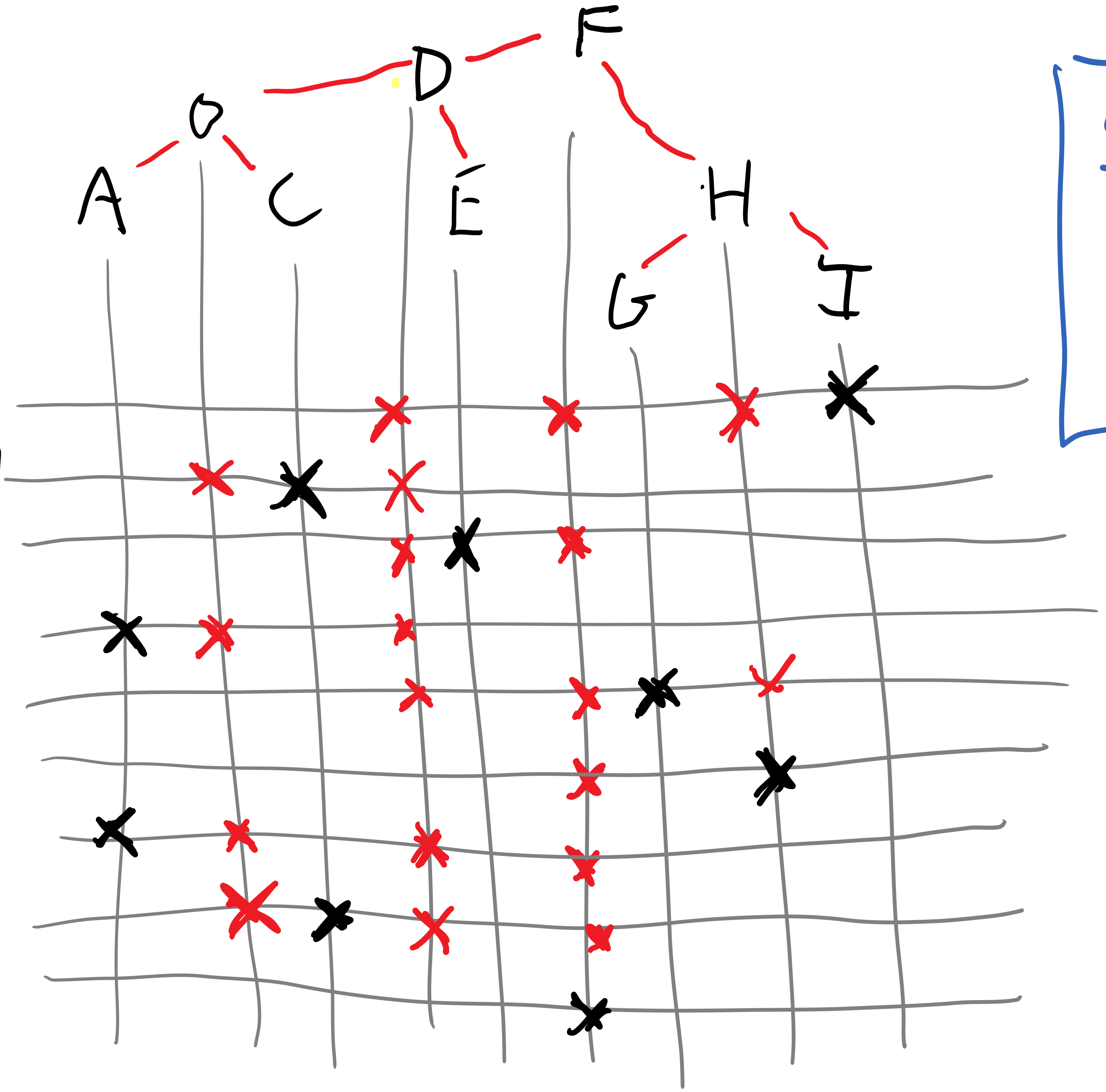
Cost to search in tree = Number of XX in this picture



Stare AND Think

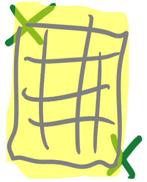
Can you Find

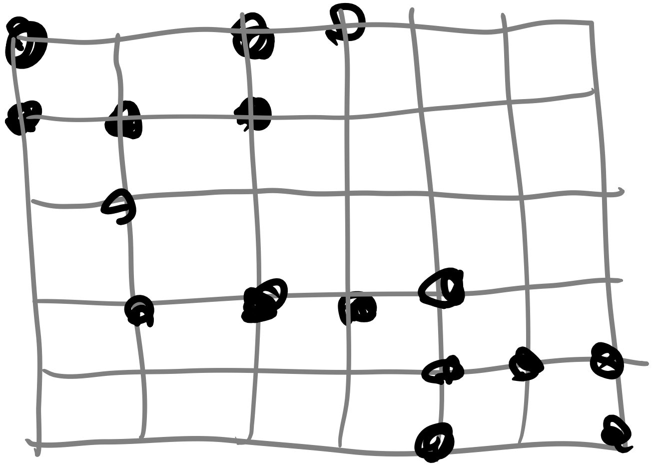


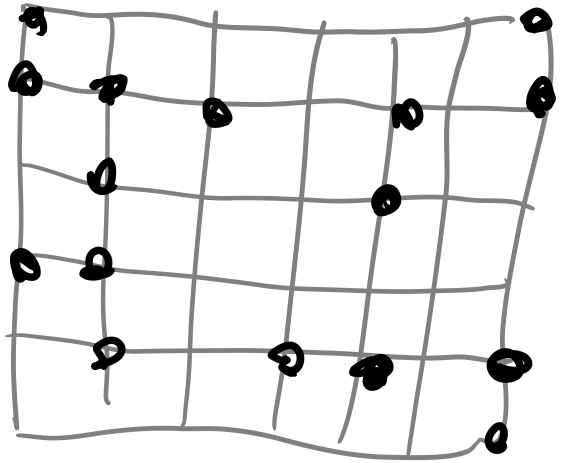


Stare AND Think

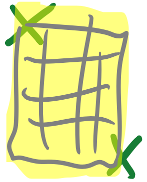
Definition:

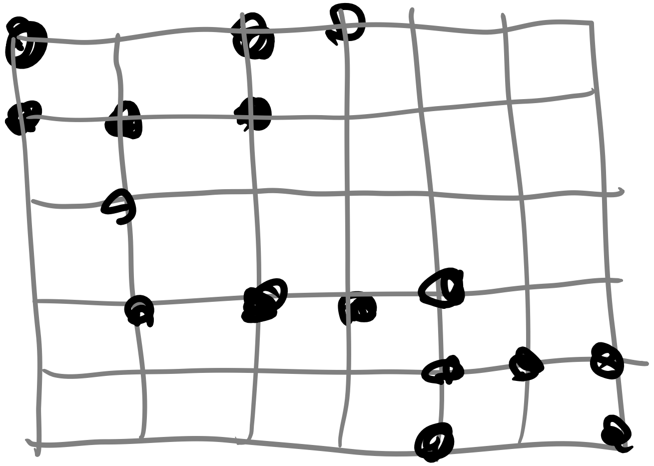
No  → "Arborally Satisfied Set"



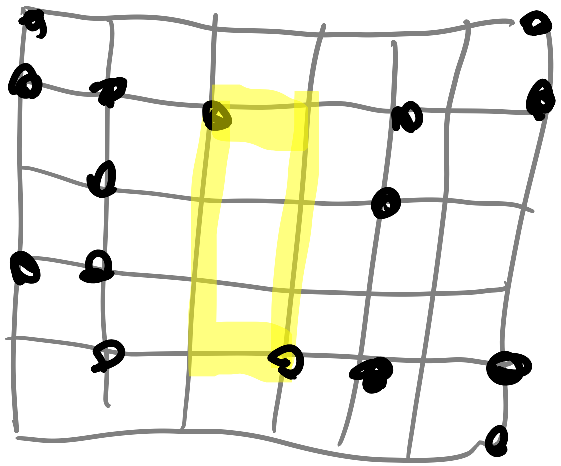


Definition:

No  → "Arbarally Satisfied Set"

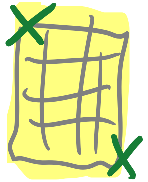


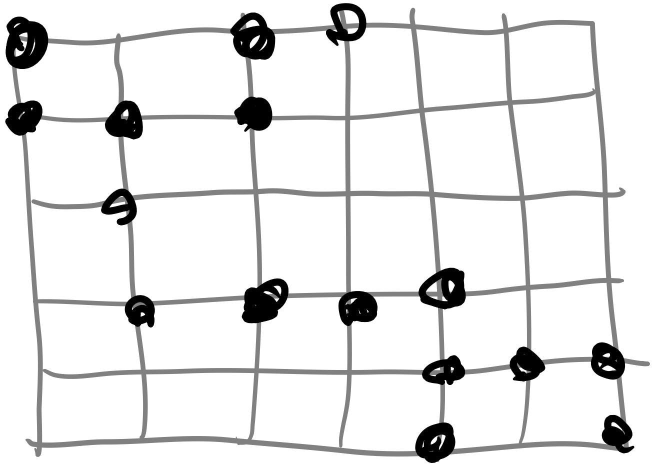
ASS



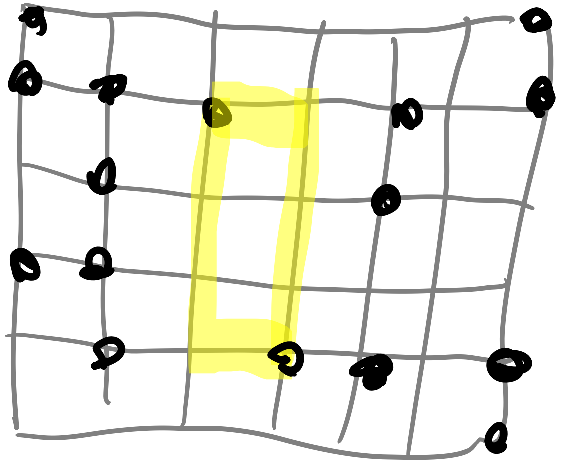
Not ASS

Definition:

No  → "Arbarally Satisfied Set"



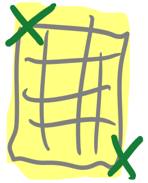
ASS

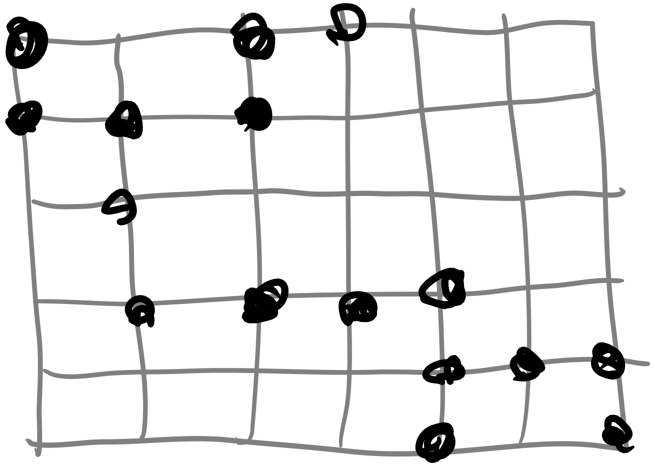


Not ASS

Theorem: Plotting any way to search for stuff → ASS Points

Definition:

No  $\rightarrow$ "Arborally Satisfied Set"

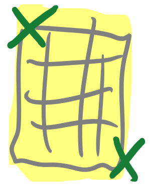


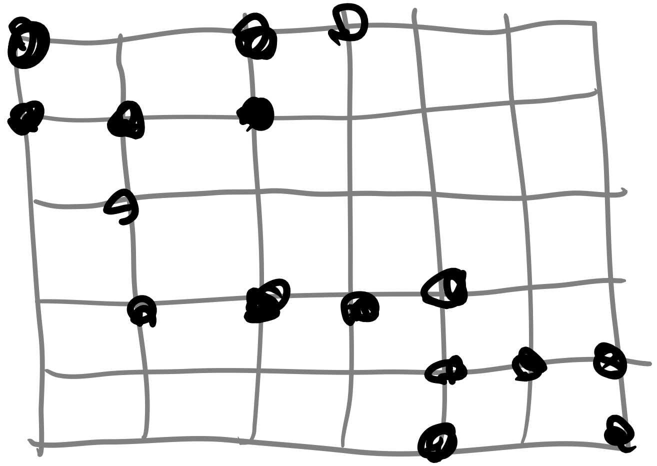
ASS

what about the reverse way?
Given an ASS point set like ←
can you find a way to
find stuff so that this
is the plot?

Theorem: Plotting any way to search for stuff
$\rightarrow$ ASS Points

Definition:

No  → "Arborally Satisfied Set"



ASS

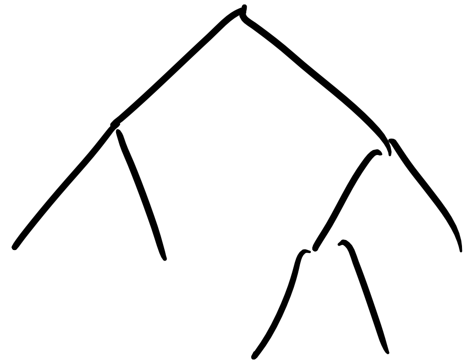
what about the reverse way?

Given an ASS point set like ←

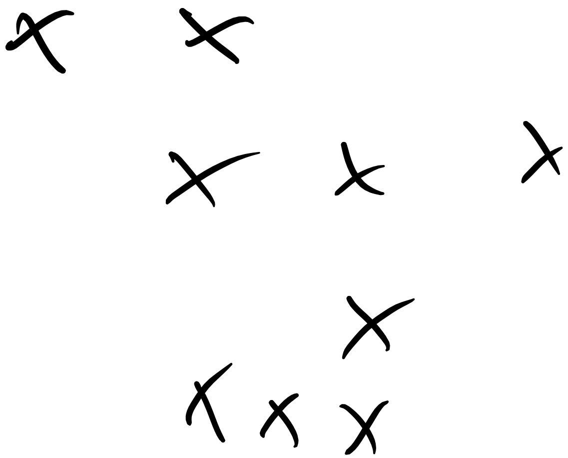
can you find a way to find stuff so that this is the plot? YES!

Theorem: Plotting any way to search for stuff → ASS Points

So we can forget about



And work with



The best way to find stuff

Looking for C, A, F, D, B, E

	A	B	C	D	E	F
C						
A						
F						
B						
D						
E						

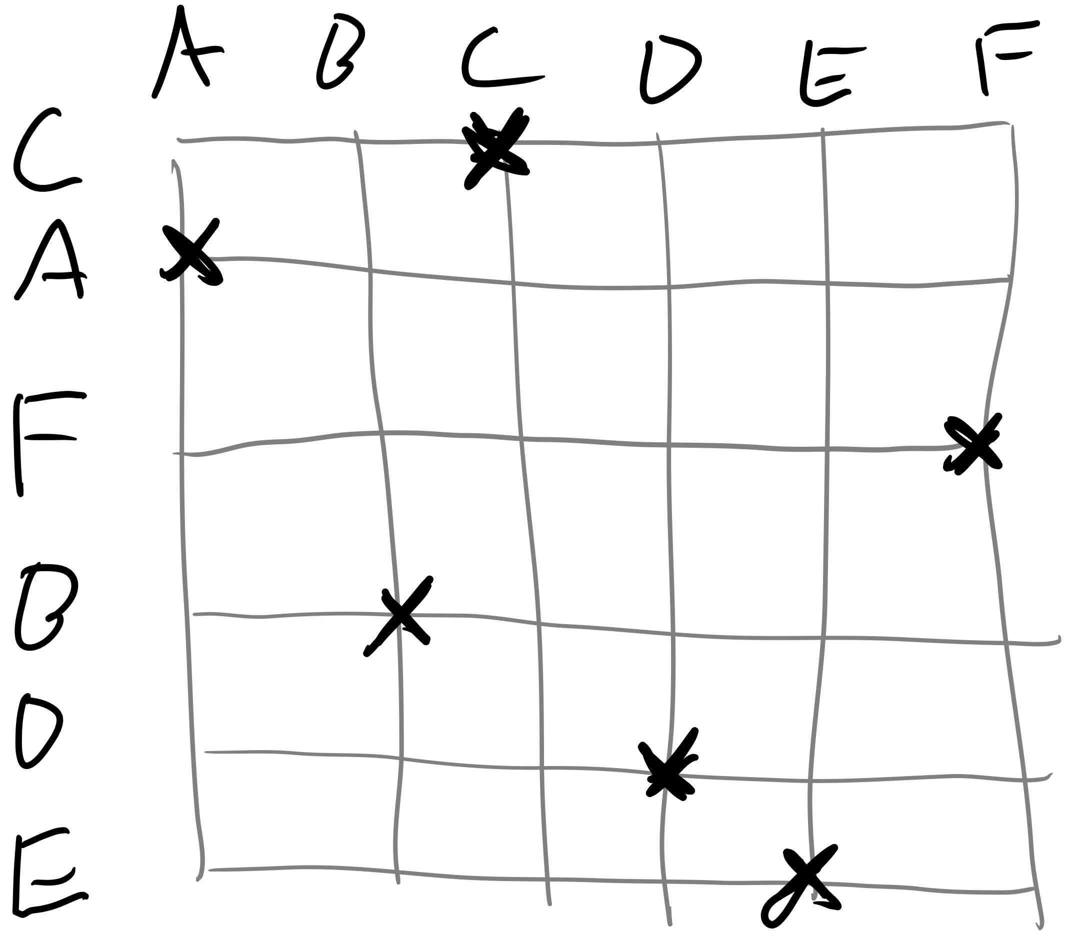
The best way to find Stuff

Looking for C, A, F, D, B, E

	A	B	C	D	E	F
C			X			
A	X					
F						X
B		X				
D				X		
E					X	

The best way to find stuff

Looking for C, A, F, D, B, E



What is the minimum number of X's to add to make this ASS?

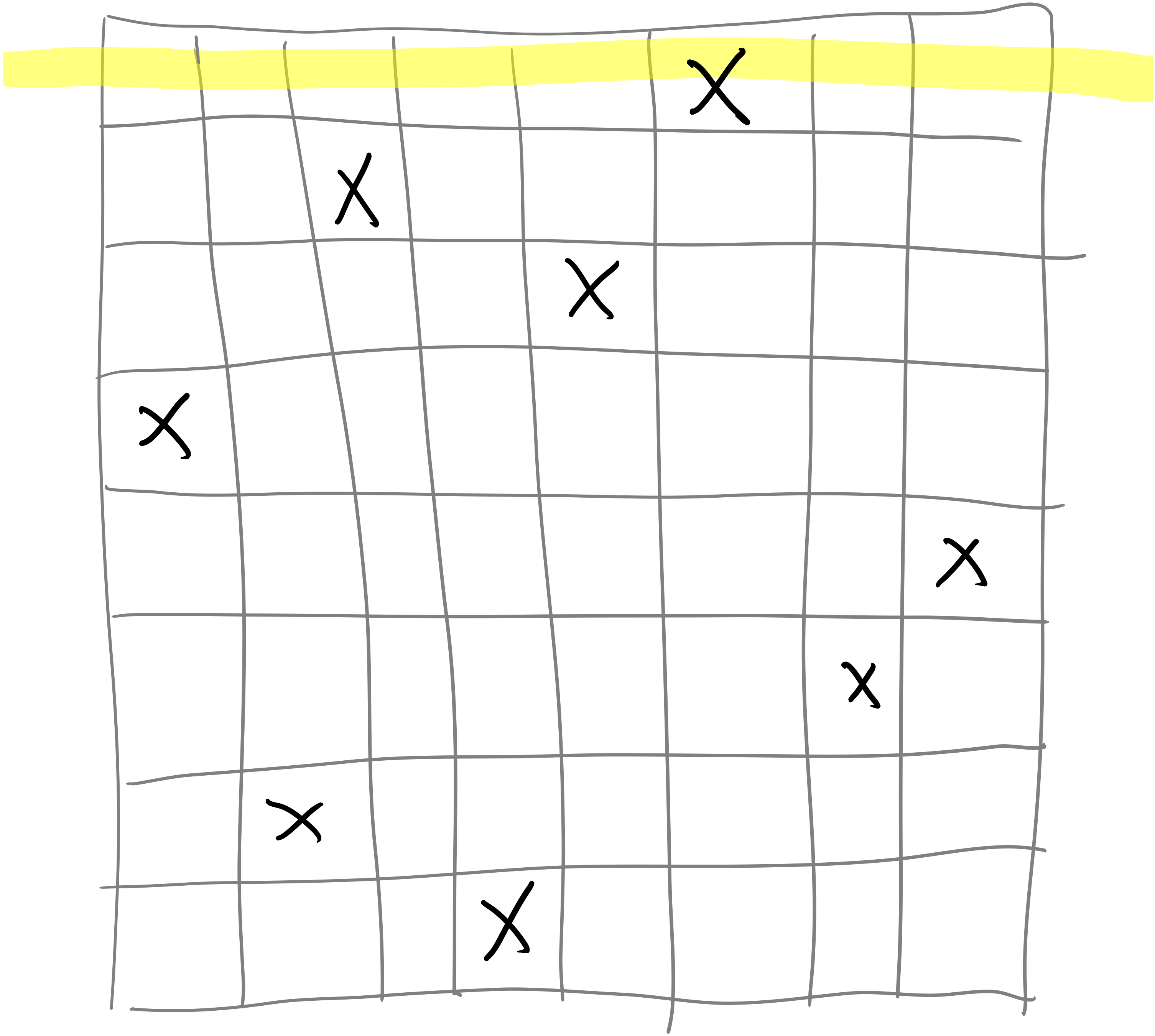
The best way to find Stuff

Looking for C, A, F, D, B, E

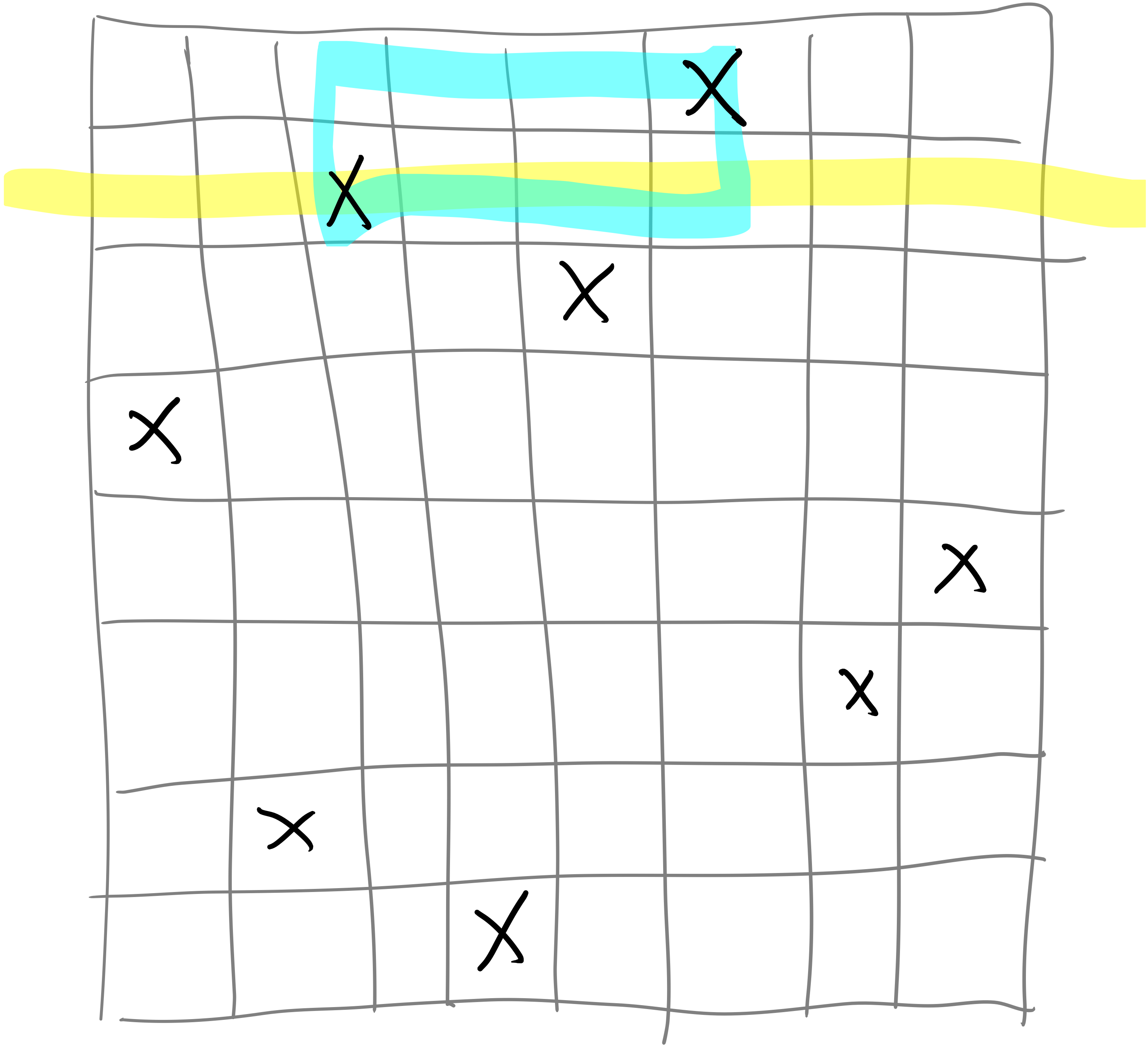
	A	B	C	D	E	F
C			X			
A	X		X			
F			X	X		X
B	X	X	X	X		
D				X	X	X
E					X	

What is the minimum number of X's to add to make this ASS?

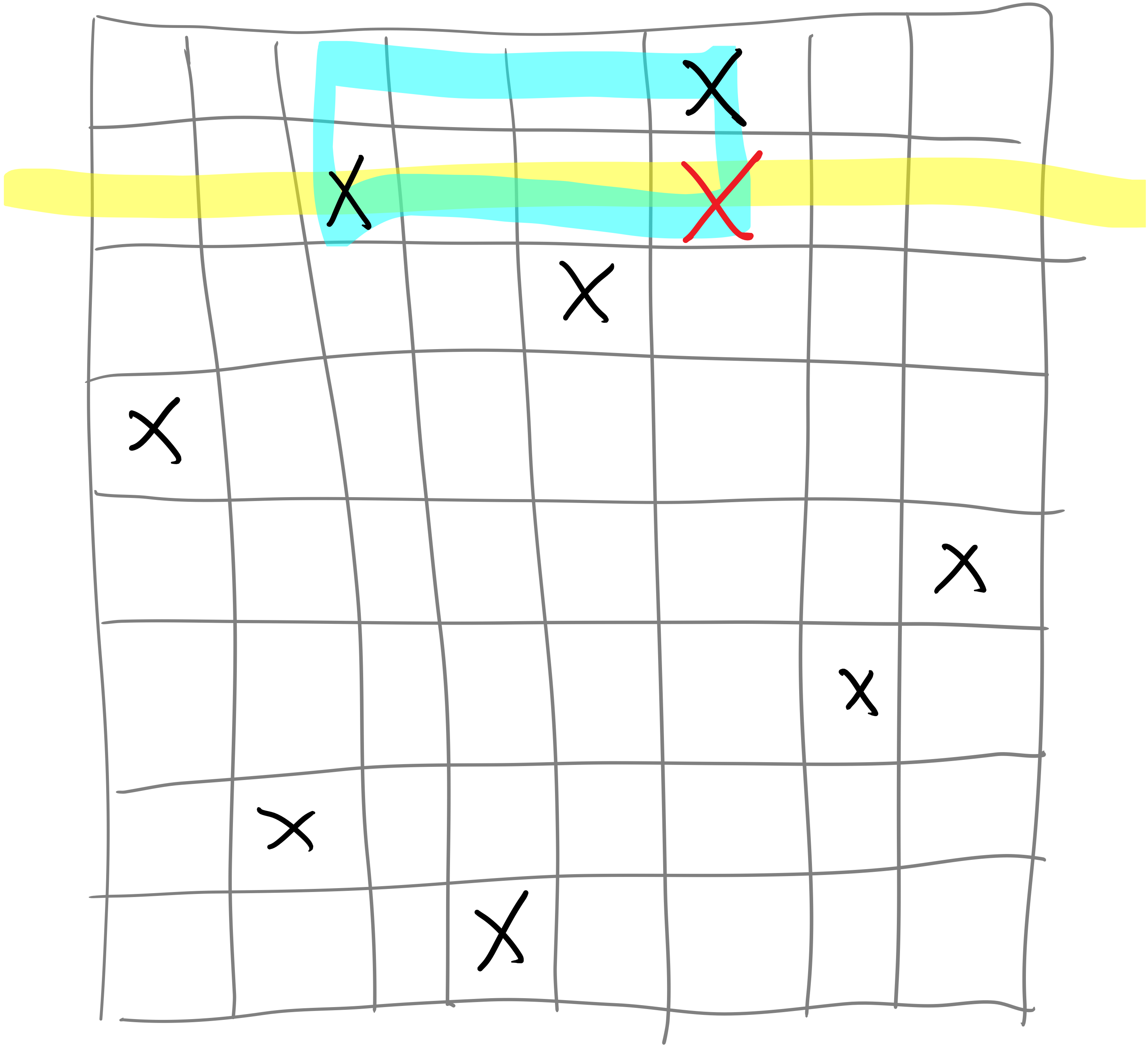
8 I think



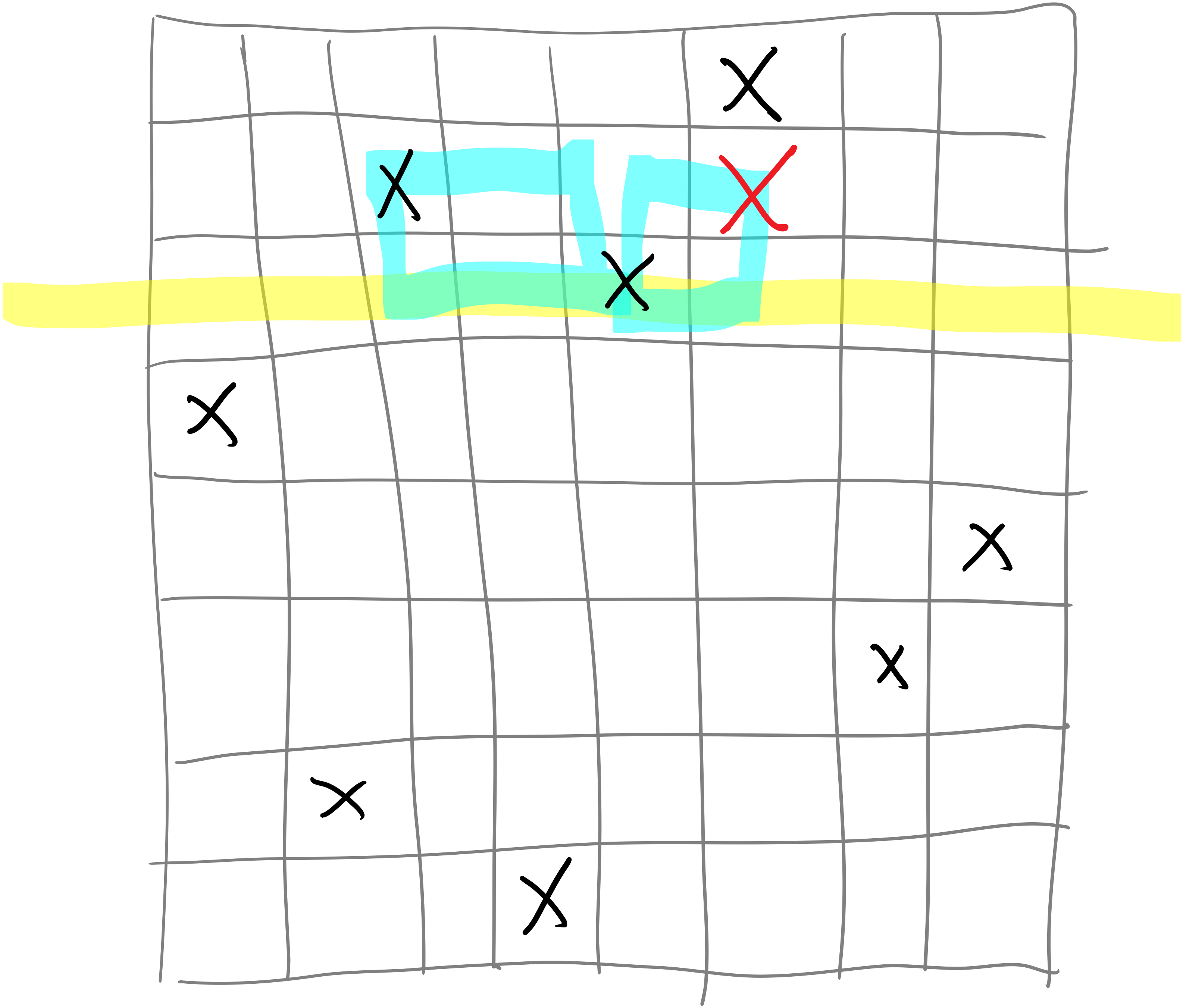
What is the most obvious way to add points to make this ASS?



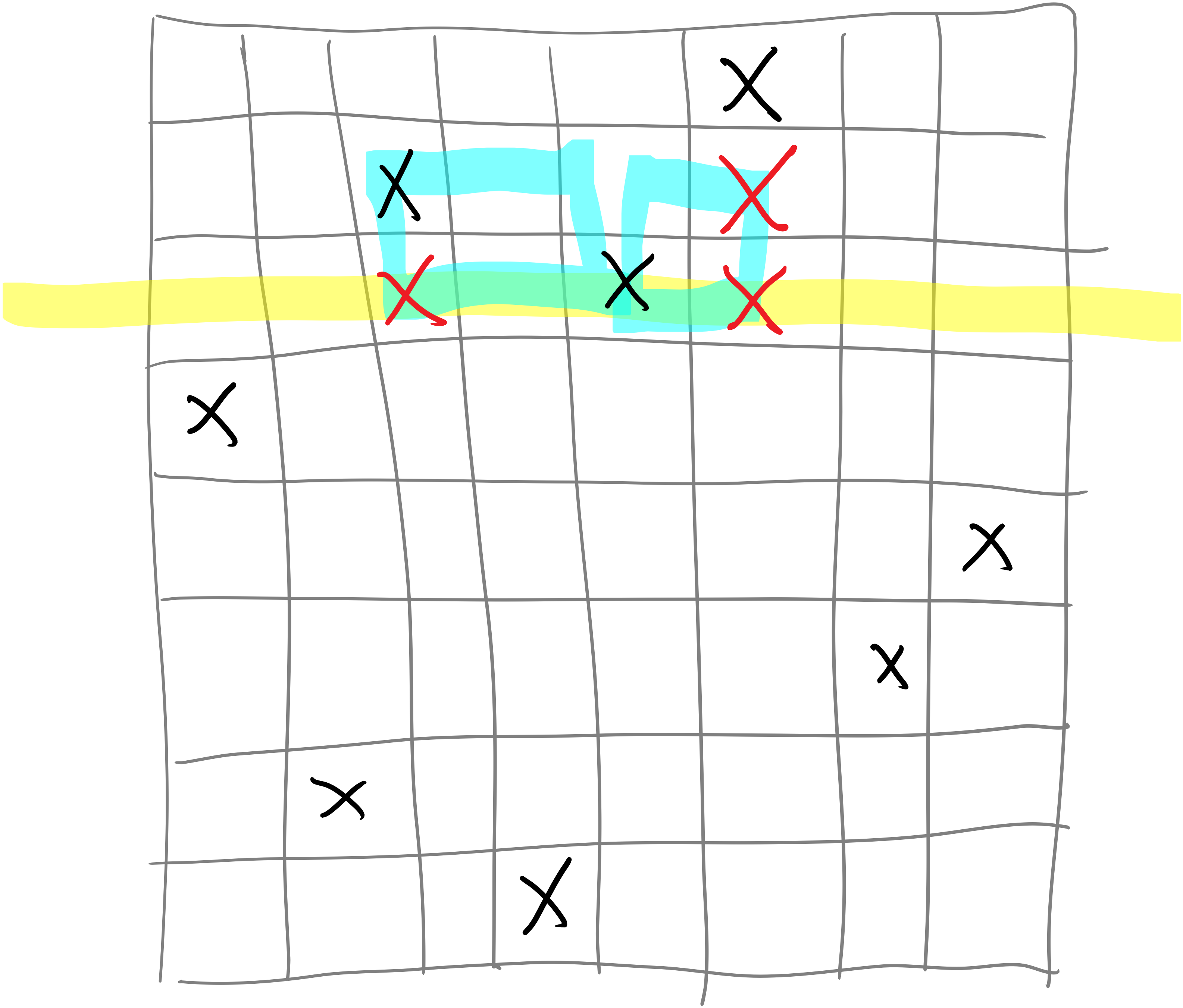
What is the most obvious way to add points to make this ASS?



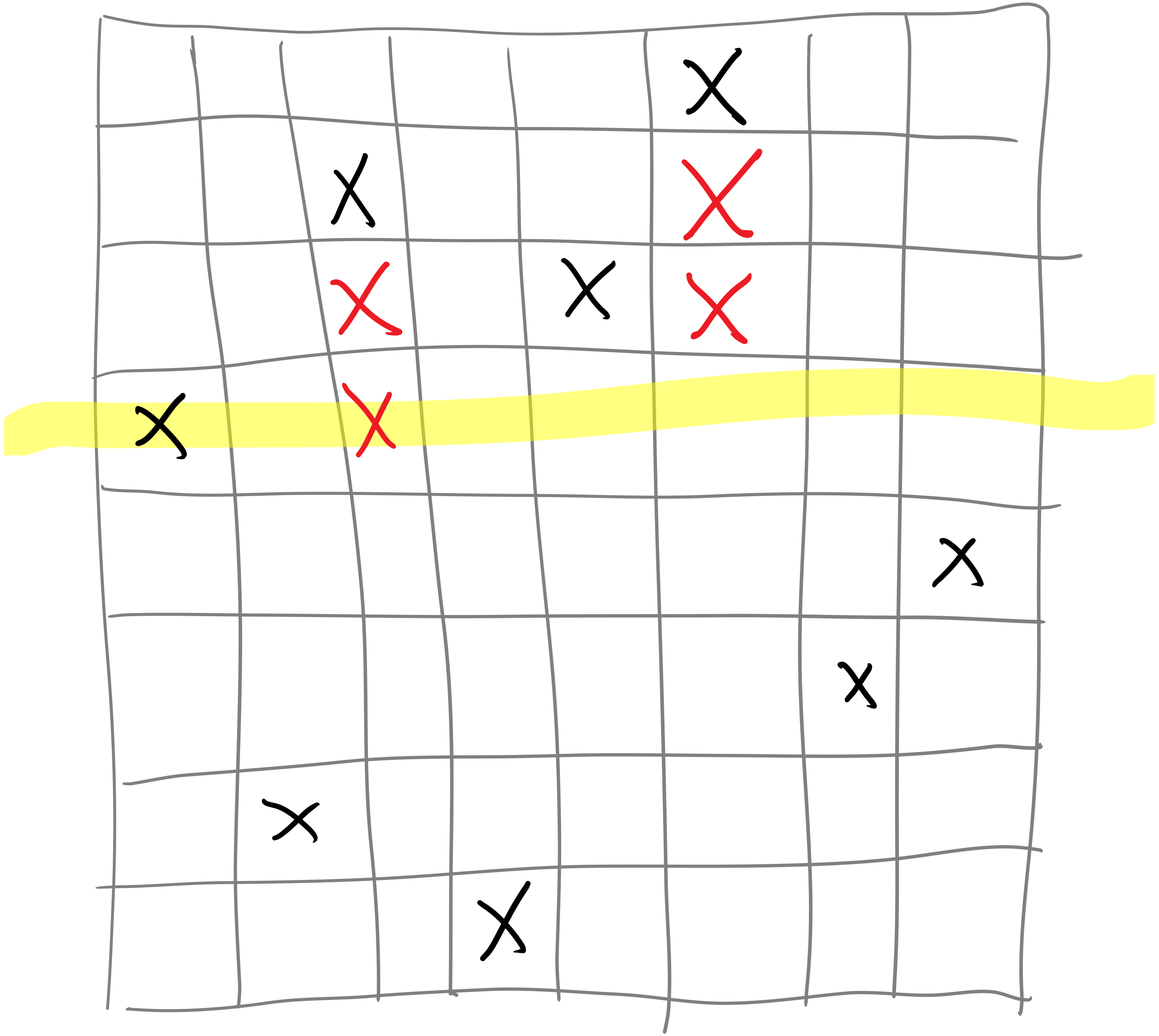
What is the most obvious way to add points to make this ASS?
.



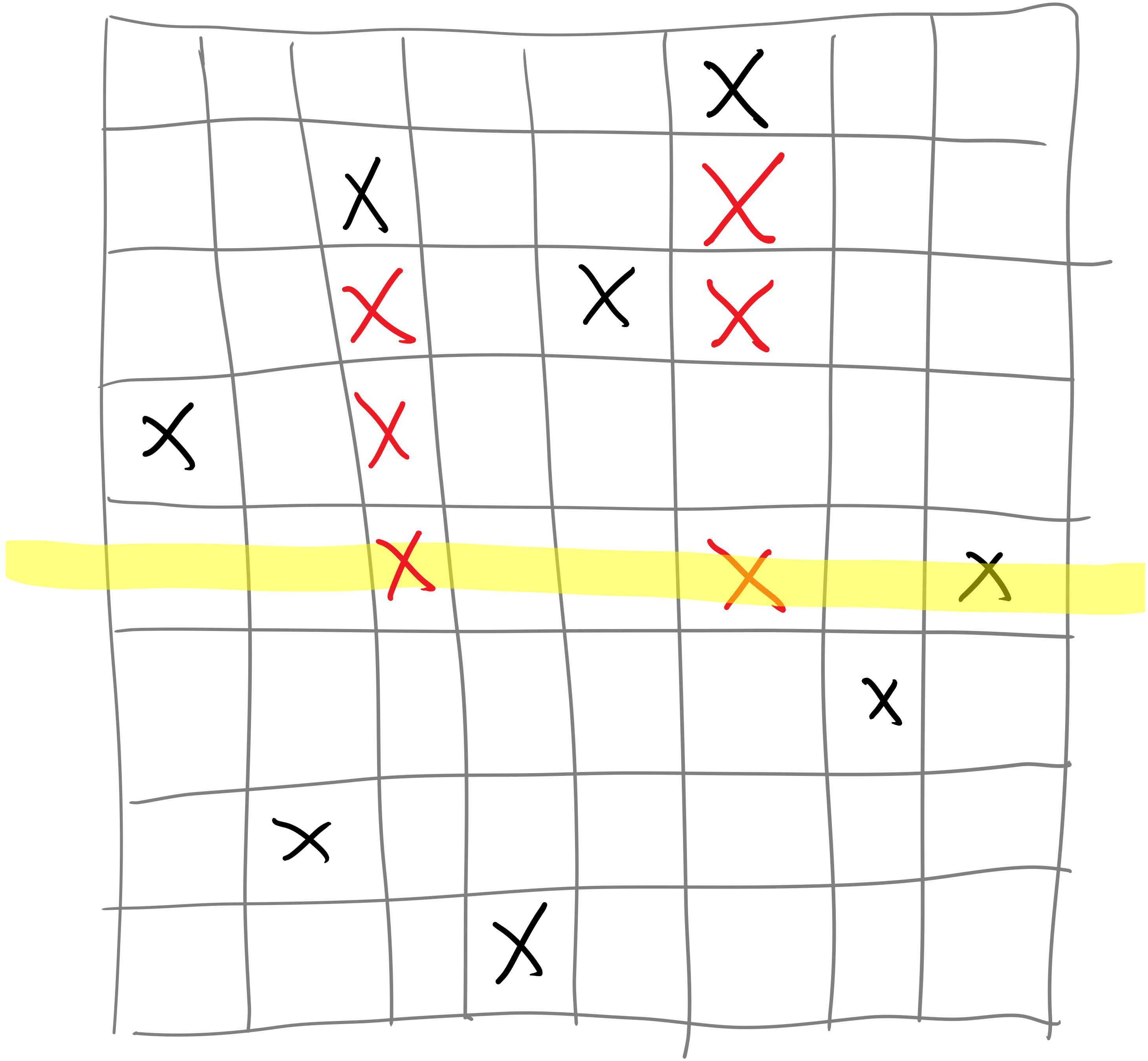
What is the most obvious way to add points to make this ASS?
.



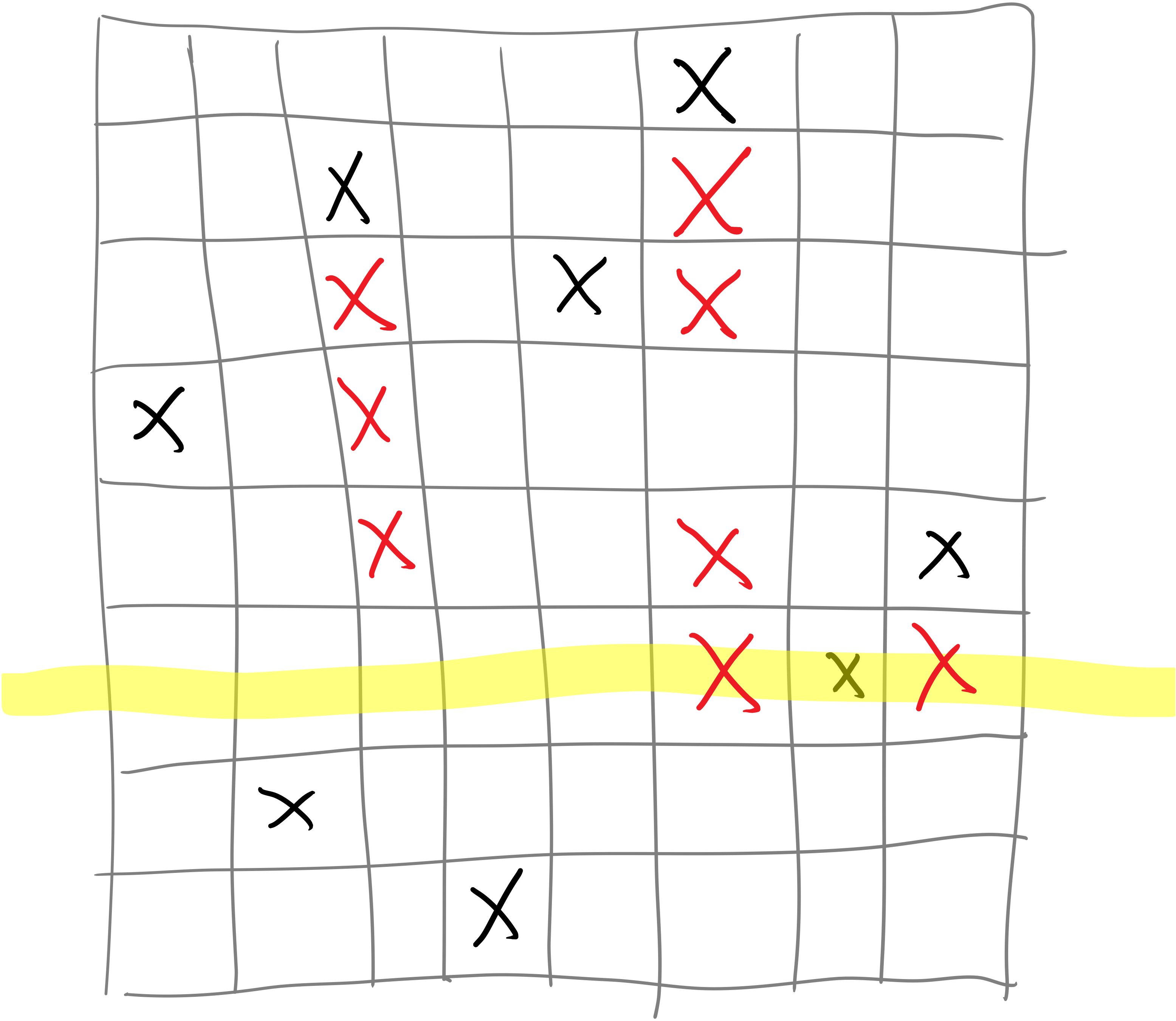
What is the most obvious way to add points to make this ASS?



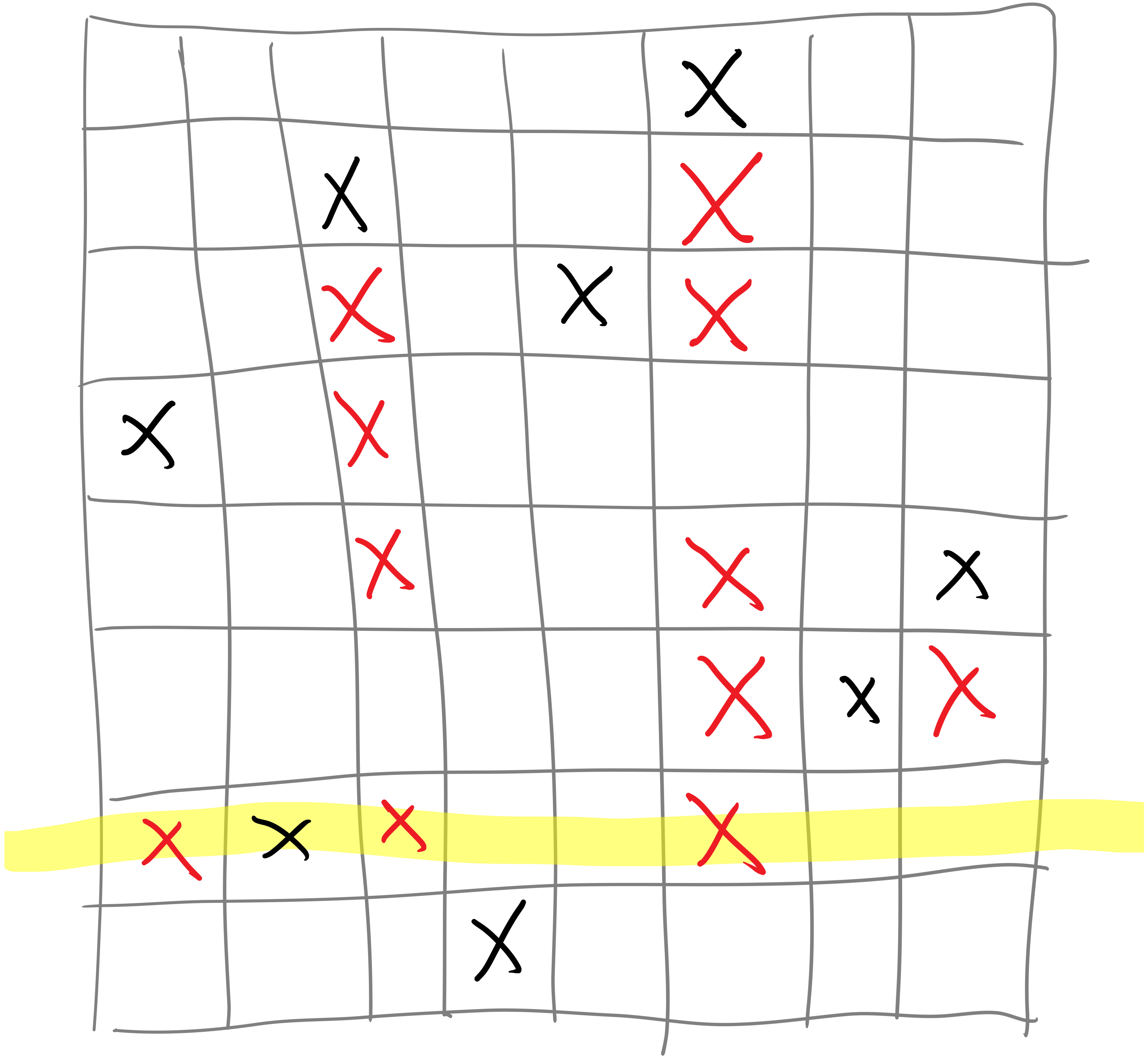
What is the most obvious way to add points to make this ASS ?



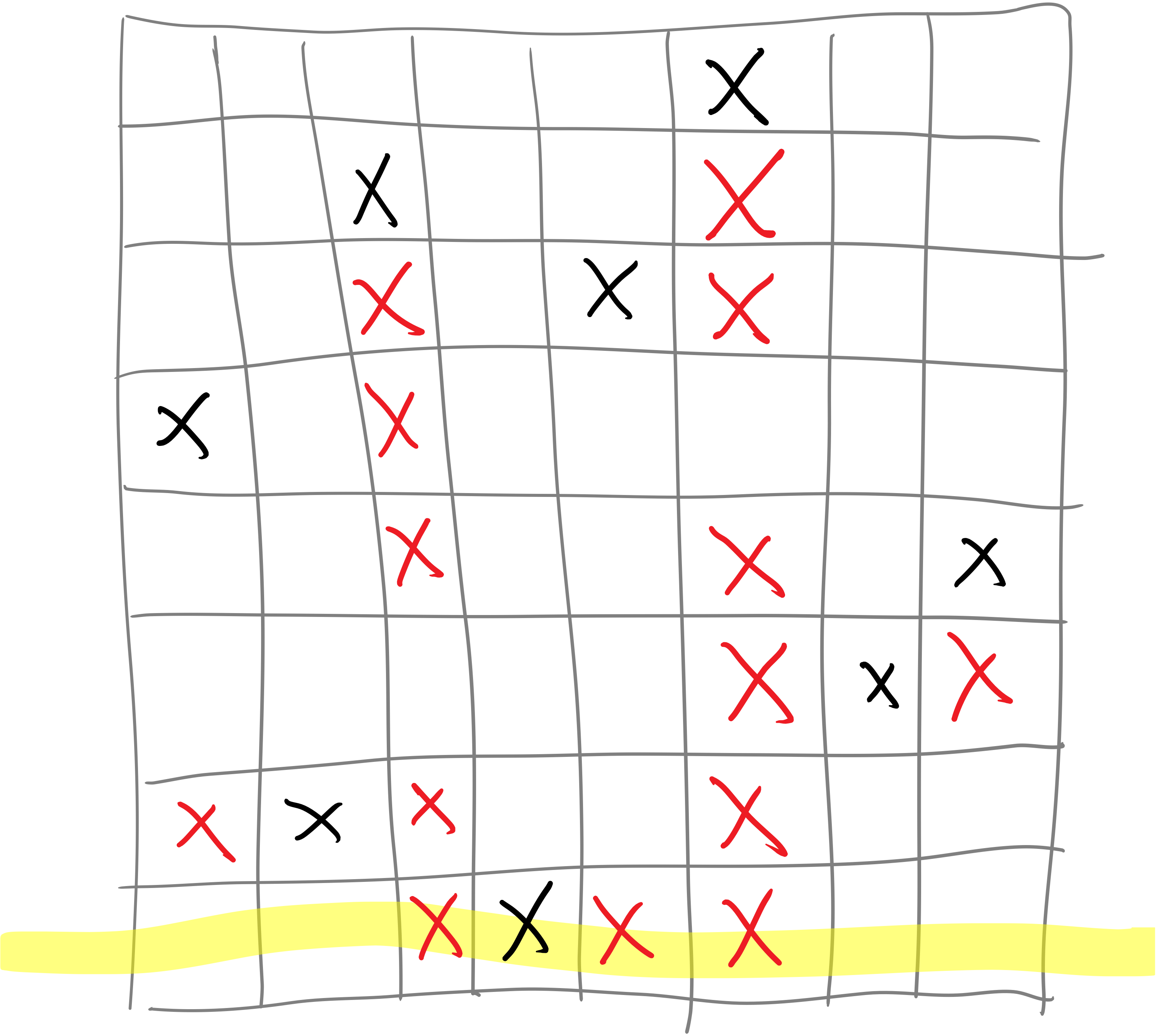
What is the most obvious way to add points to make this ASS?



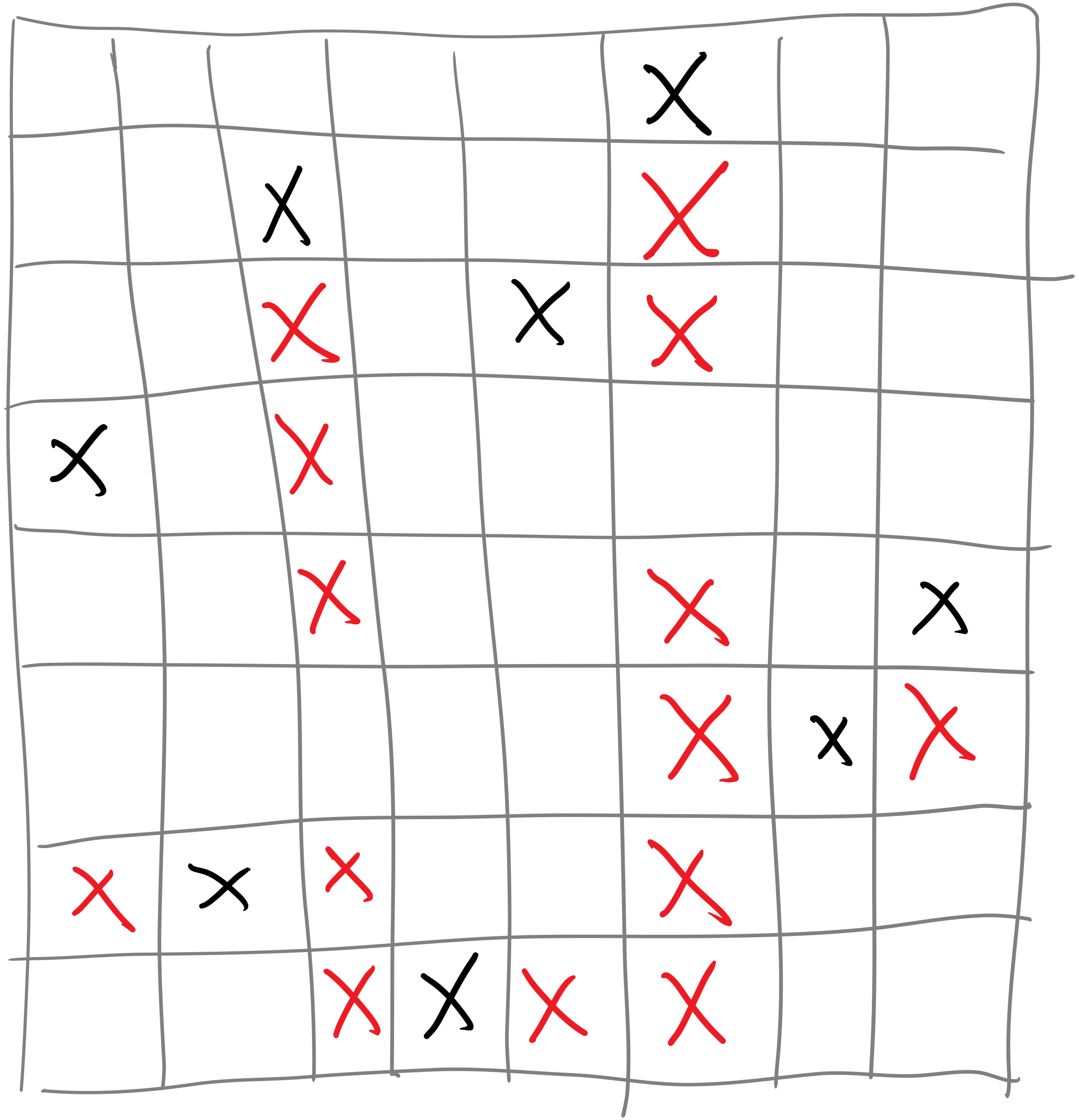
What is the most obvious way to add points to make this ASS?
.



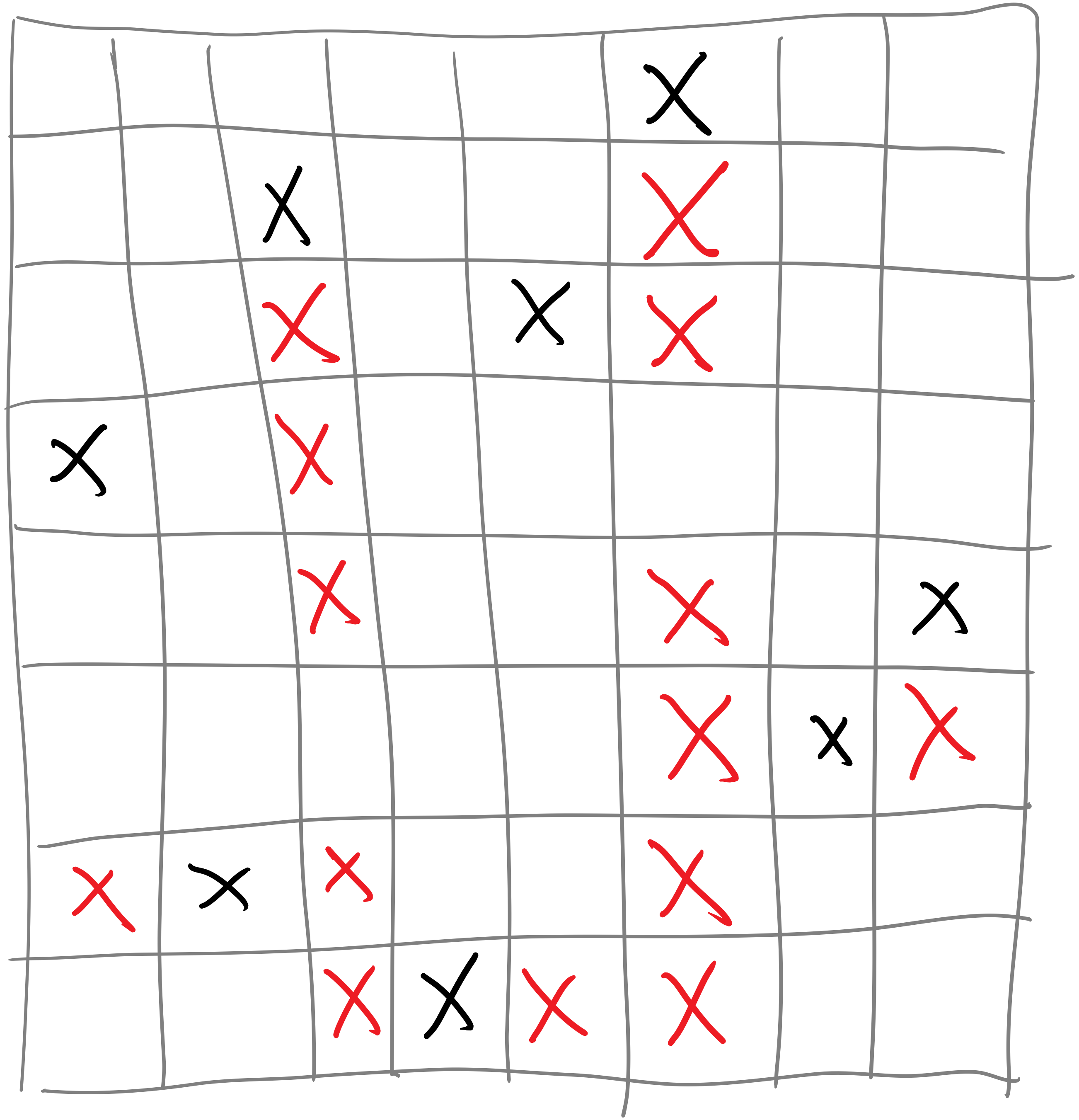
What is the most obvious way to add points to make this ASS?



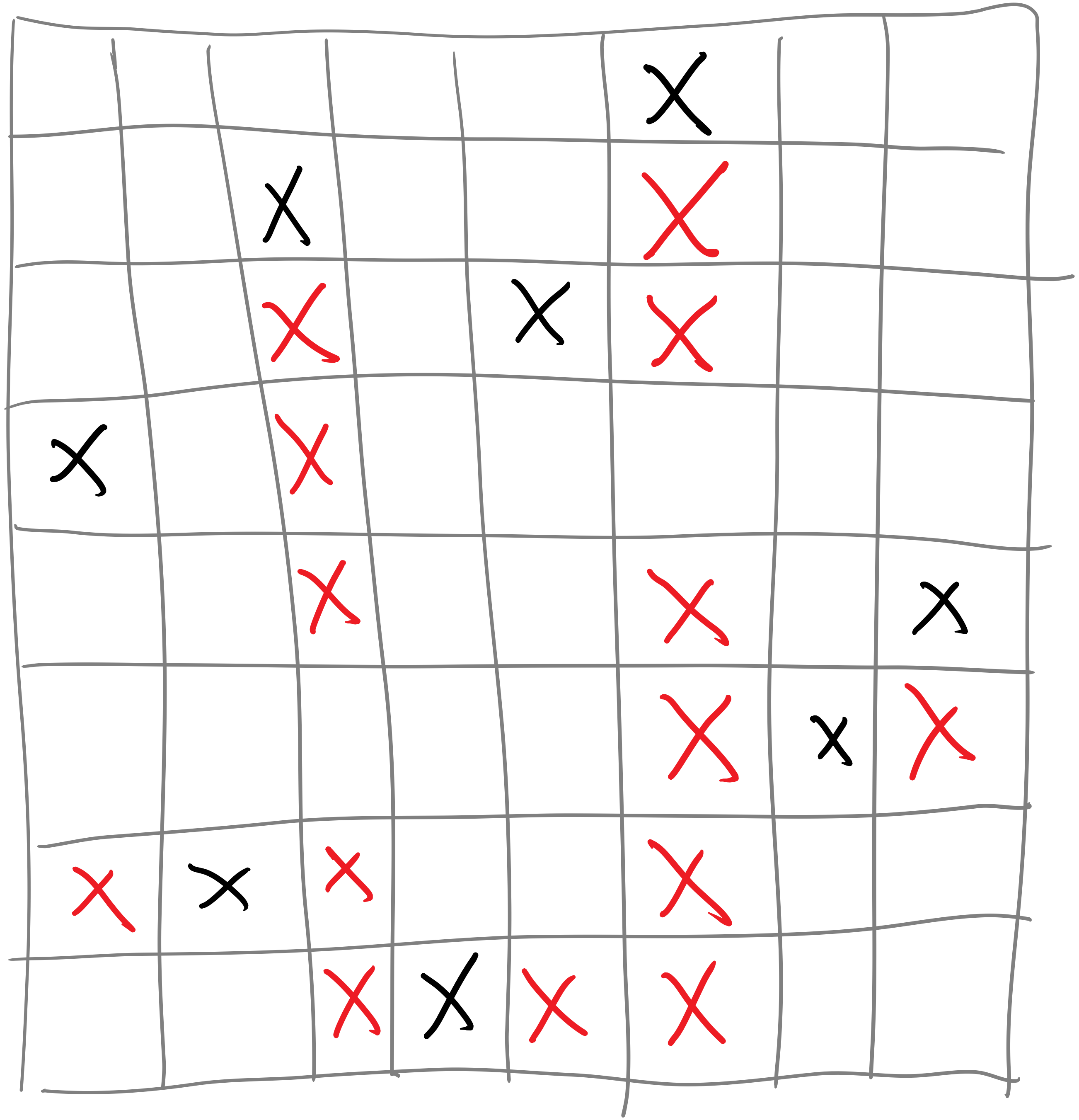
What is the most obvious way to add points to make this ASS?
.



What is the most obvious way to add points to make this ASS ?

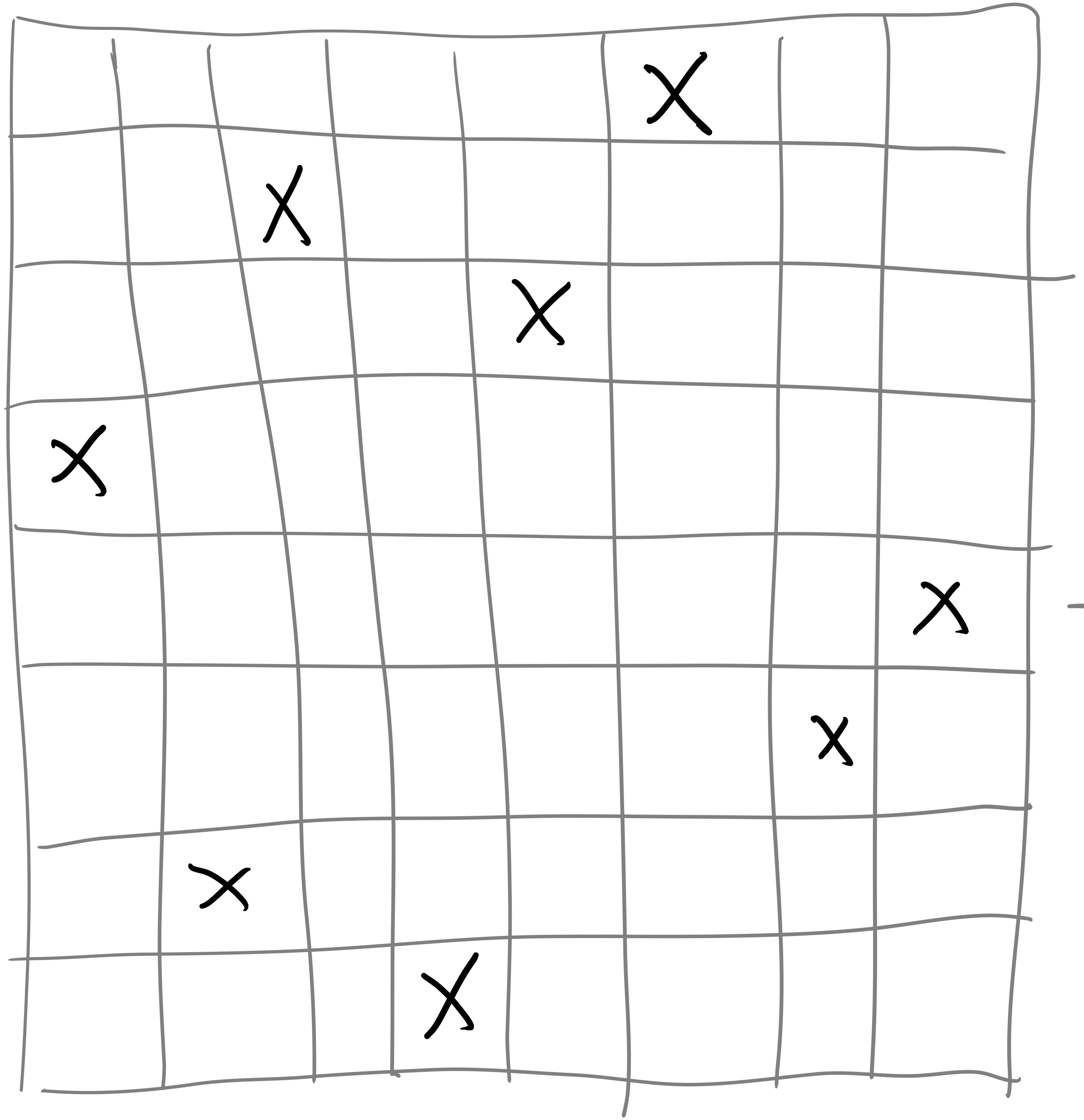


Is this the minimum # of X's?



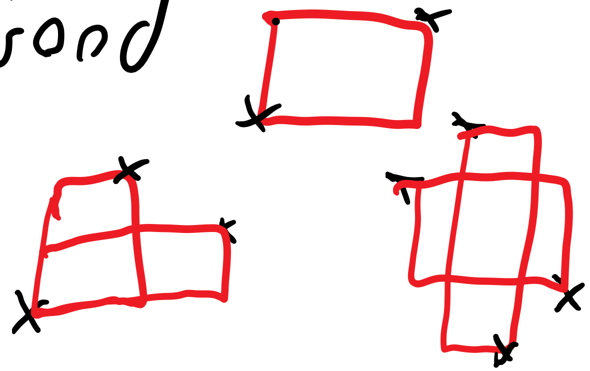
Is this the minimum # of X's?

No, but it appears to be at most double the minimum

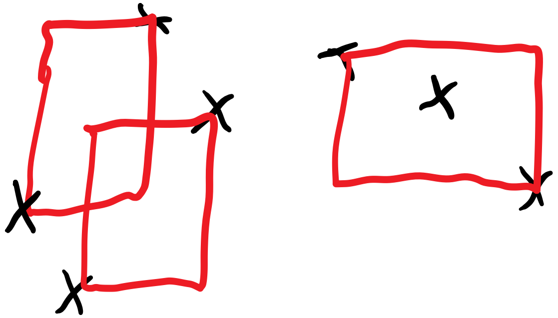


Independent Rectangles

Good

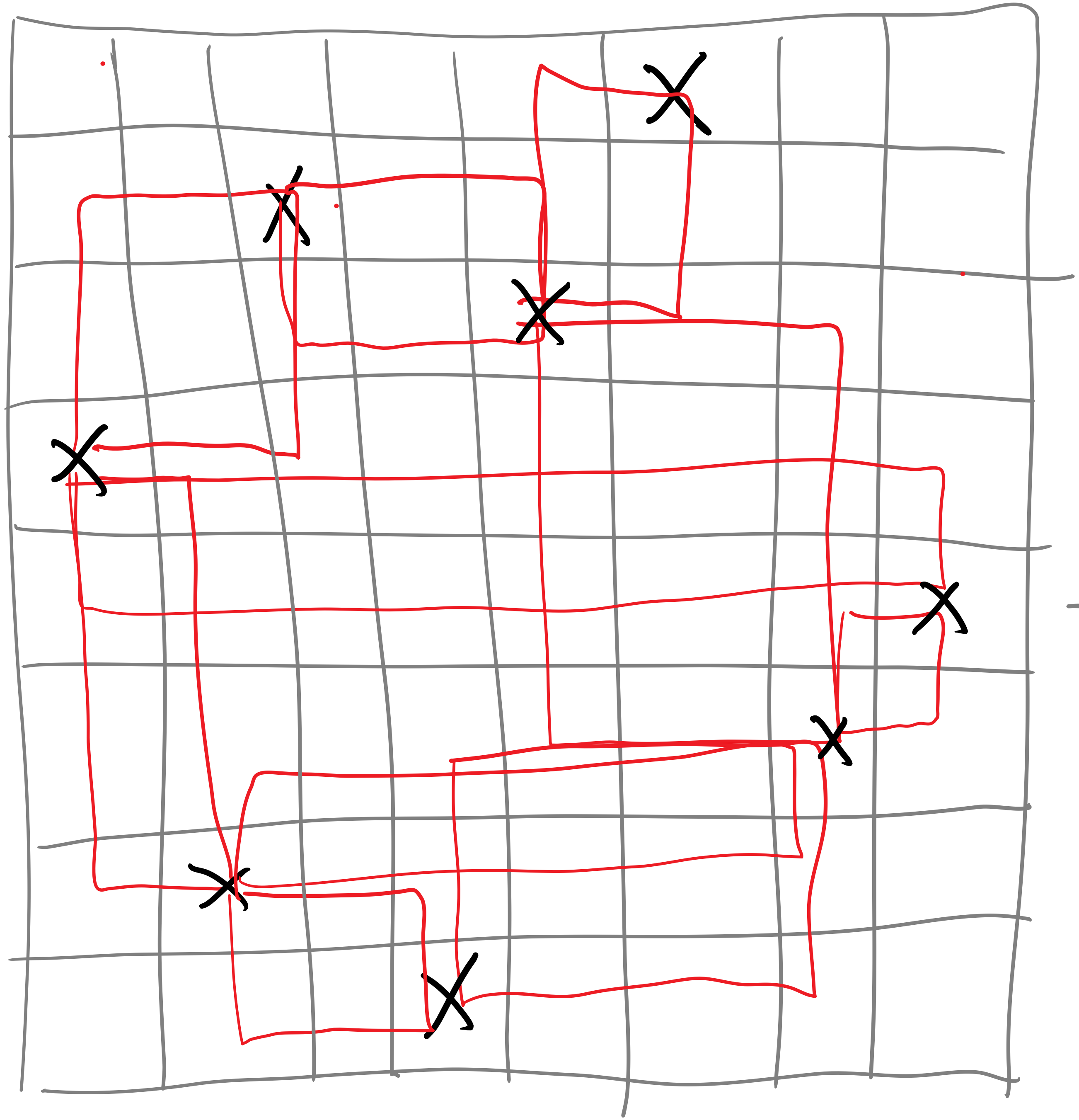


Bad

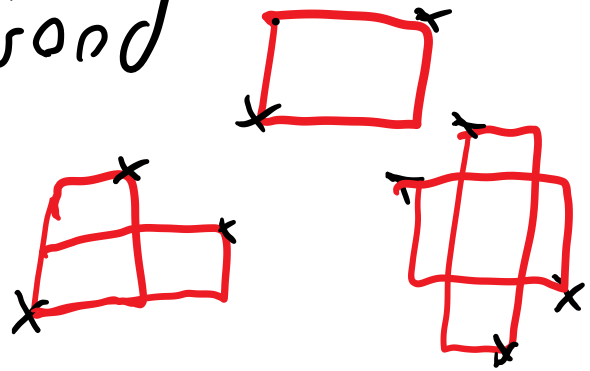


Independent Rectangles

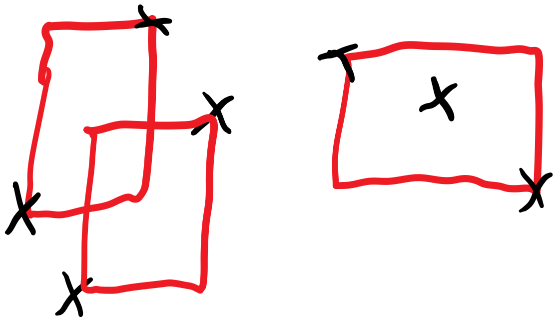
10 IND Rects



Good



Bad

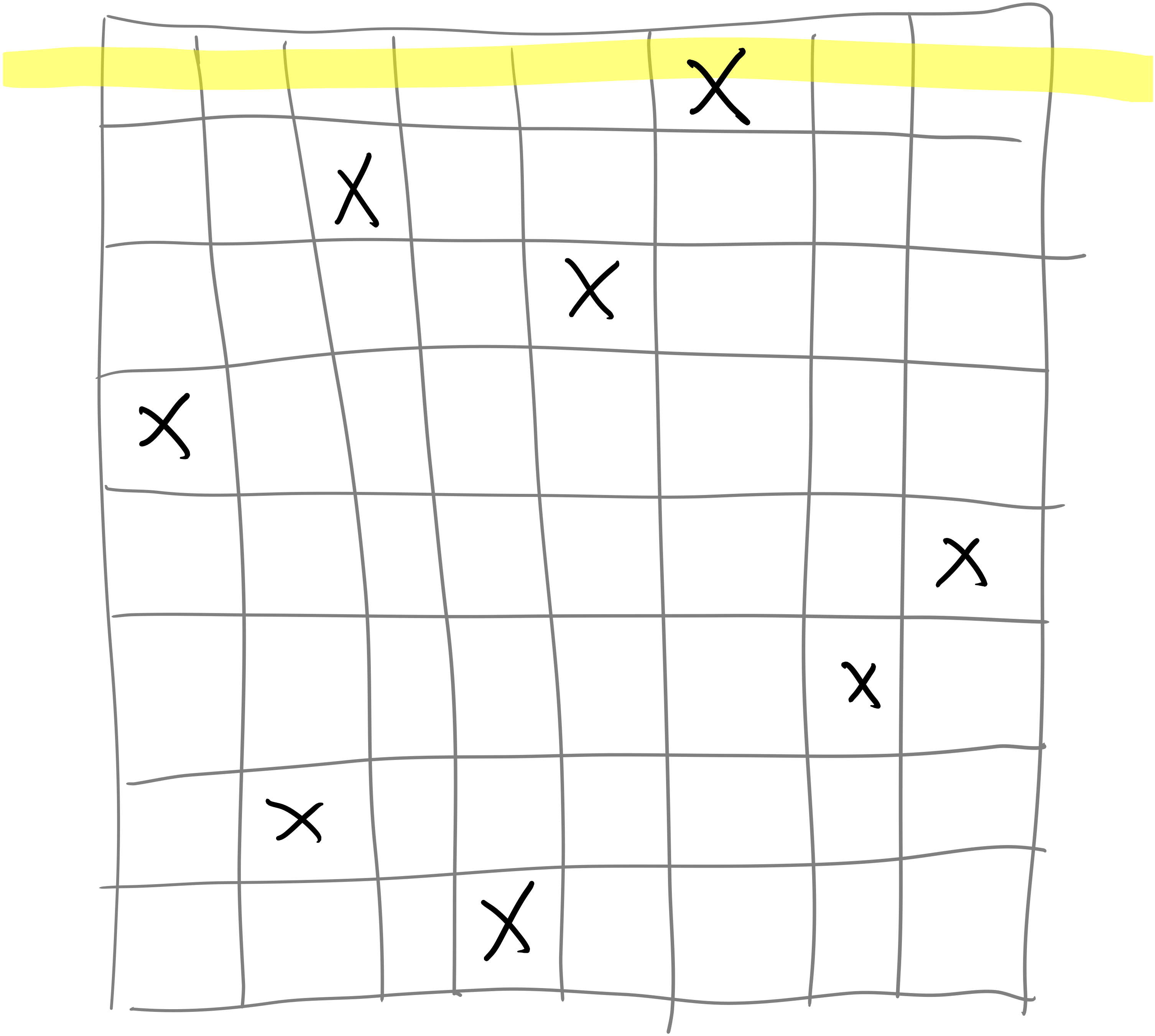


Independent Rectangles = Lower Bound

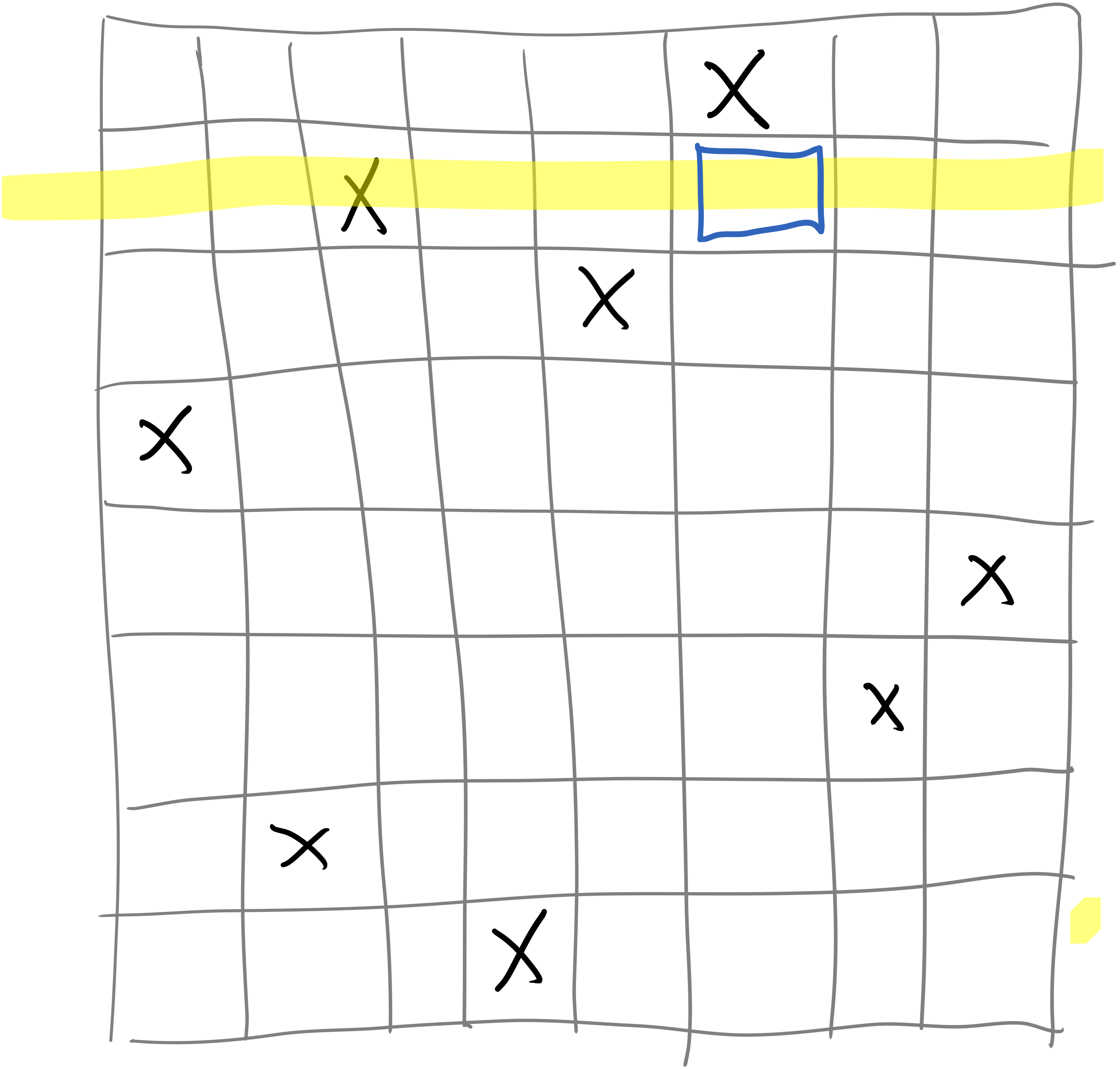
\# of ind Rects in geometric view of X $\leq$ OPT(X)

Independent Rectangles = Lower Bound

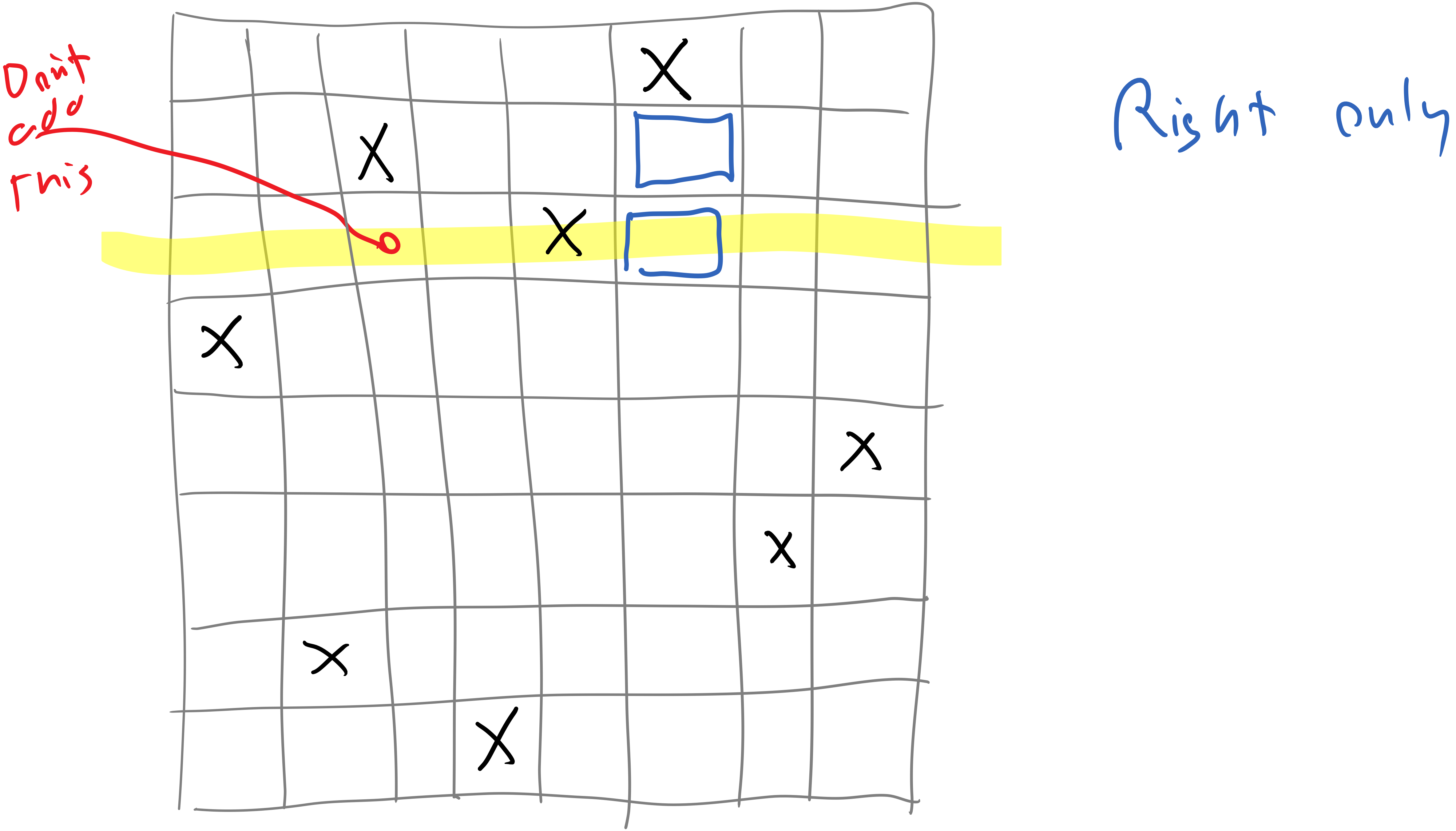
MAX # of ind Rects in geometric view of X $\le$ OPT(X).

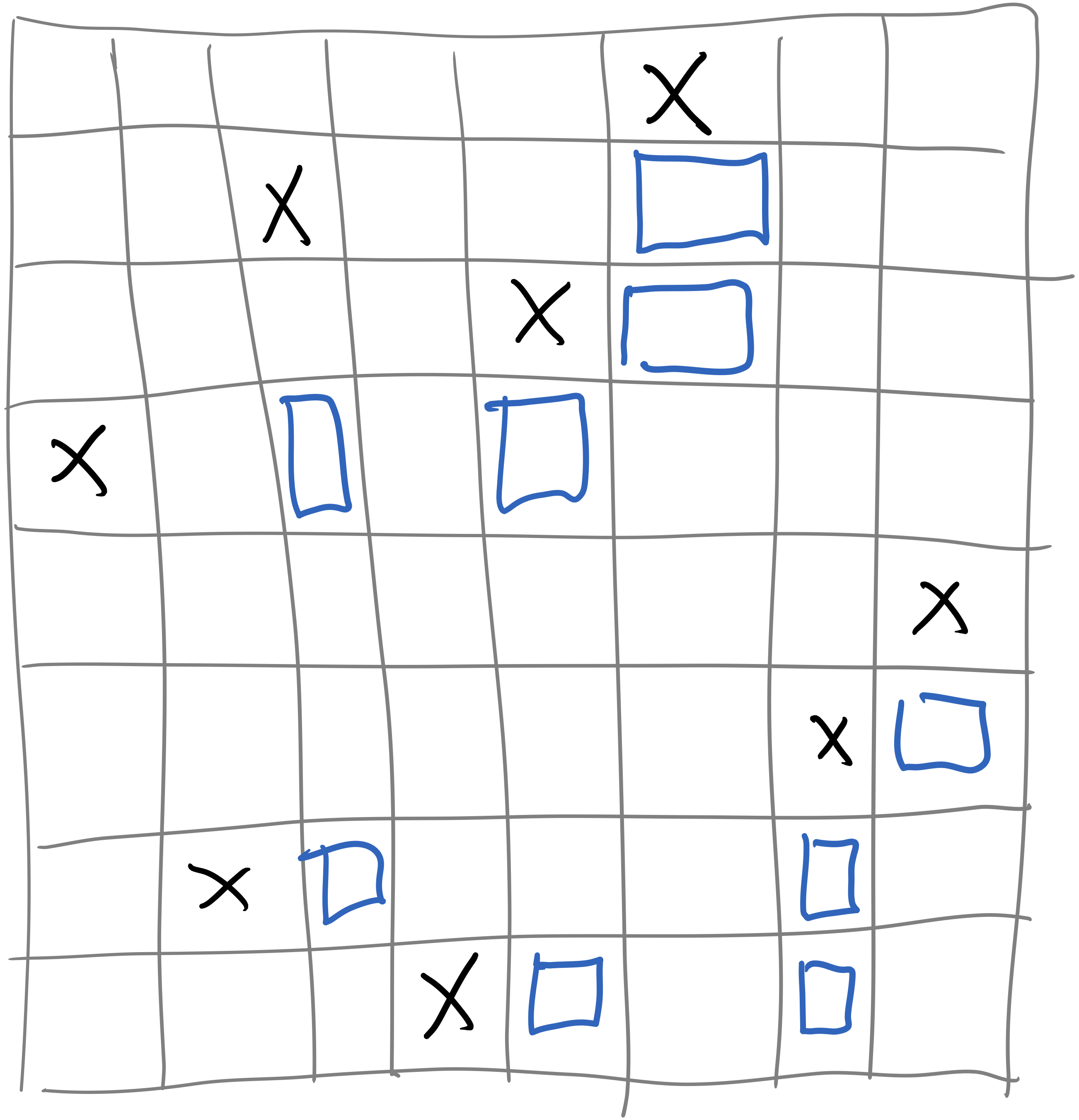


Right only



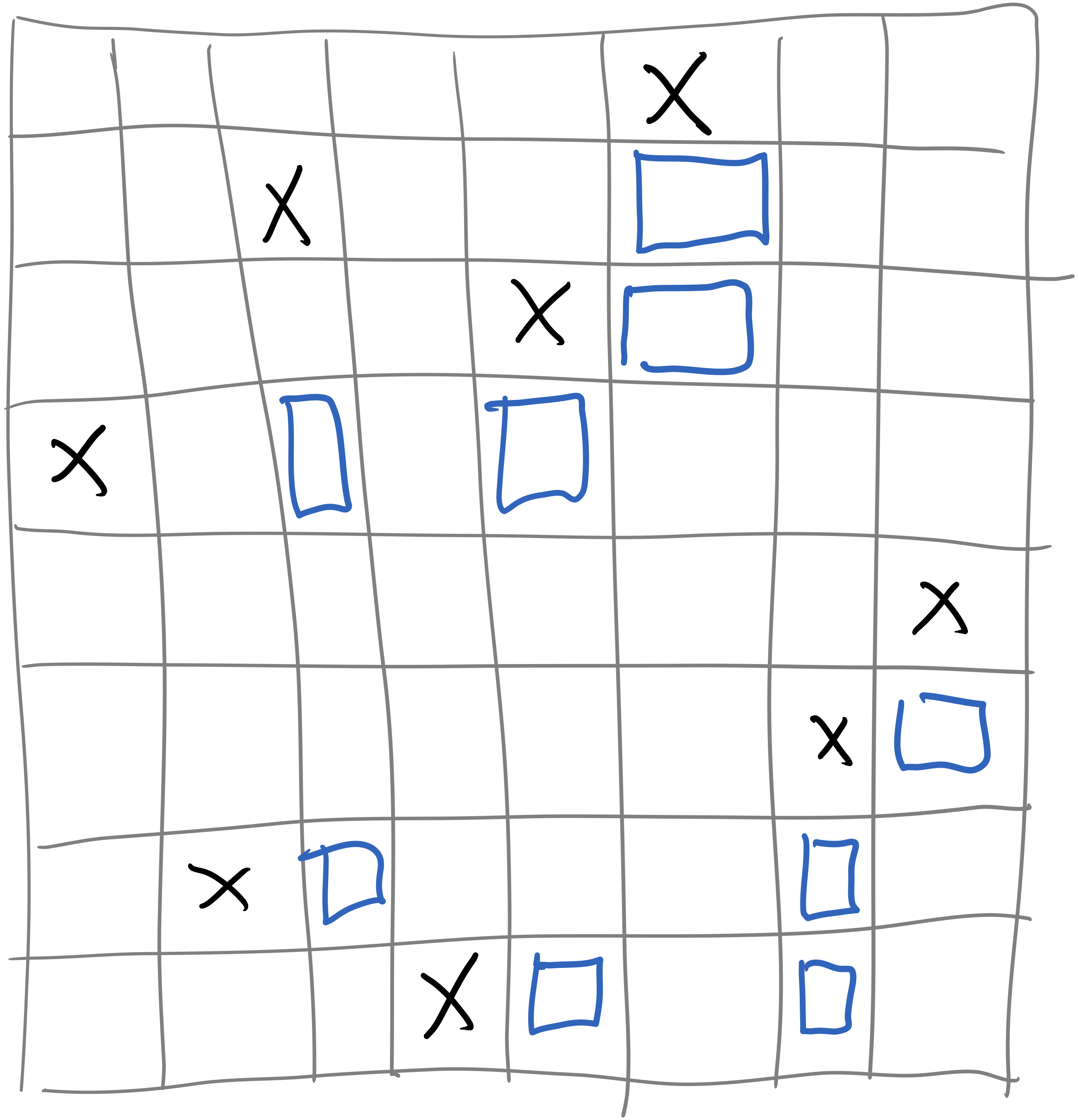
Right only

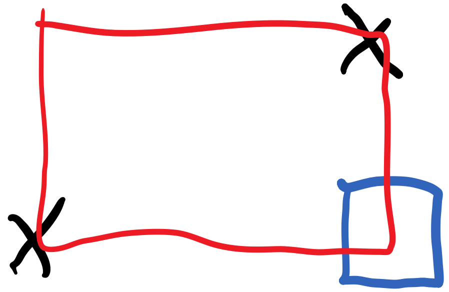

Don't add this
Right only

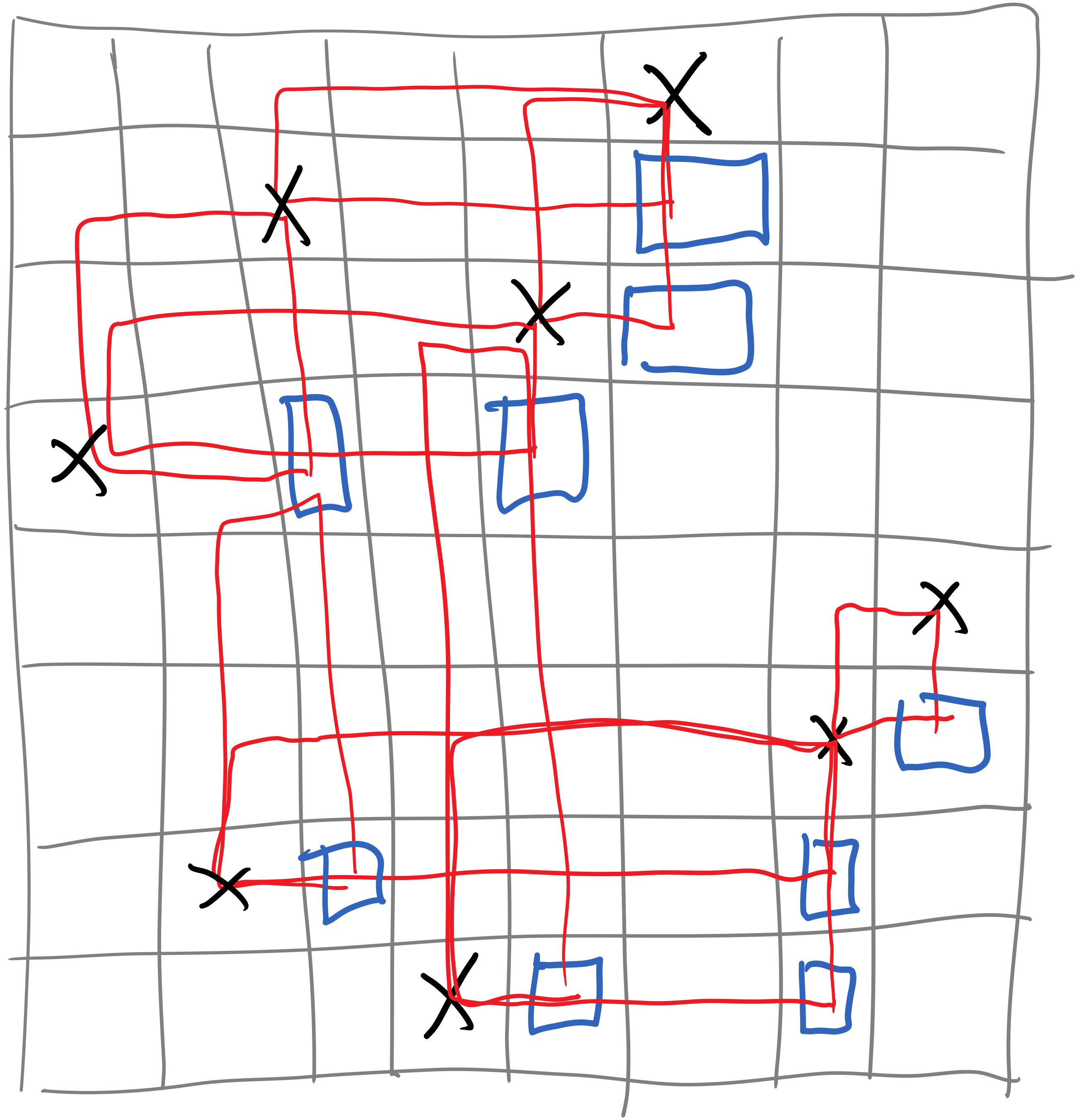


Right only

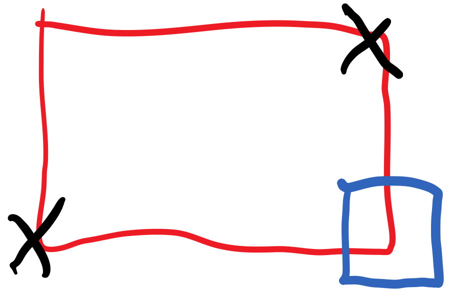
Draw Rectangles

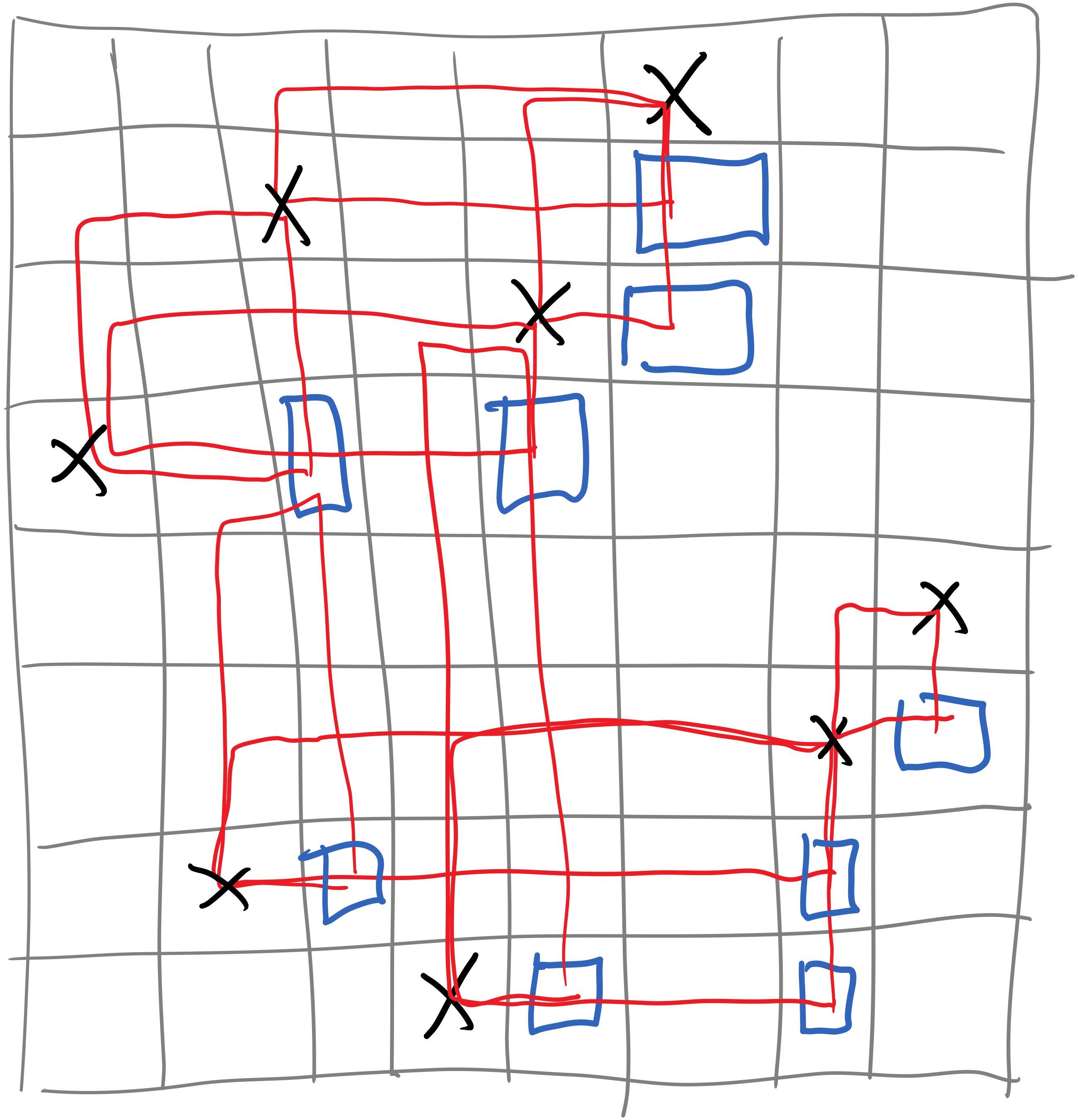




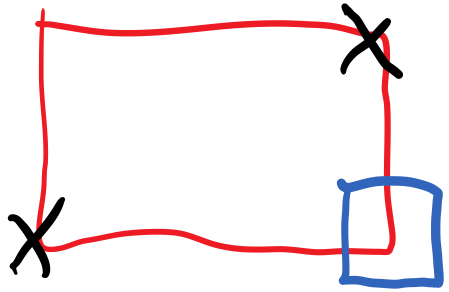


Draw Rectangles

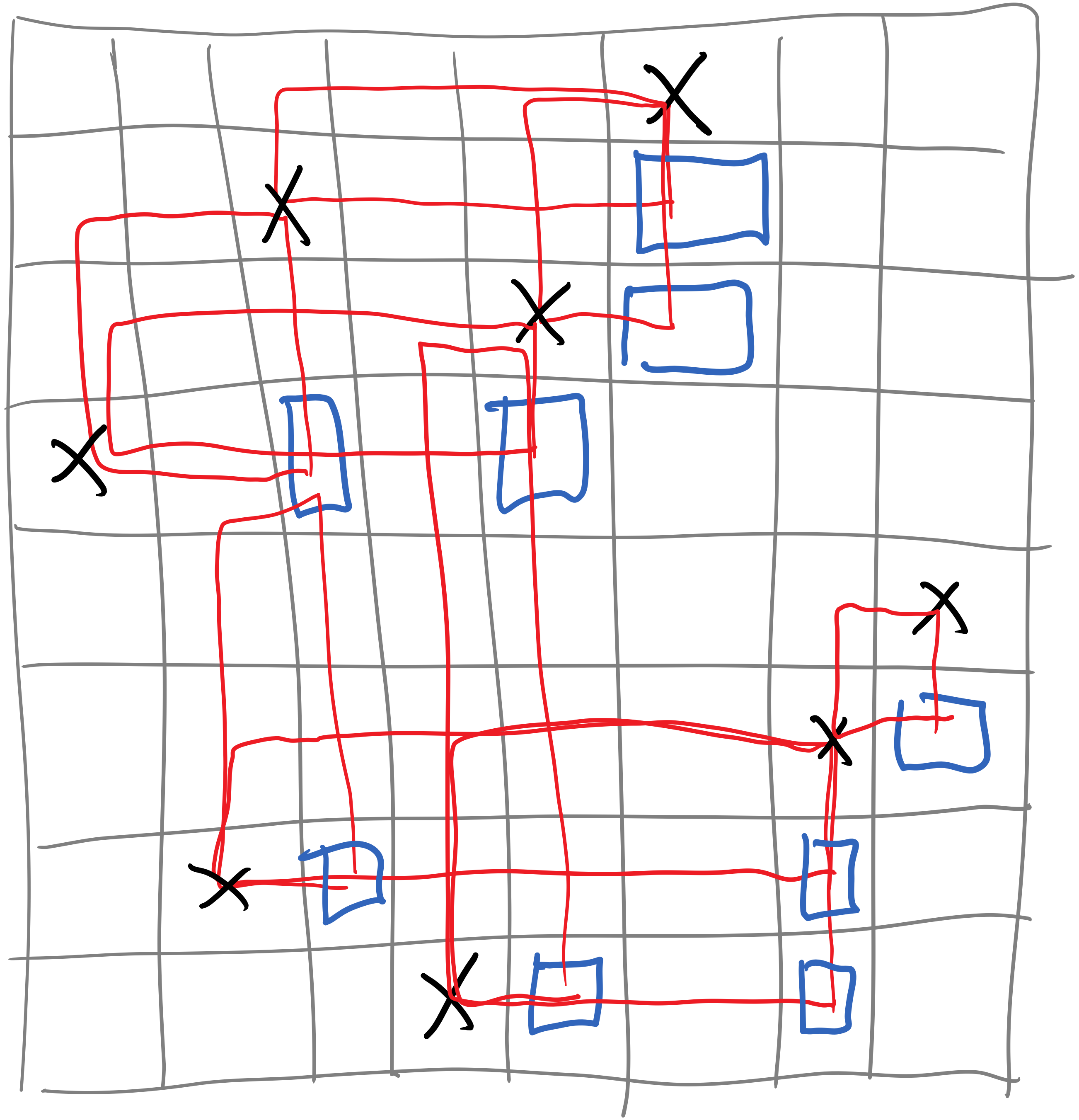




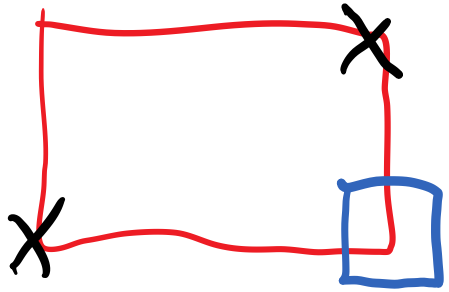
Draw Rectangles



They are Independent!

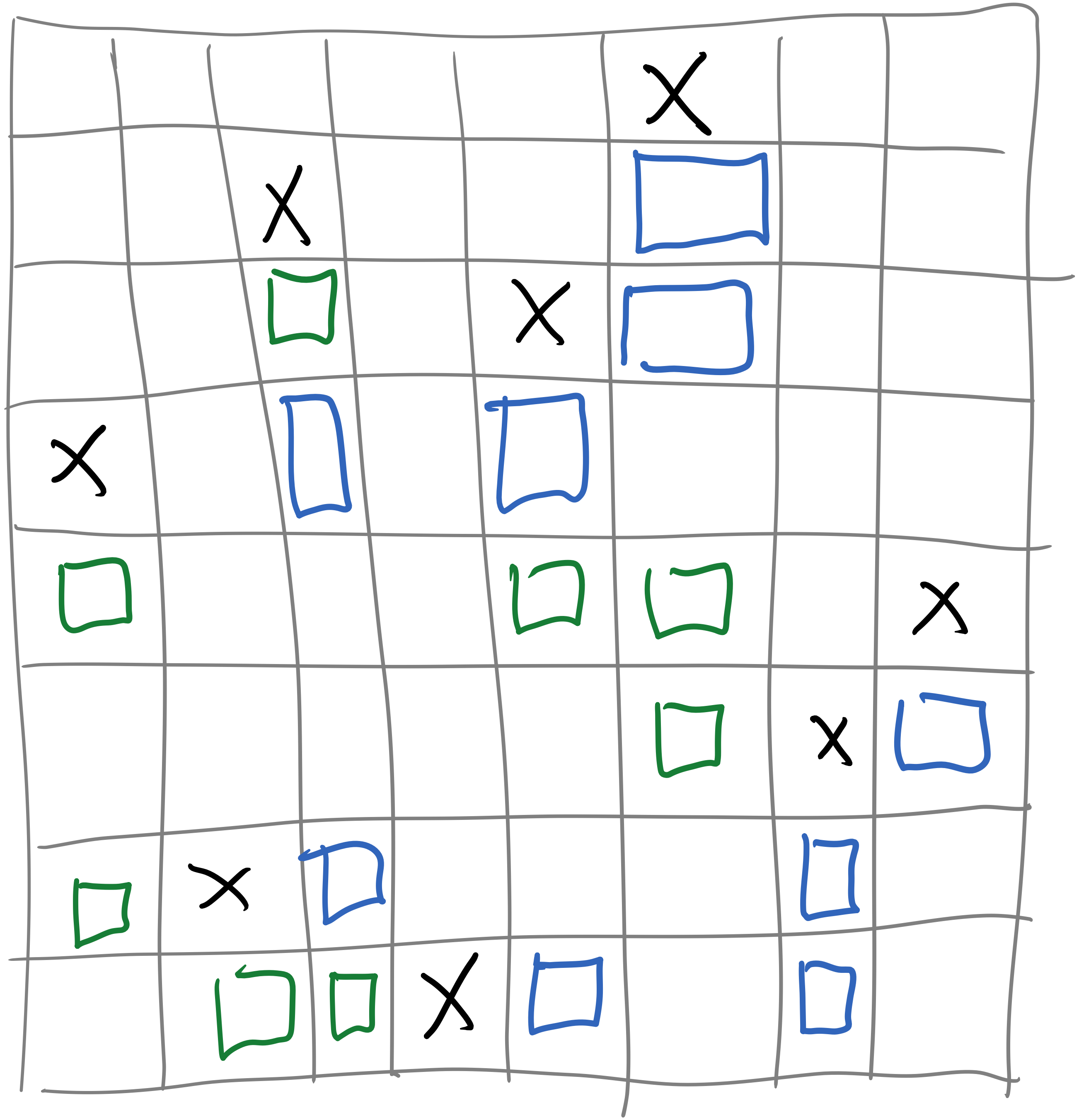


Draw Rectangles



They are Independent!

Thus # □ is a lower bound to make this ASS!



Left only

Right only

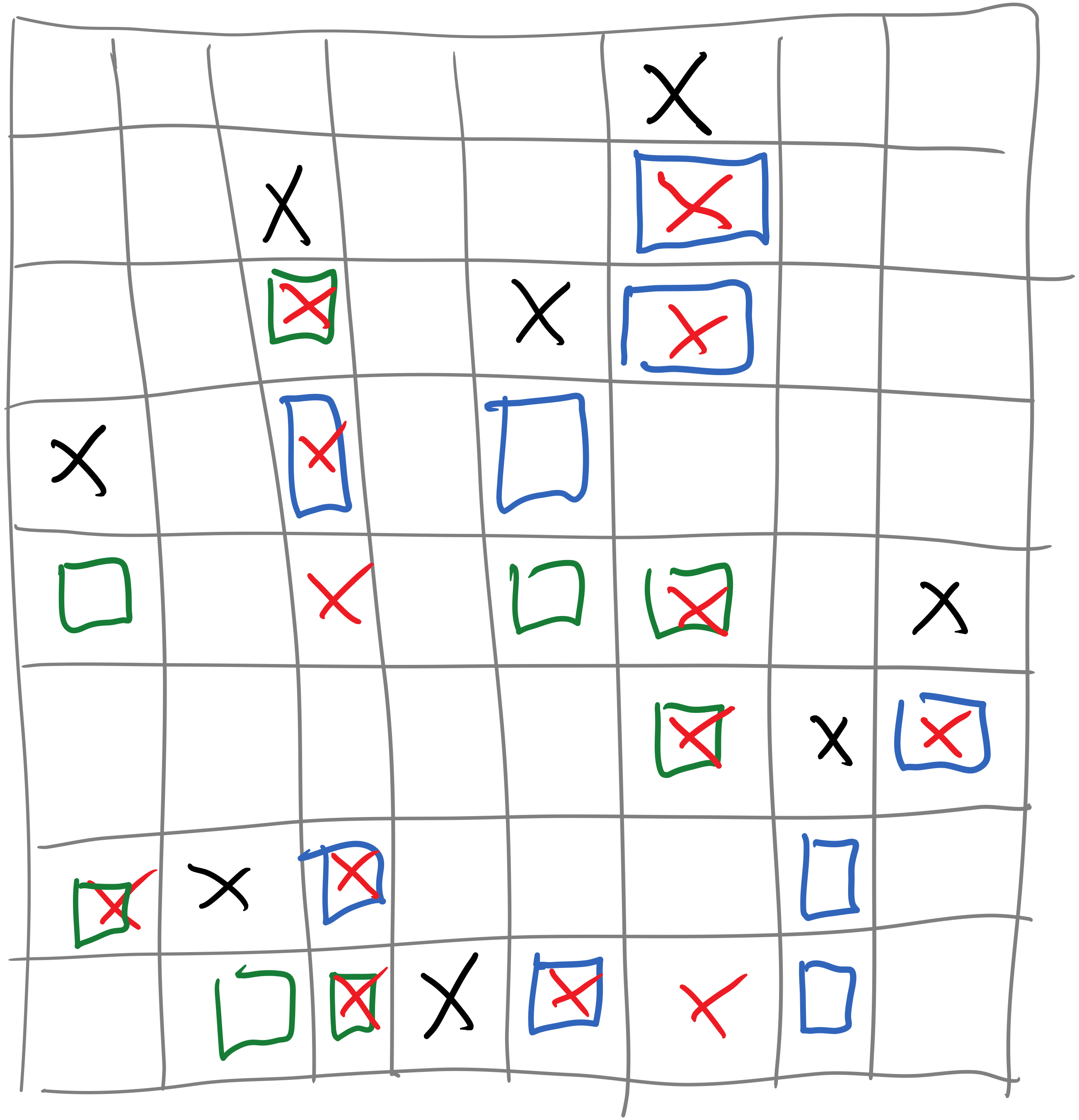
Number of □ □

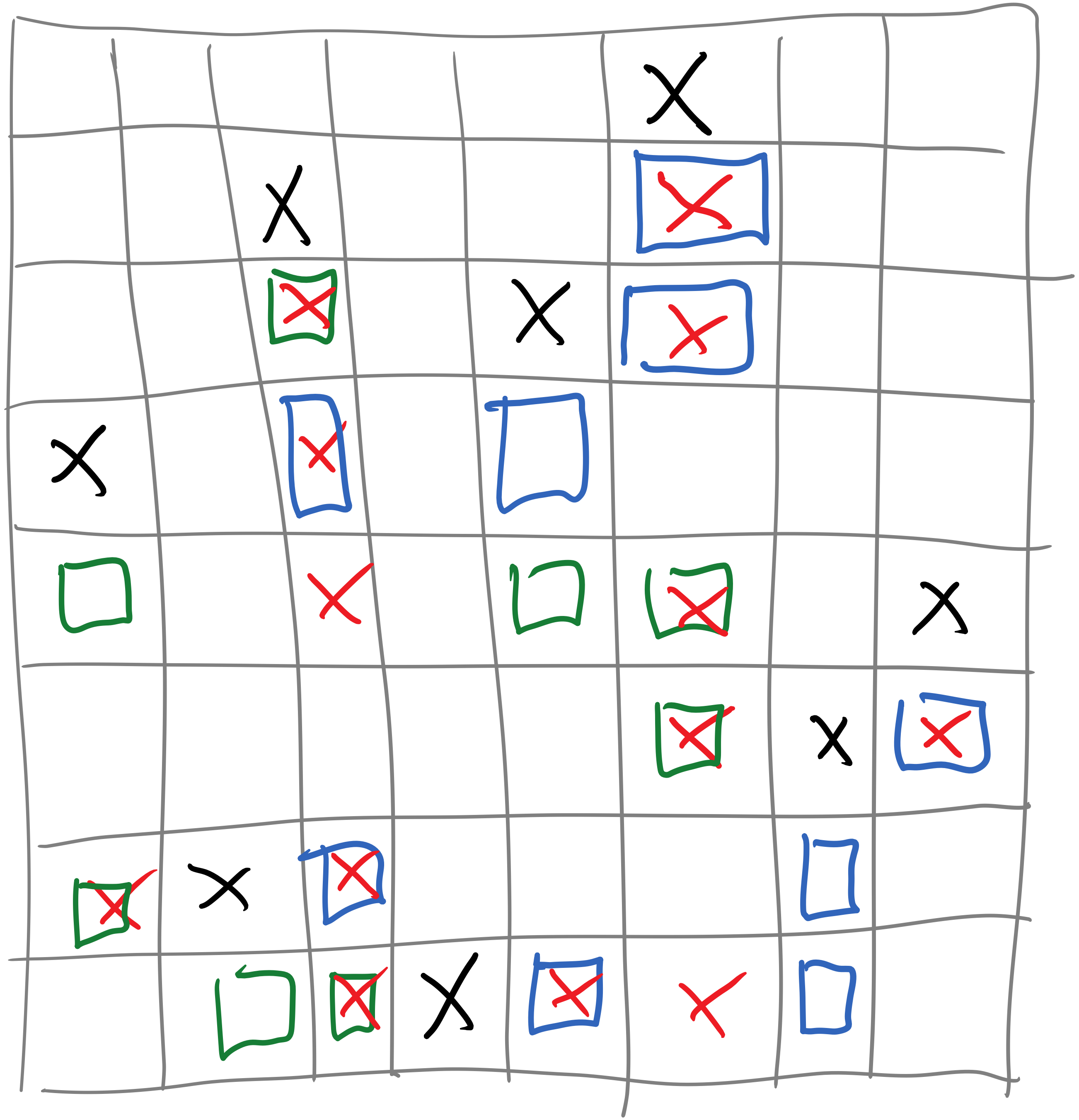
is less than

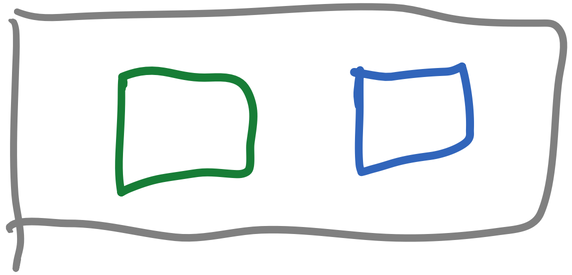
the minimum

X needed

to make it ASS



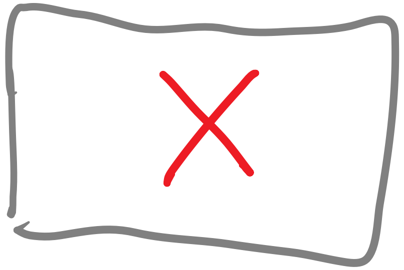


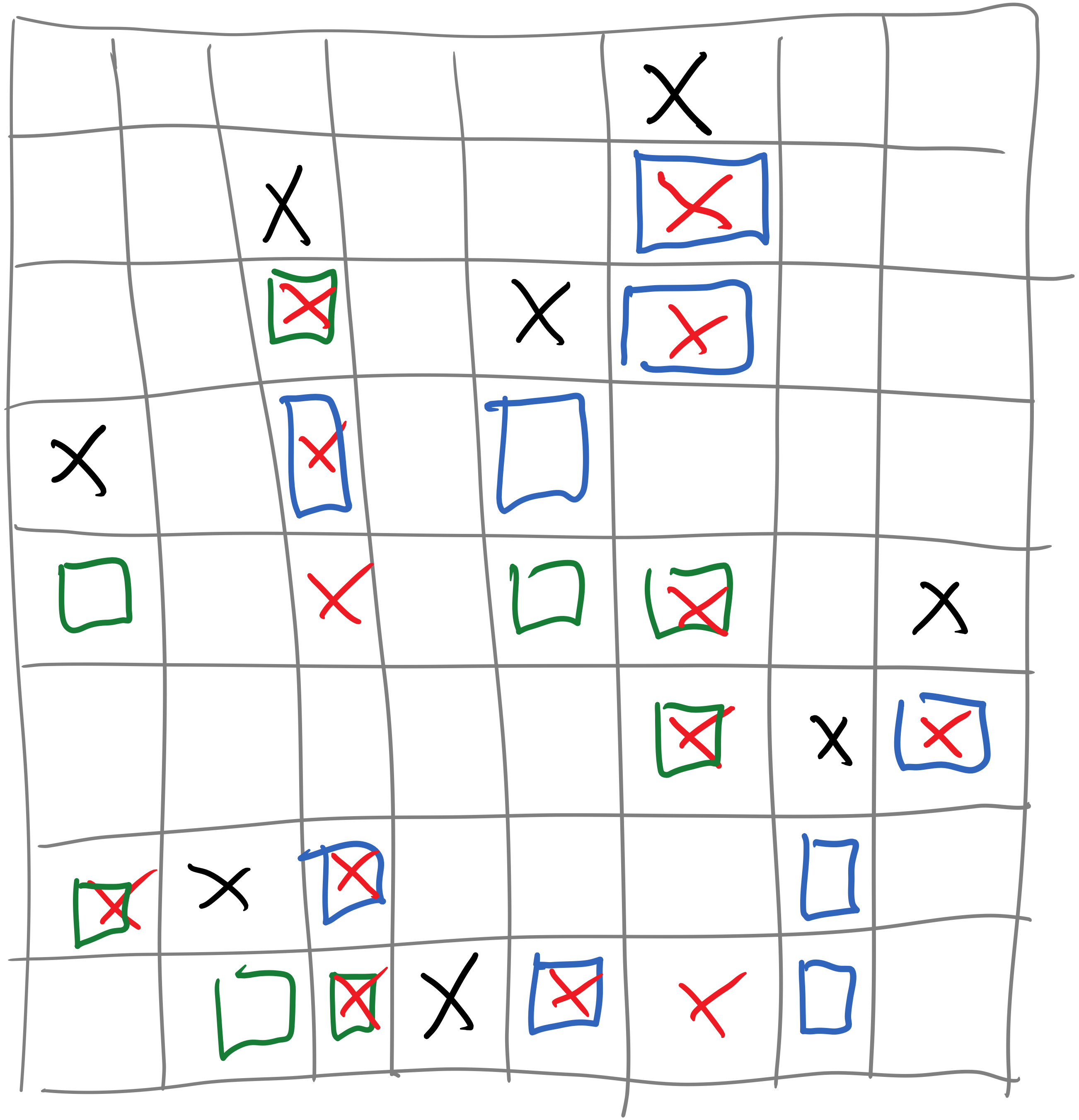


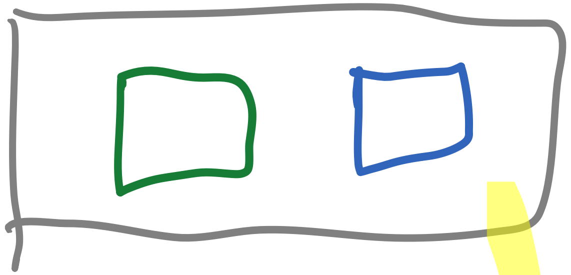
is less than

Minimum X to make it ASS

is less than



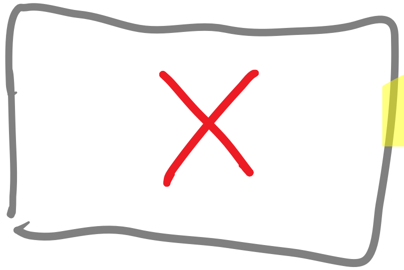




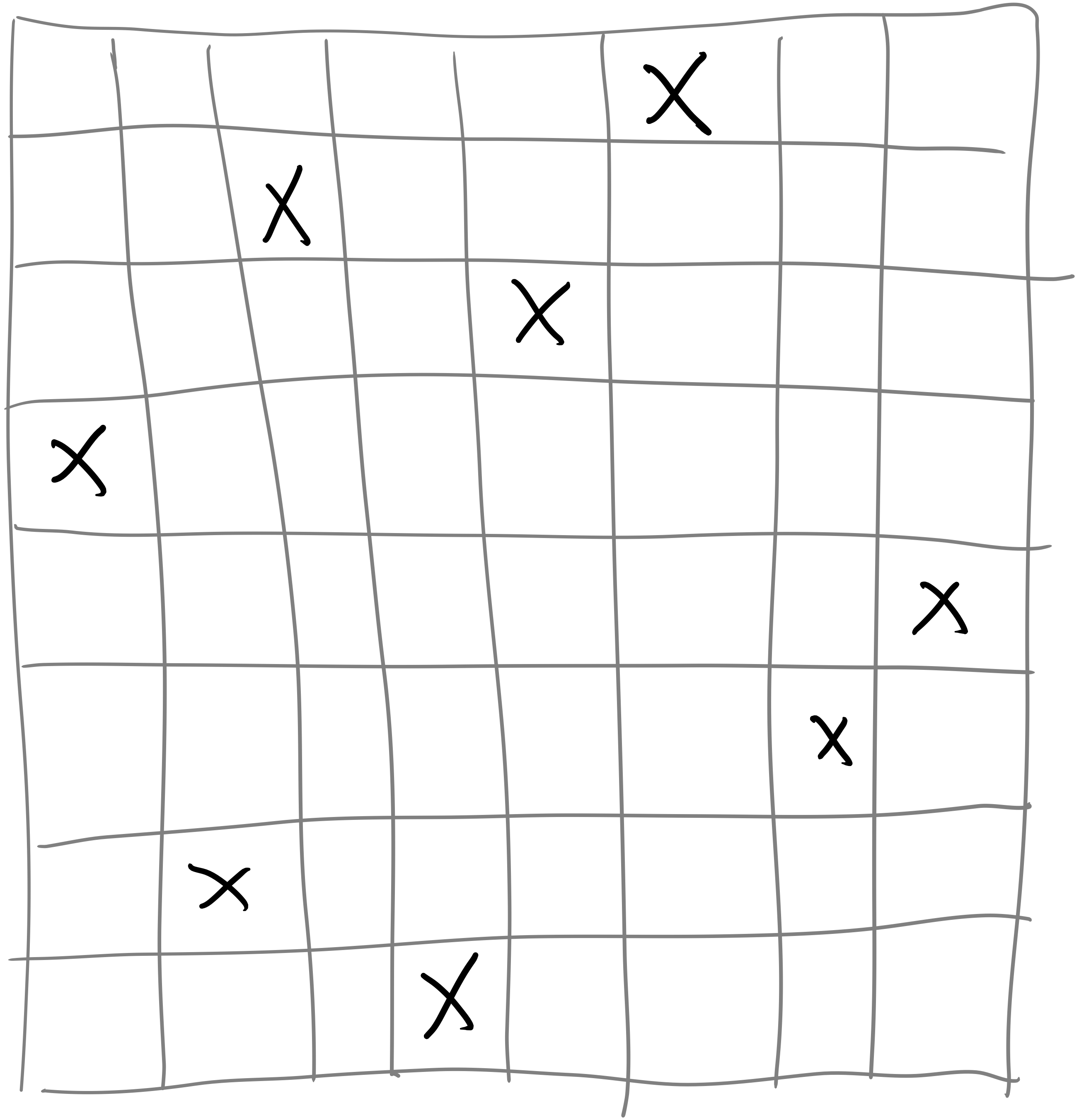
is less than

Minimum X to make it ASS

is less than

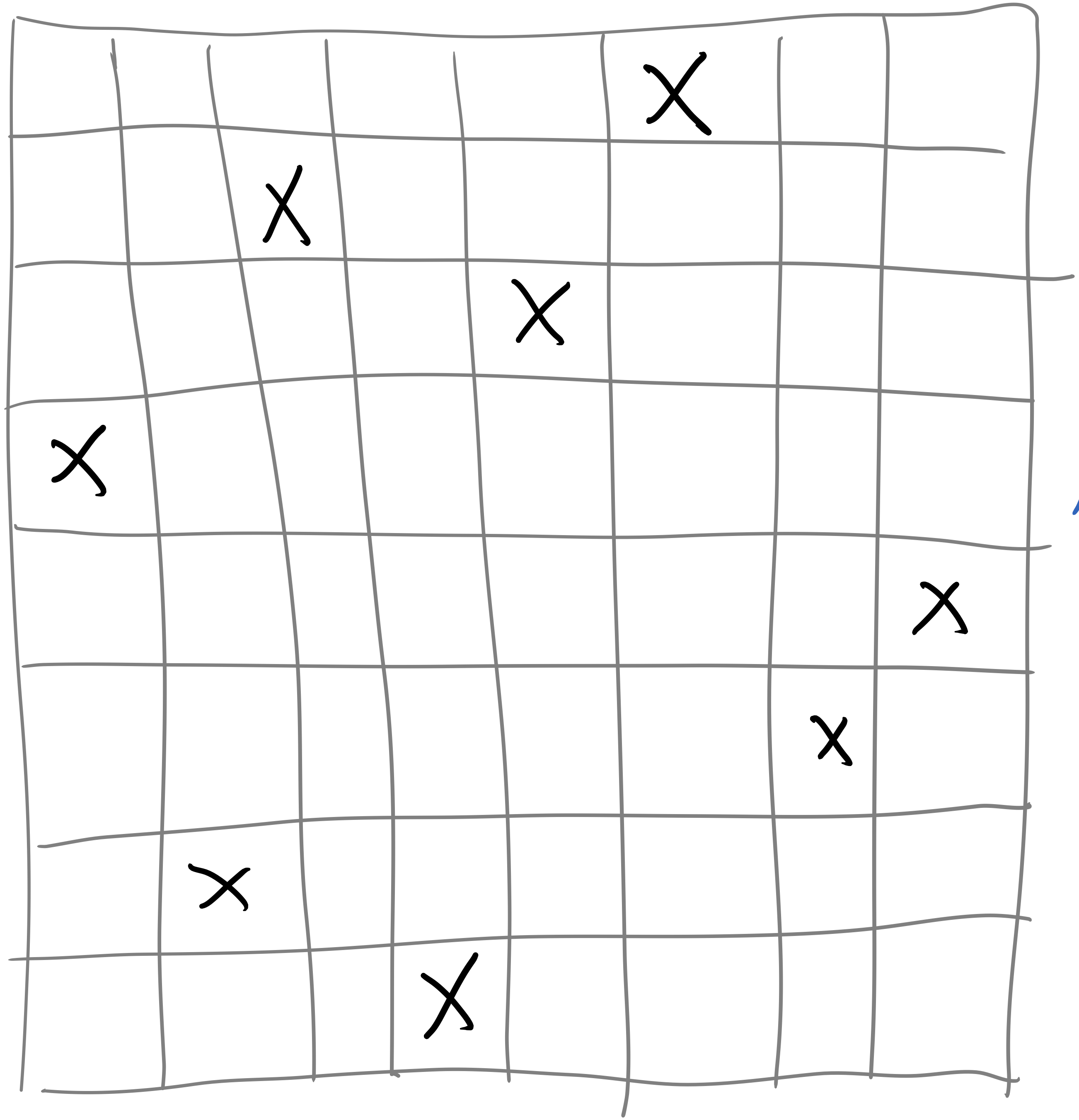


Related?



Questions

Find minimal ASS superset: Per NIC

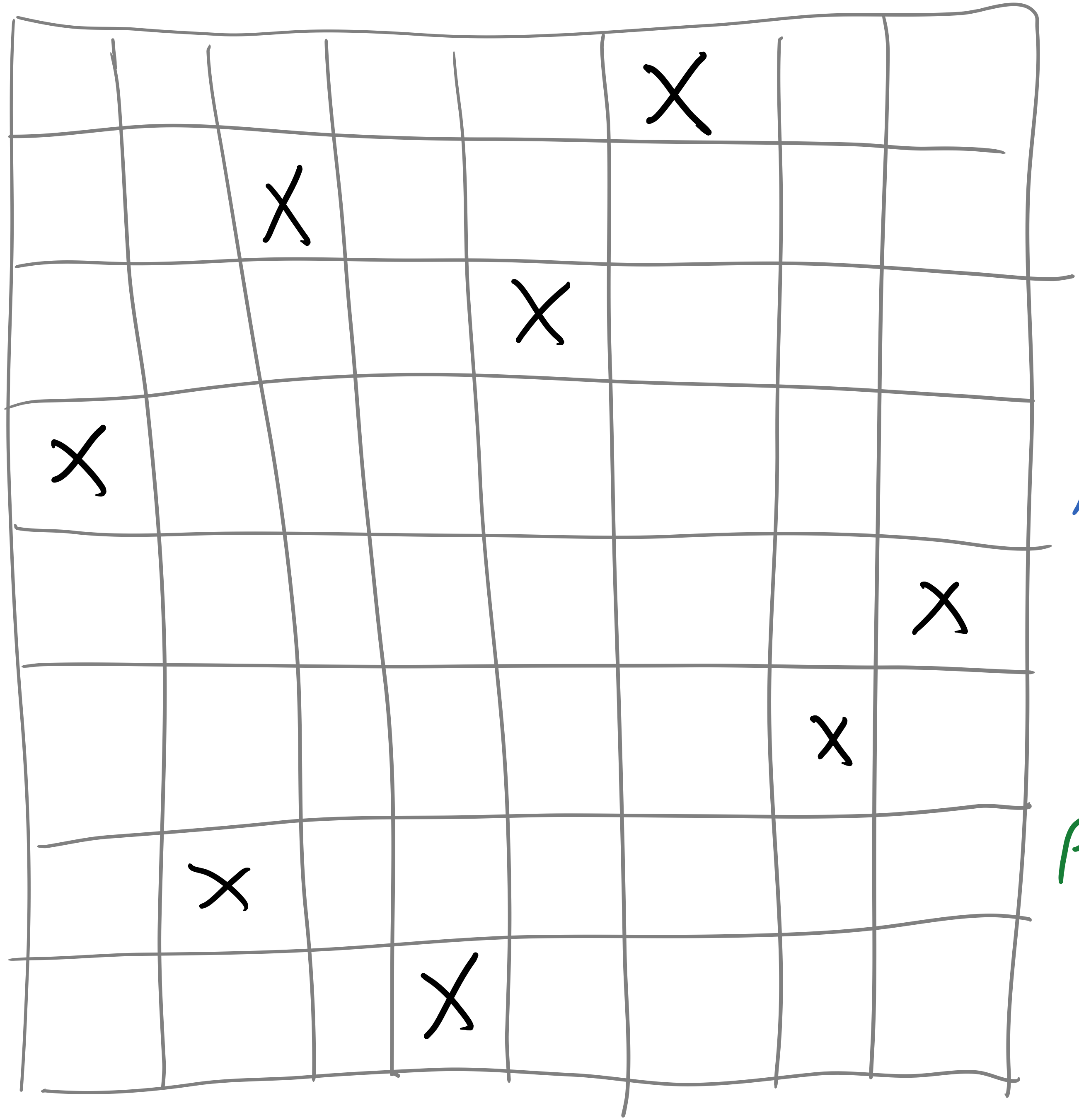


Questions

Find minimal ASS superset: Per NPC

Approximate minimal ASS superset.

O(log log n)-approx known but nothing better

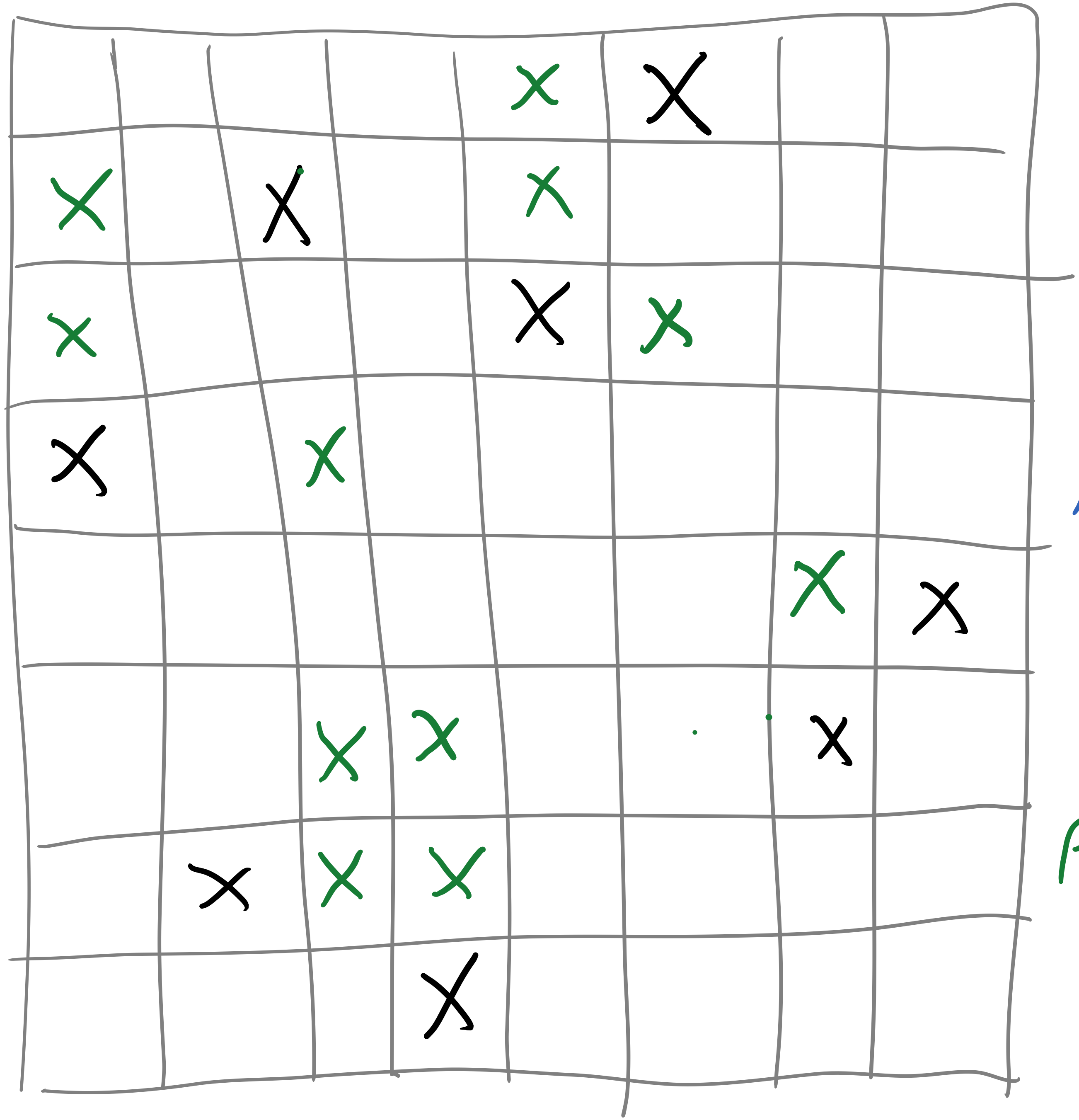


Questions

Find minimal ASS superset: P or NPC

Approximate minimal ASS superset.
O(loglog n)-approx known but nothing better

Problem is "secondary effects"

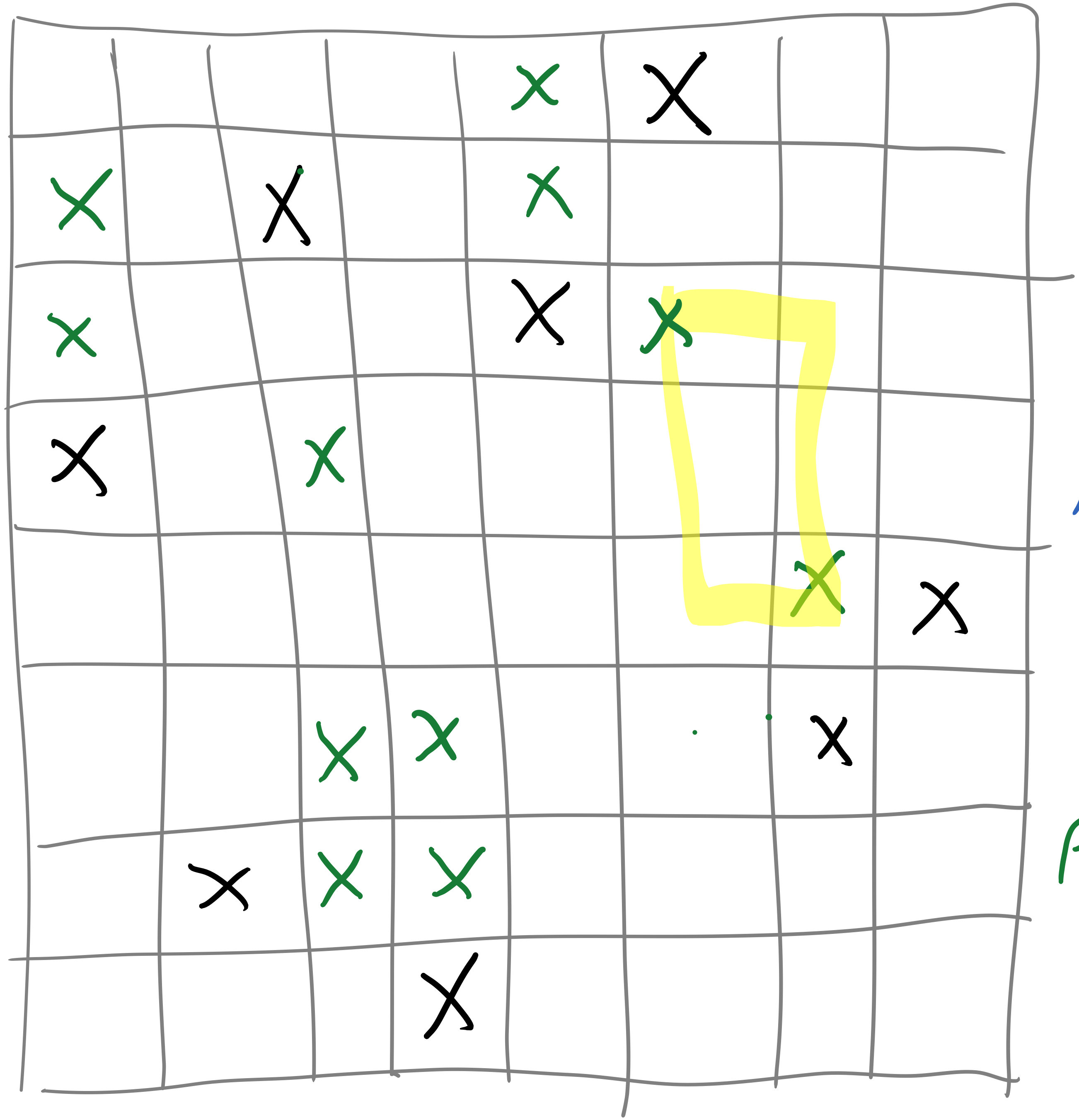


Questions

Find minimal Ass superset: Per NPC

Approximate minimal Ass superset.
O(loglog n)-approx known but nothing better

Problem is "secondary effects"



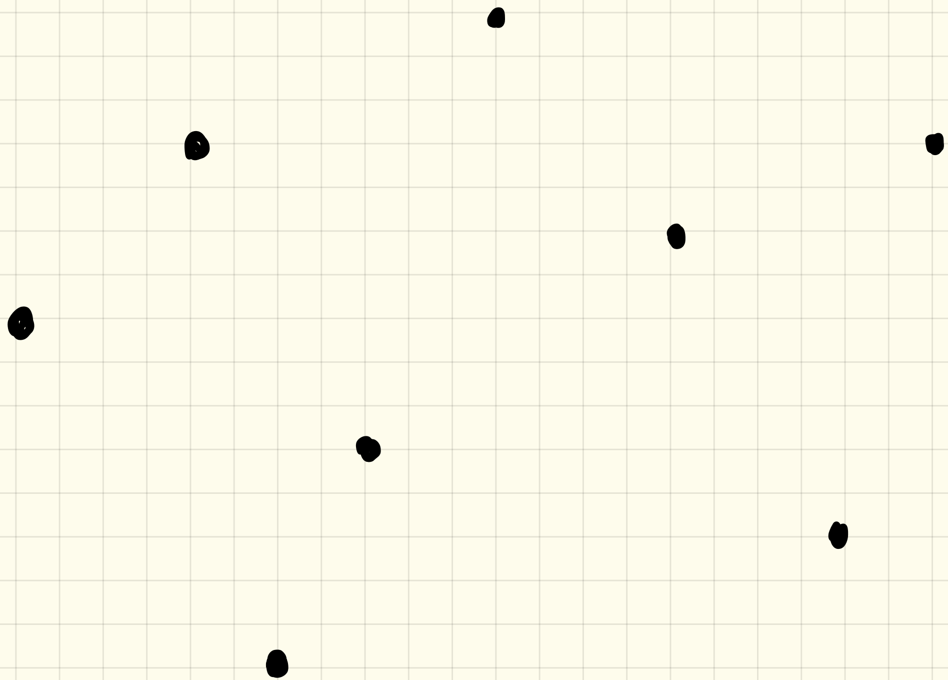
Questions

Find minimal Ass superset: P or NPC

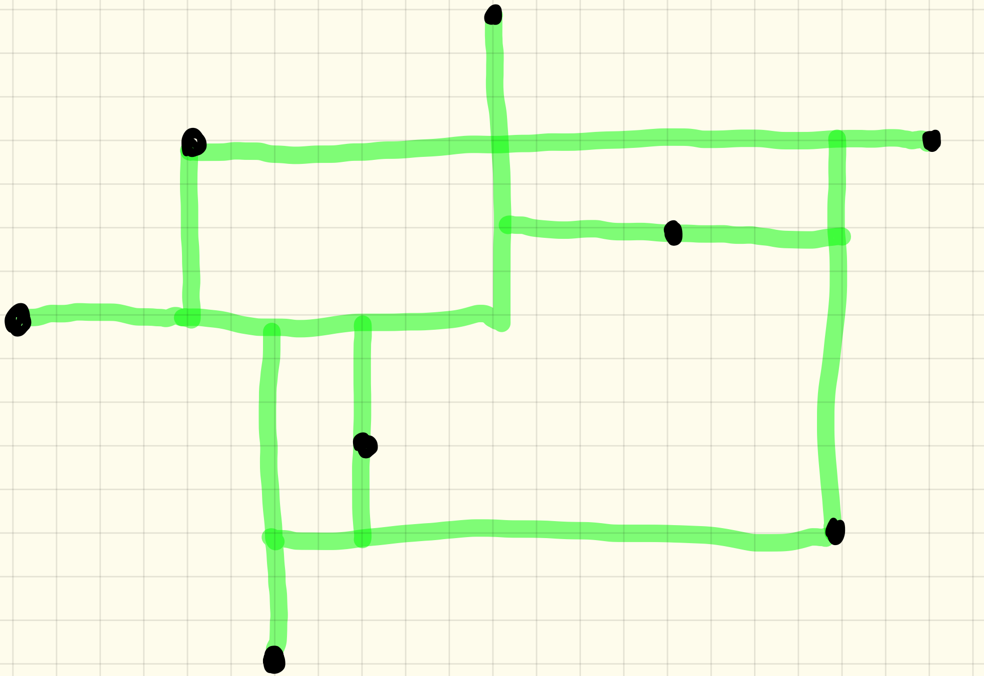
Approximate minimal Ass superset.
$O(\log\log n)$-approx known but nothing better

Problem is "secondary effects"

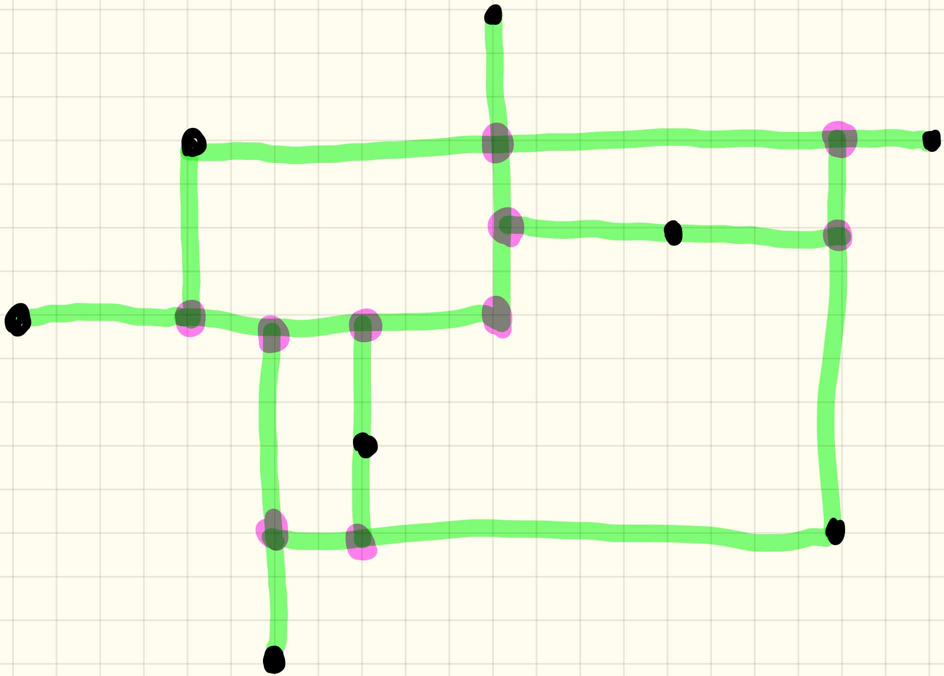
Seems Similar To Minimum Manhattan Network



Seems Similar To Minimum Manhattan Network

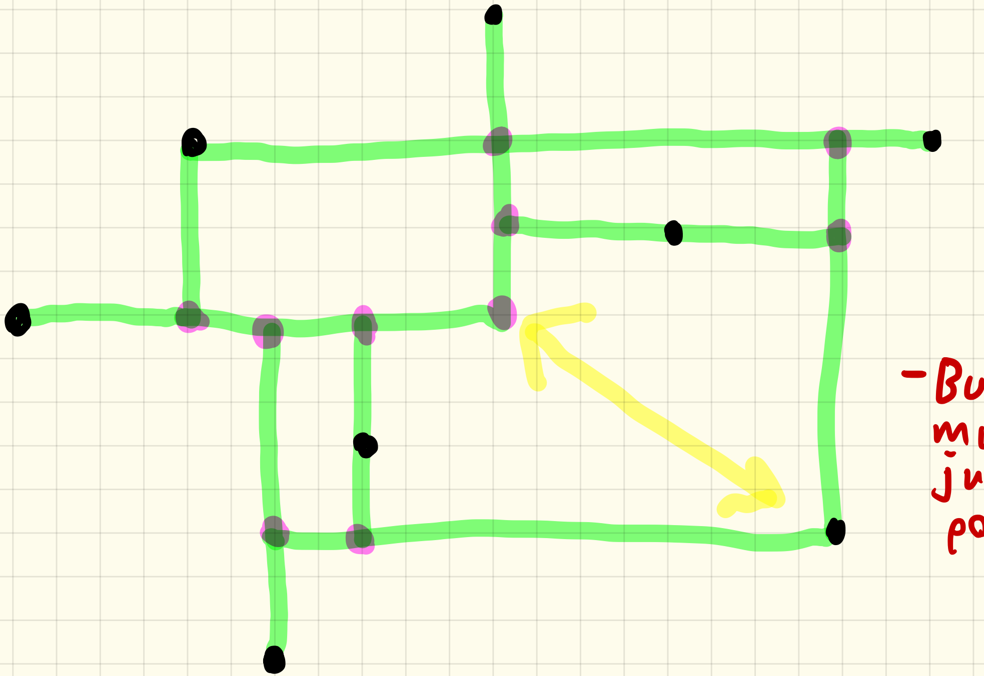


Seems Similar To Minimum Manhattan Network



- Measure of interest is # of added junctions

Seems Similar To Minimum Manhattan Network



- Measure of interest is # of added junctions

- But network must treat junctions as points

PART II

Cache Oblivious

+

Persistence

PART II

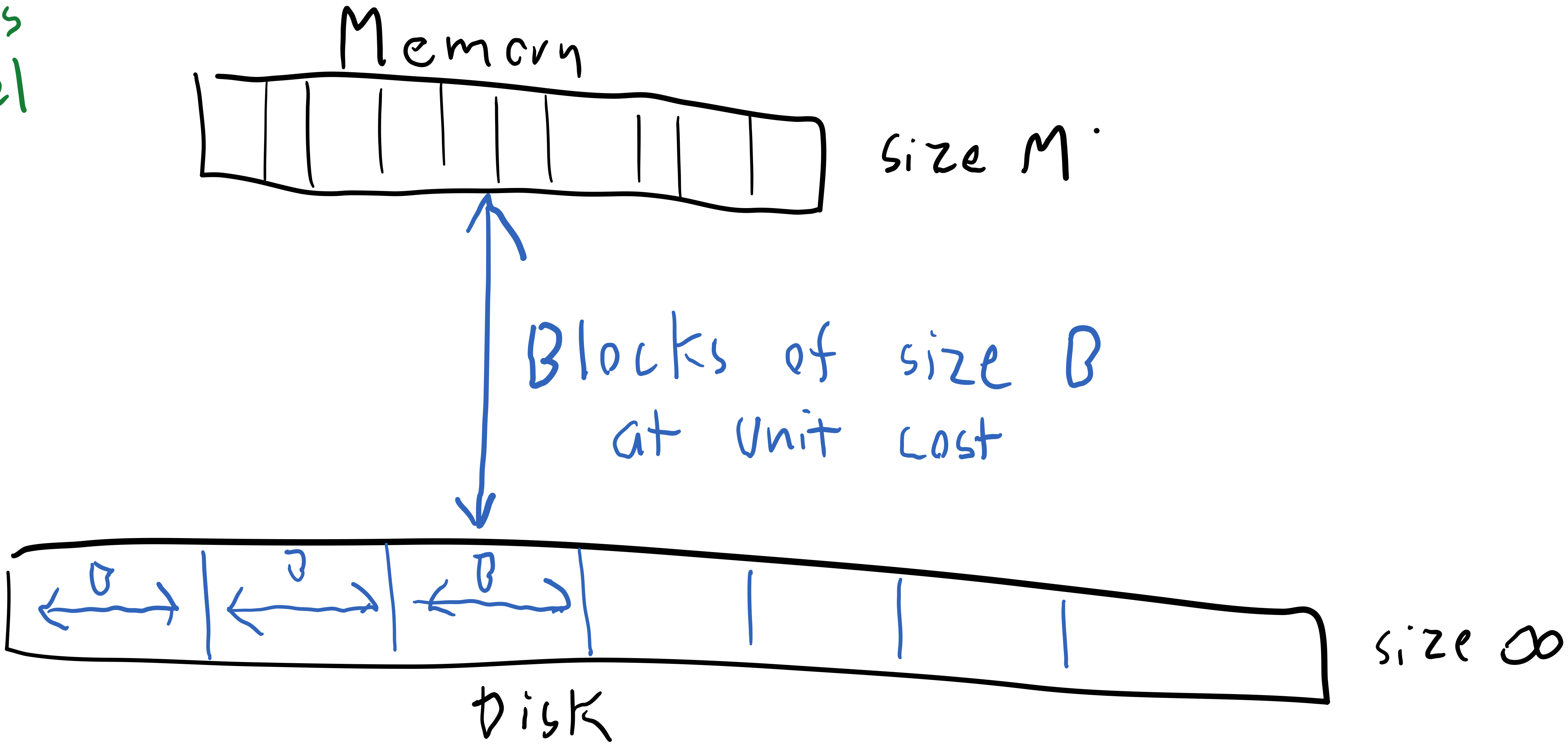
Cache Oblivious

+

Persistence

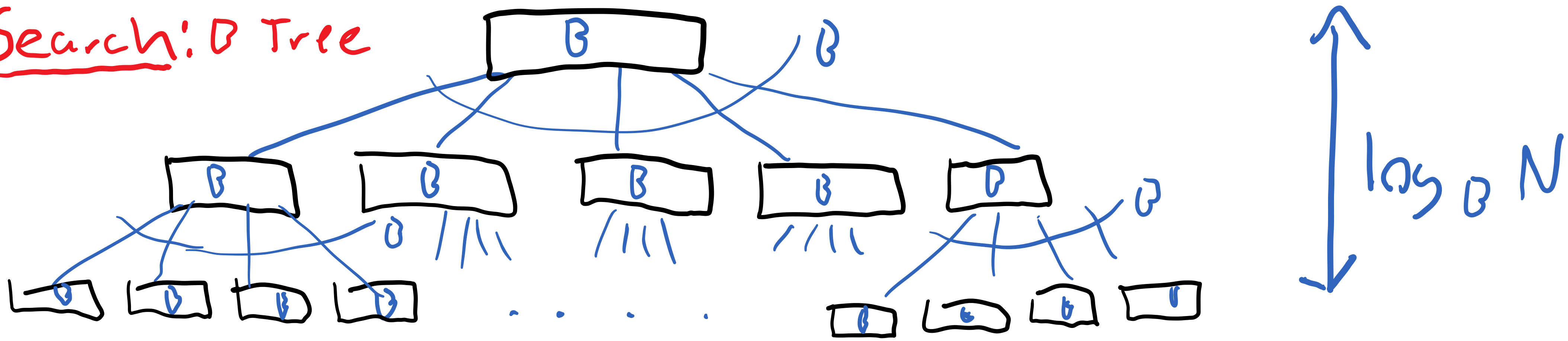
with P. Davoodi, O. Ozkan, J. Fineman

Disk
Access
Model



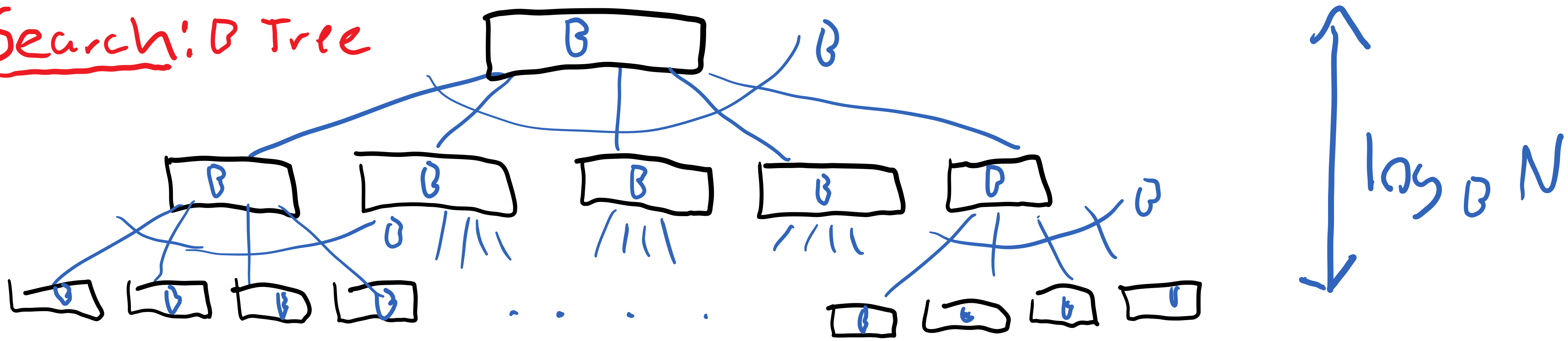
Two Examples

Search: B Tree

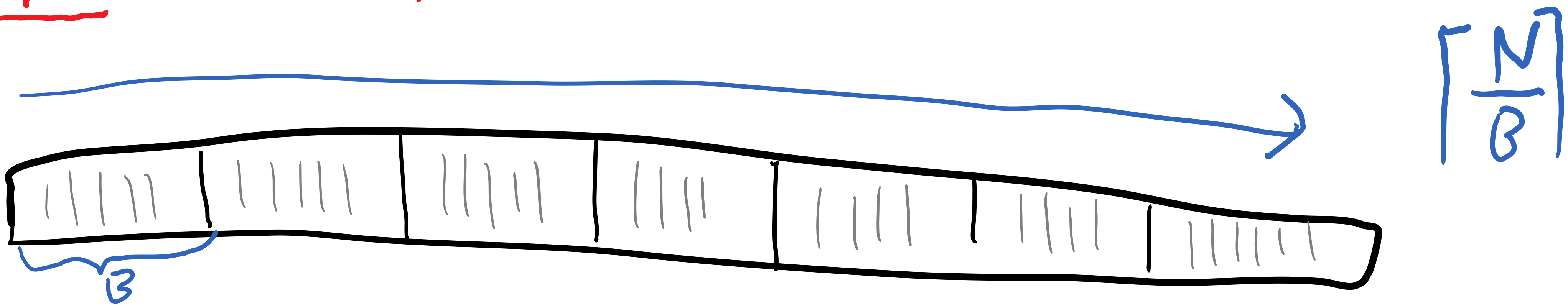


Two Examples

Search: B Tree

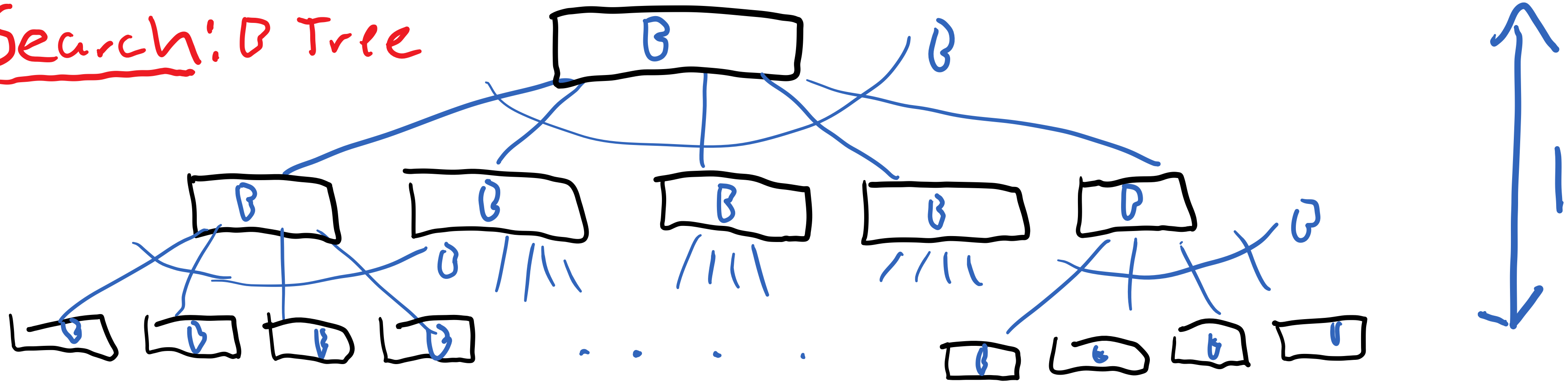


Mini: For loop



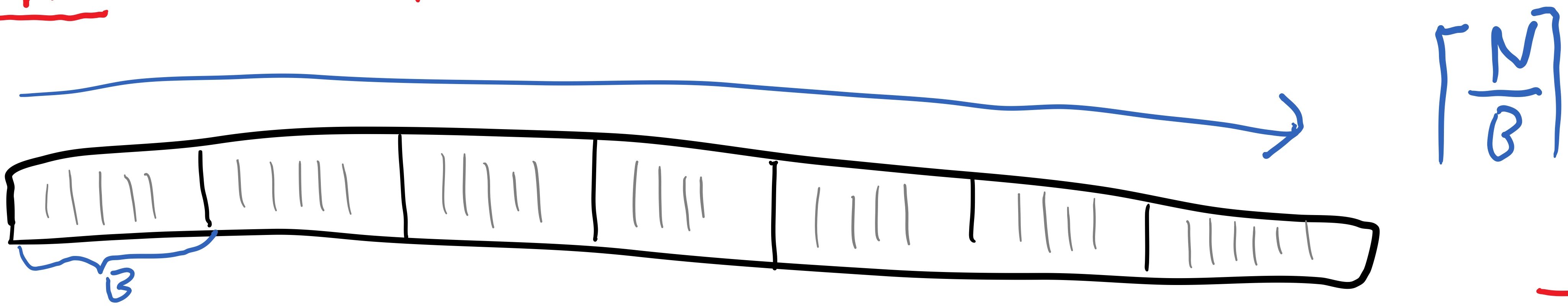
Two Examples

Search: B Tree

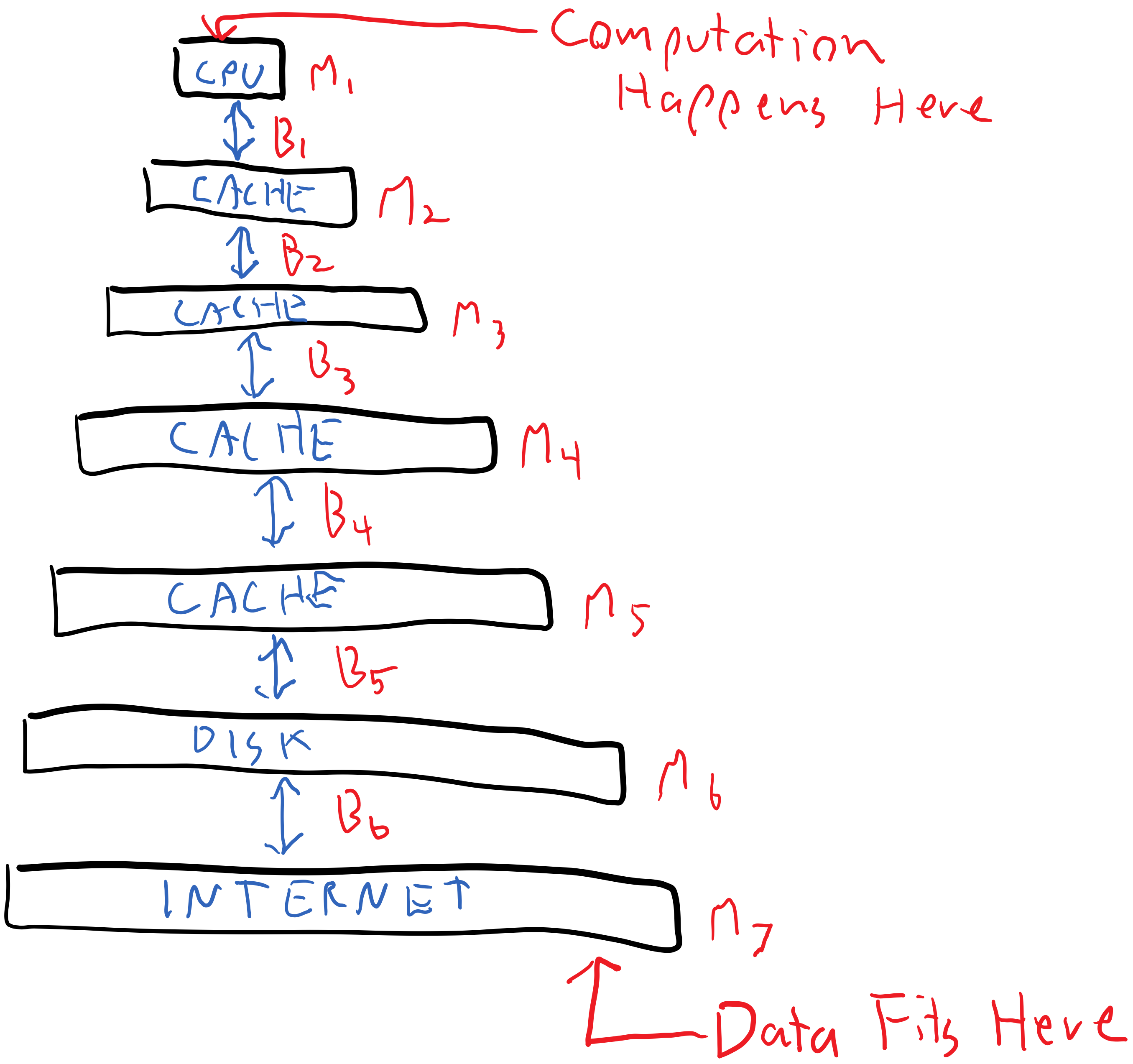


Needs to know B

Mini: For loop

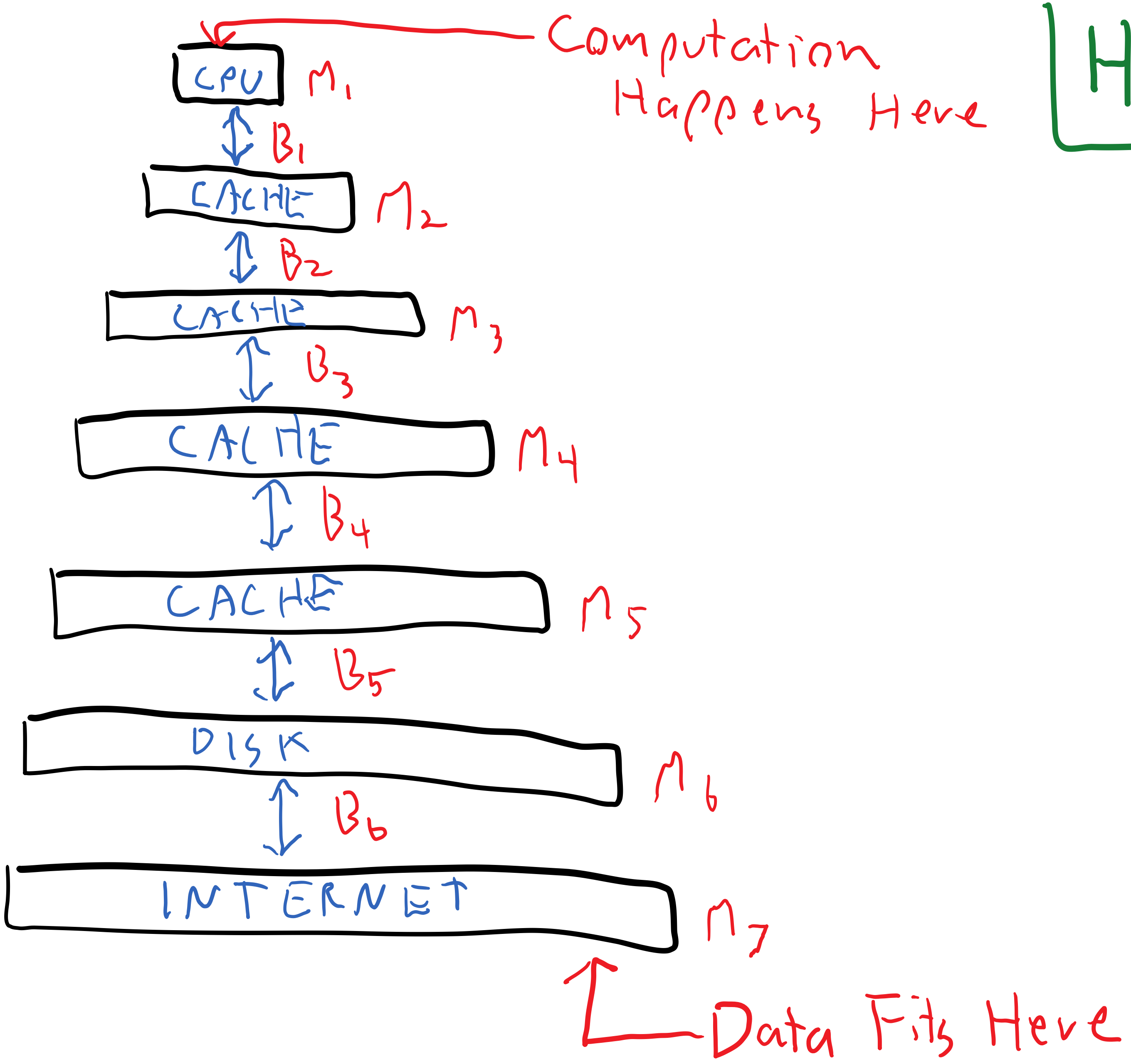


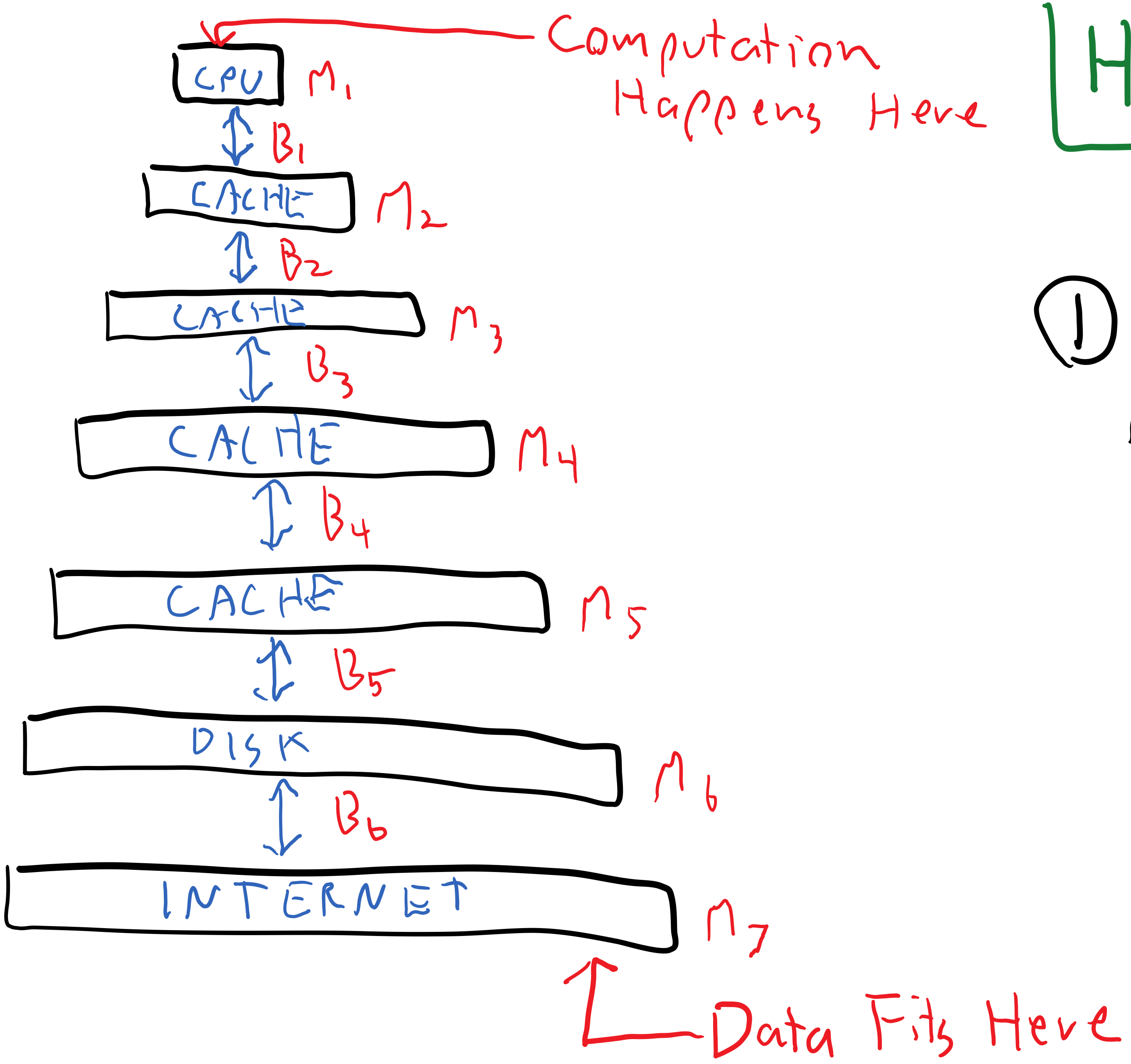
Does not need to know B



Computation Happens Here
CPU
M_1
B_1
CACHE
M_2
B_2
CACHE
M_3
B_3
CACHE
M_4
B_4
CACHE
M_5
B_5
DISK
M_6
B_6
INTERNET
M_7
Data Fits Here

How do we deal with this?

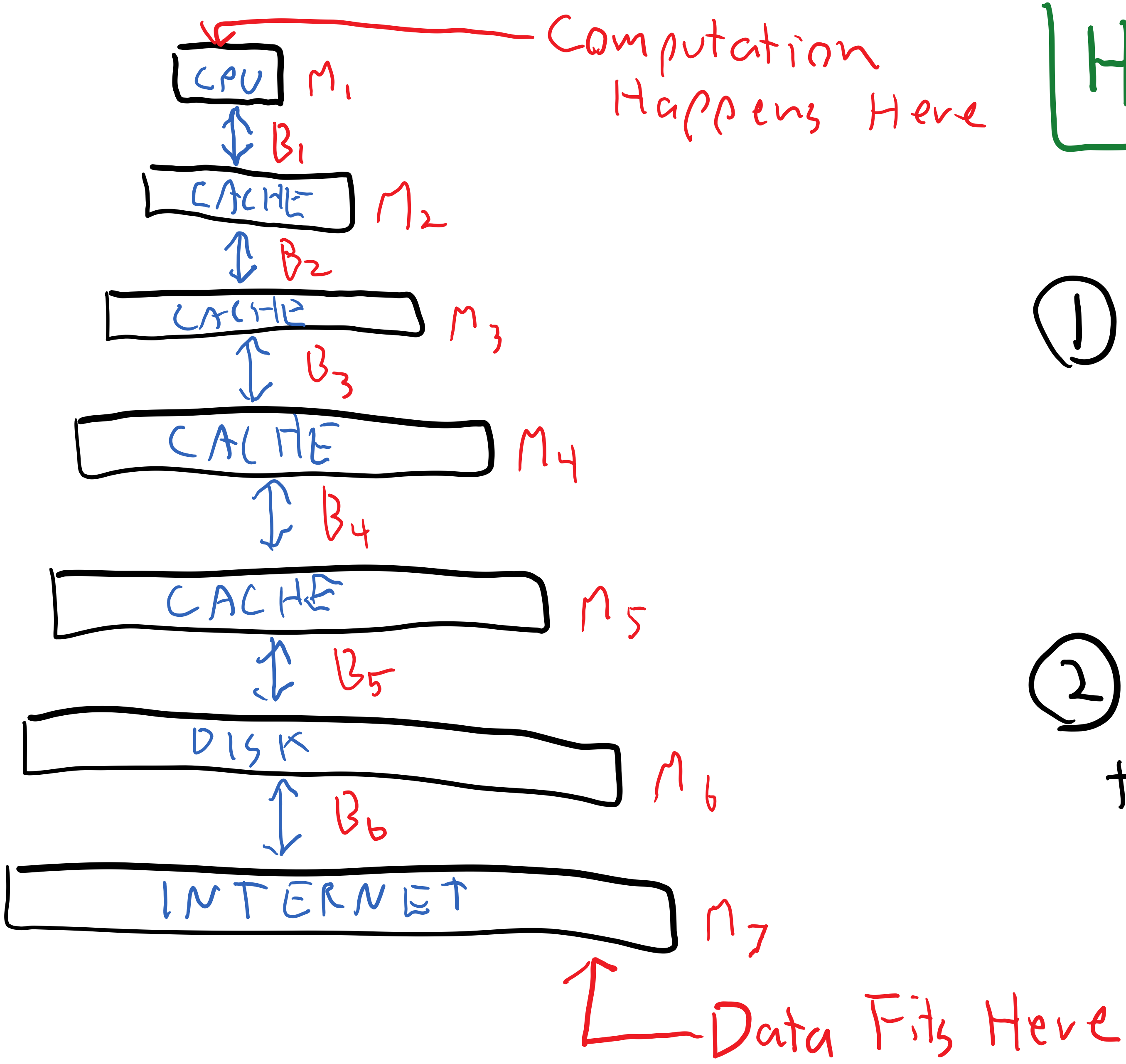




How do we deal with this?

1. Make alg that uses $M_1, M_2 \ldots M_7 ; B_1, B_2 \ldots B_7$

   Yuck



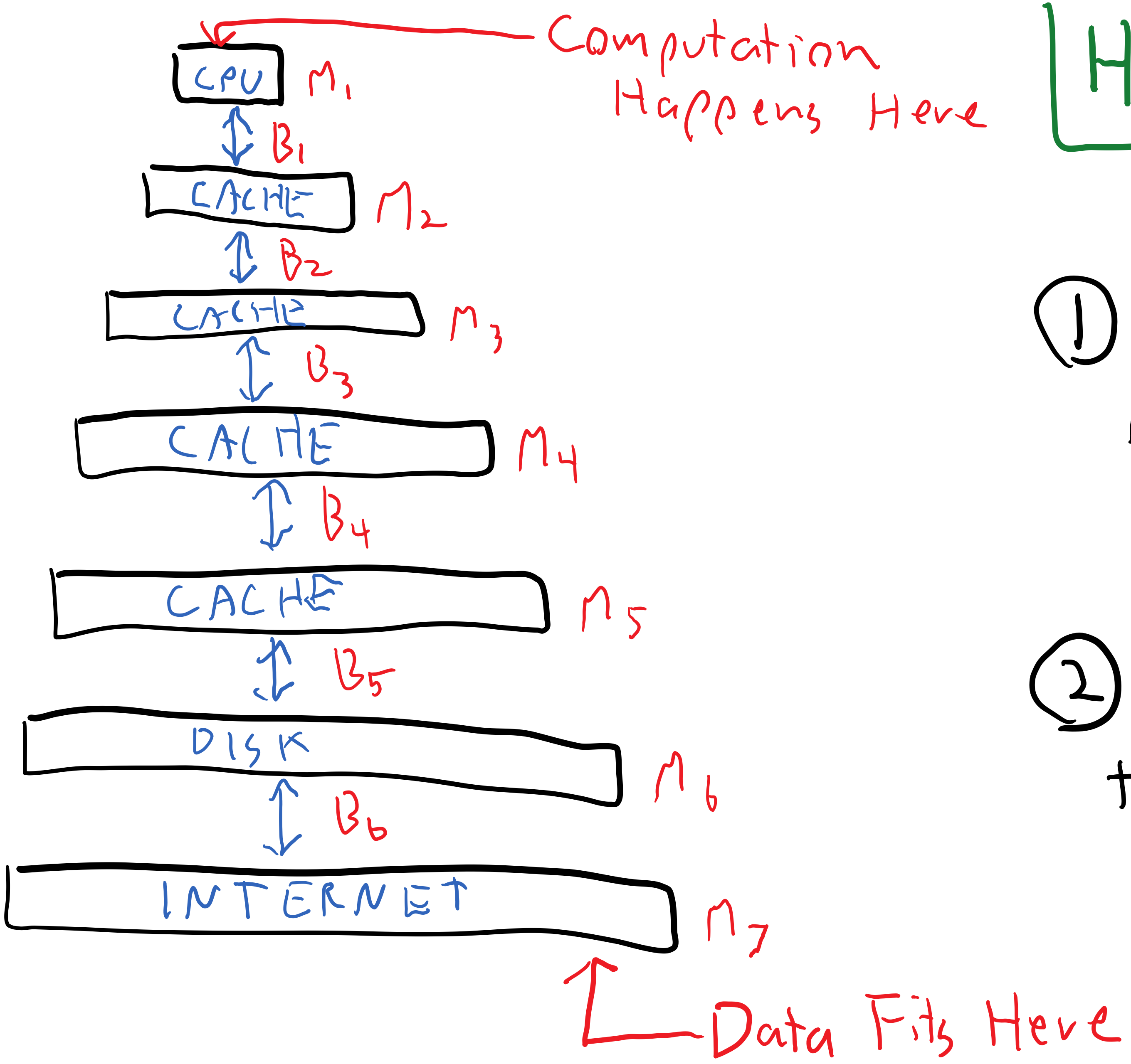
How do we deal with this?

1. Make alg that uses $M_1, M_2 \ldots M_7 ; B_1, B_2 \ldots B_7$

   Yuck

2. Use 2-level (DAM) alg that does not know M, B

   Cache Oblivious



How do we deal with this?

1. Make alg that uses $M_1, M_2 \dots M_7 ; B_1, B_2 \dots B_7$

   Yuck

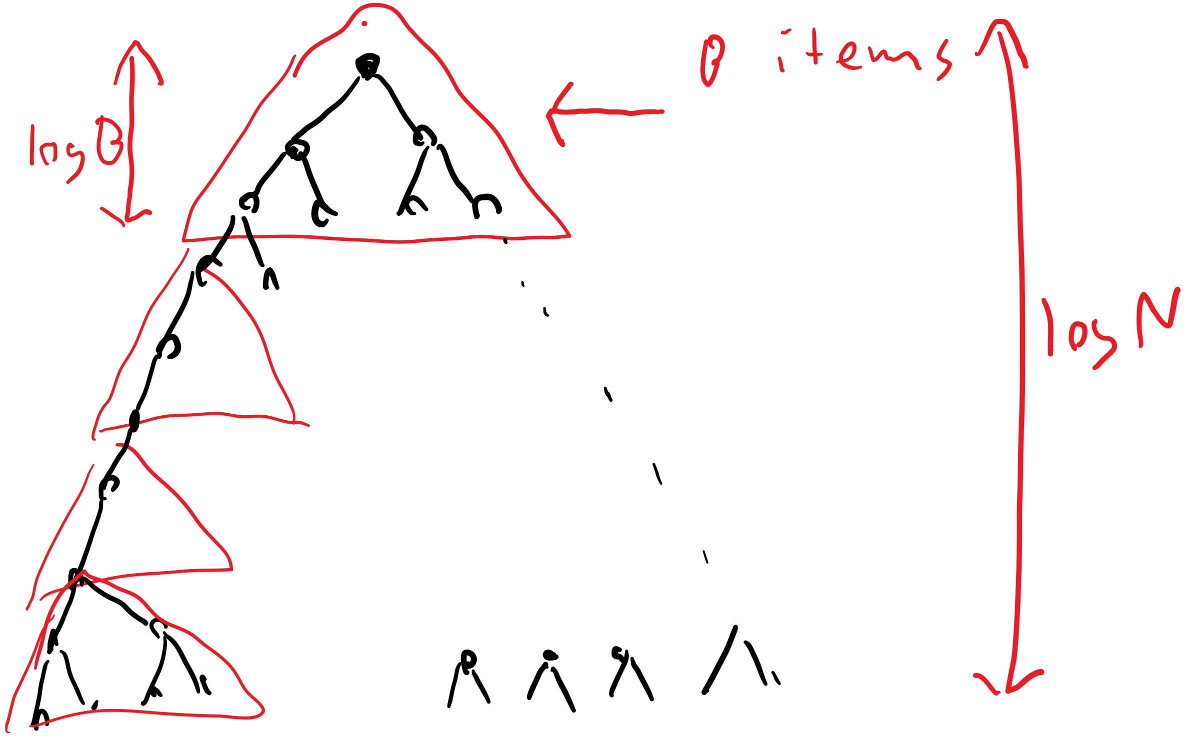
2. Use 2-level (DAM) alg that does not know M, B

   Cache Oblivious

E.G. Scan works well

How Do We Make a β tree with no β?

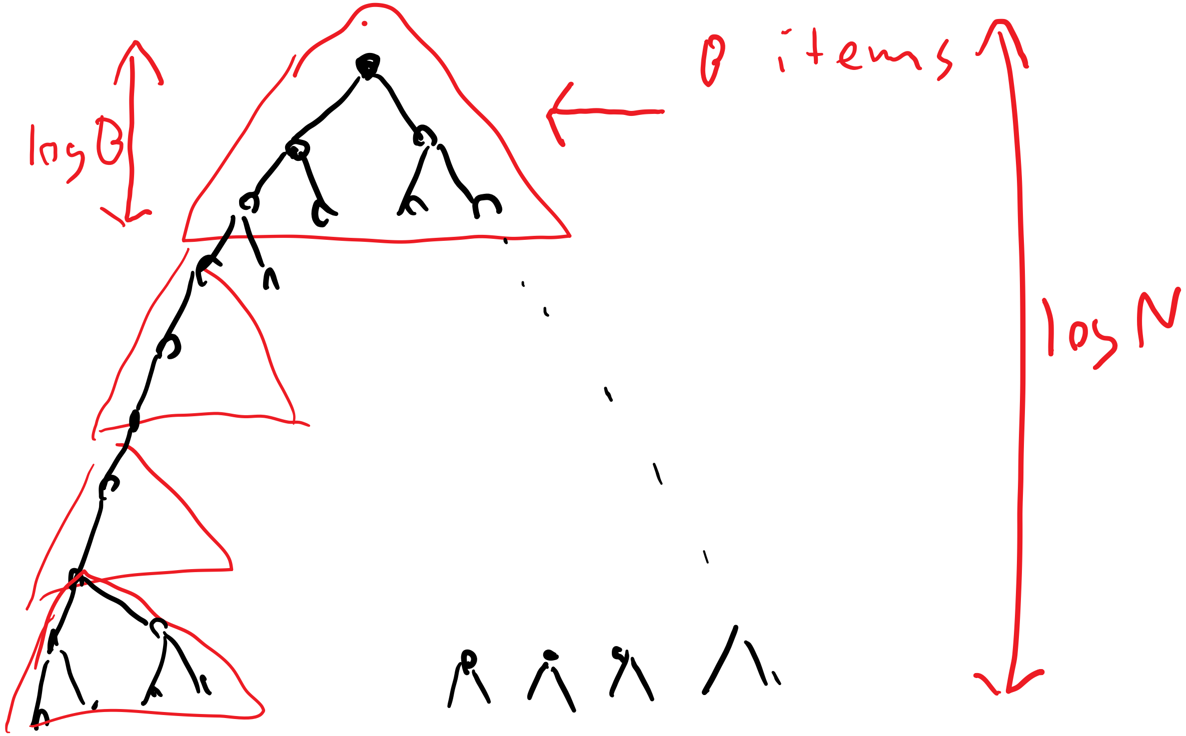
How Do We Make a B tree with no B?



Search goes through

$$\frac{\log N}{\log B}$$ red trees

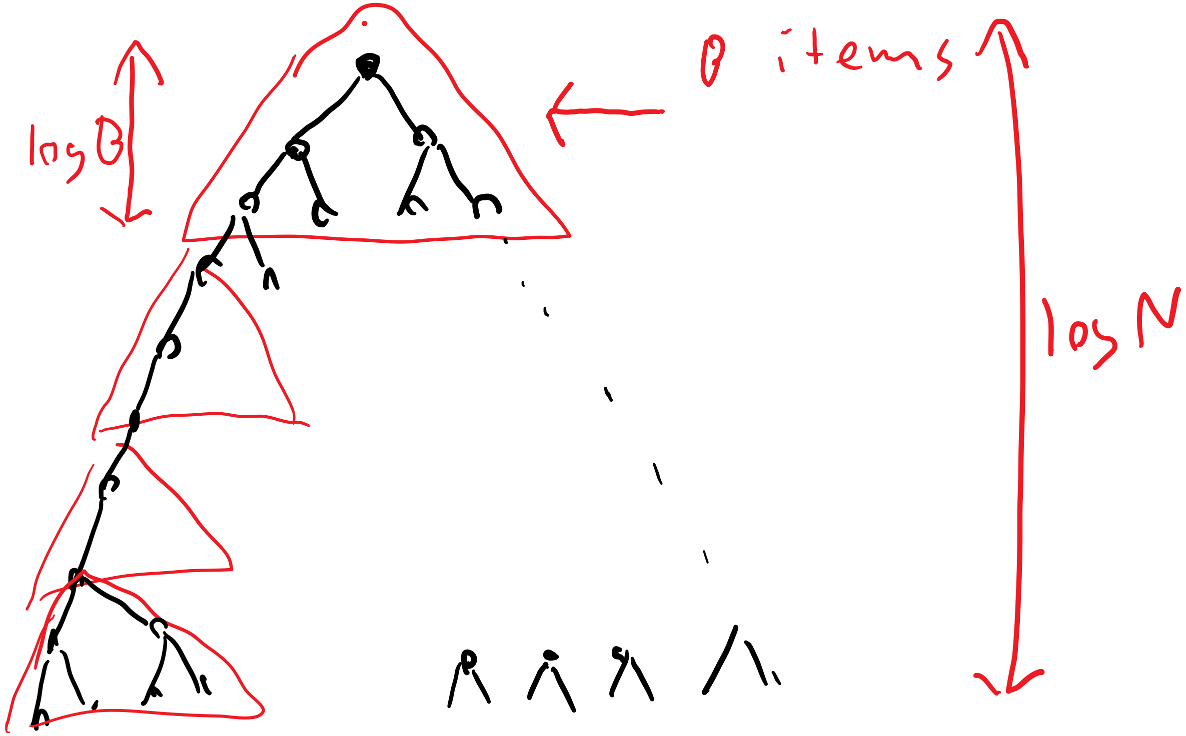
How Do We Make a B tree with no B?



Search goes through $\frac{\log N}{\log B}$ red trees

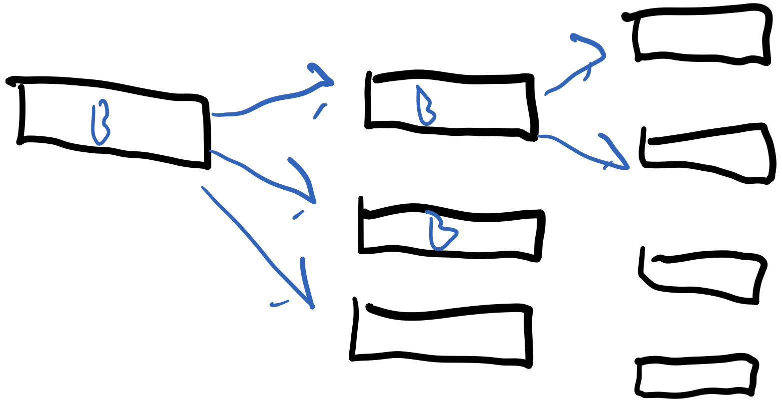
How are these blocks stored?

How Do We Make a B tree with no B?

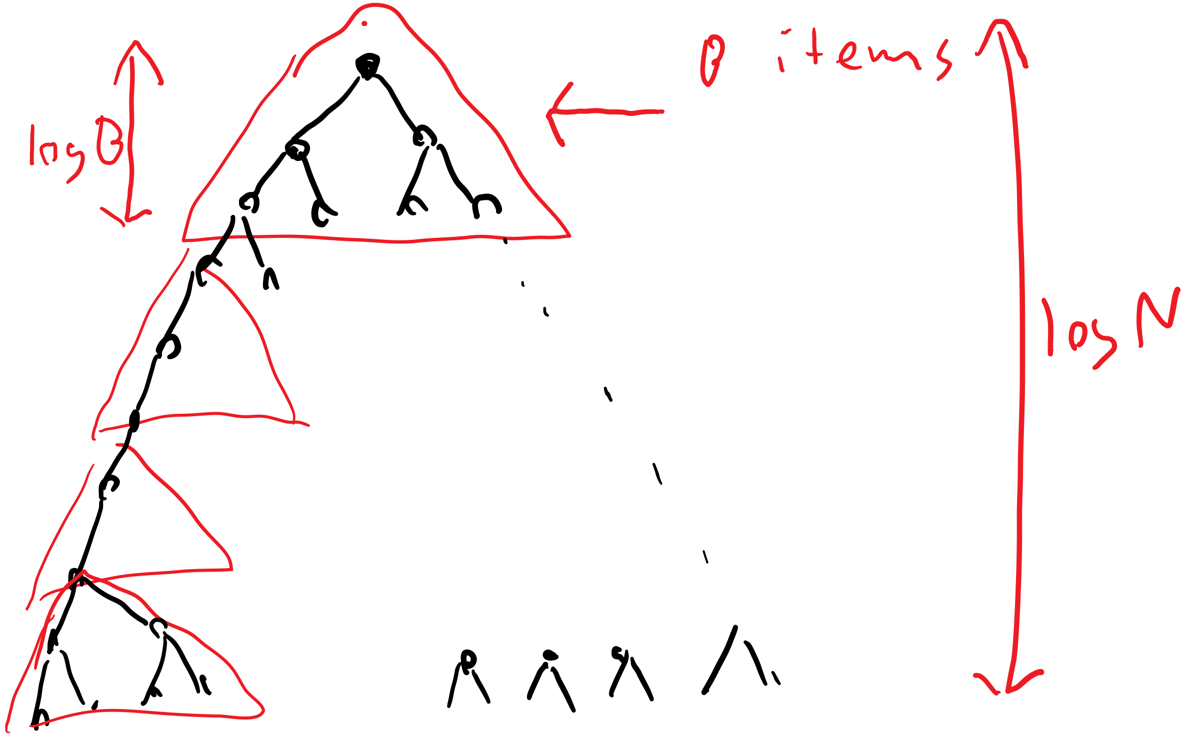


Search goes through $\frac{\log N}{\log B}$ red trees

How are these blocks stored?

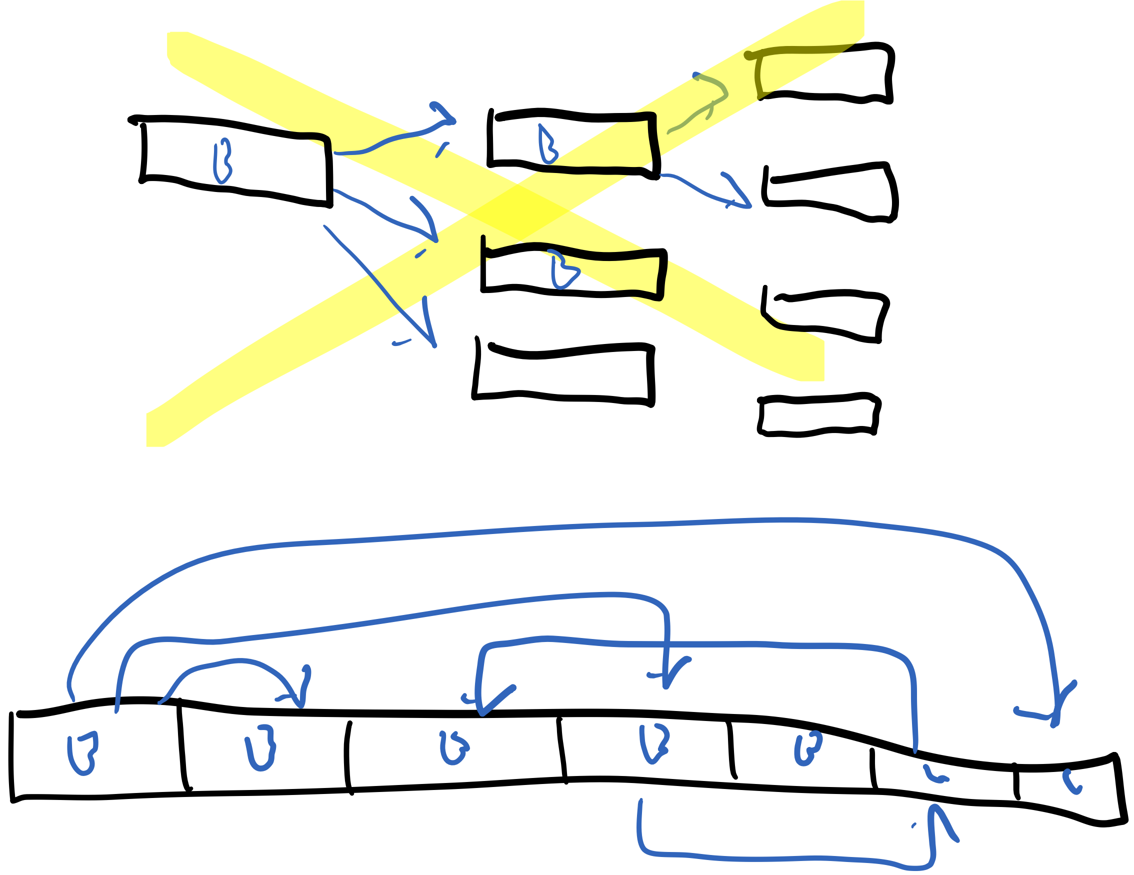


How Do We Make a B tree with no B?

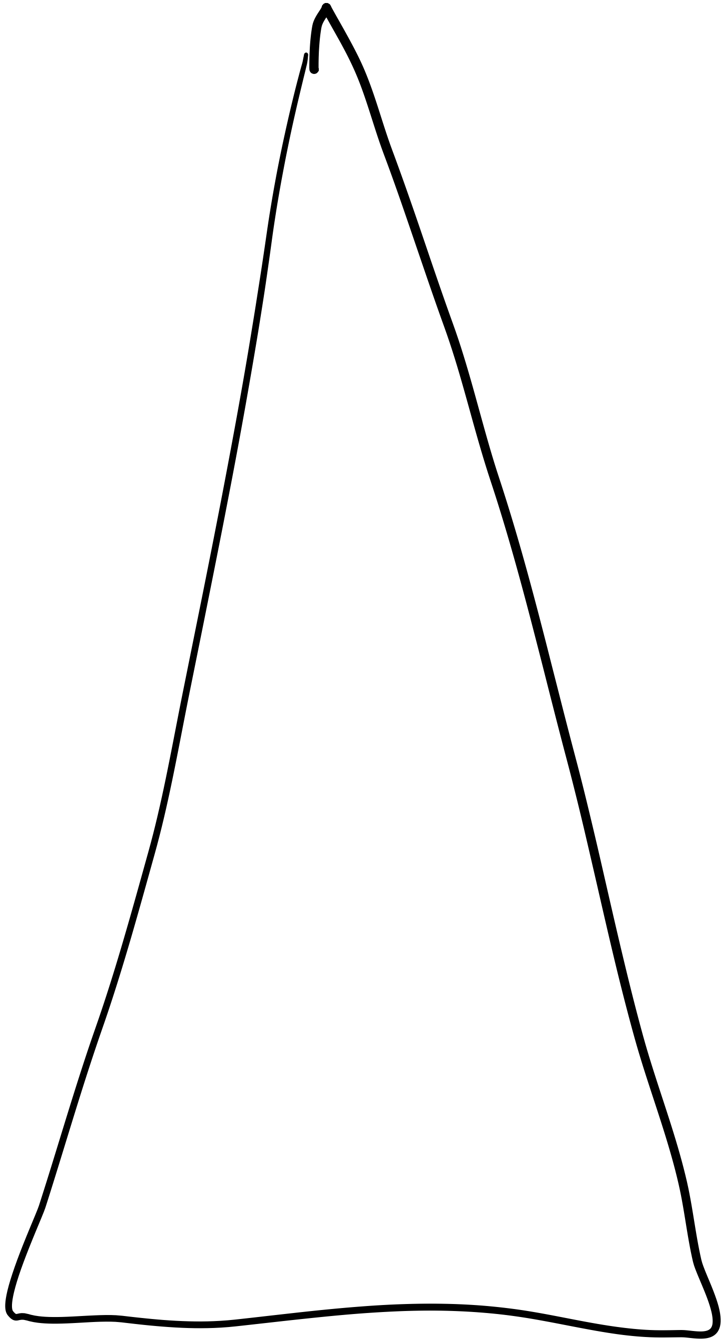


Search goes through $\frac{\log N}{\log B}$ red trees

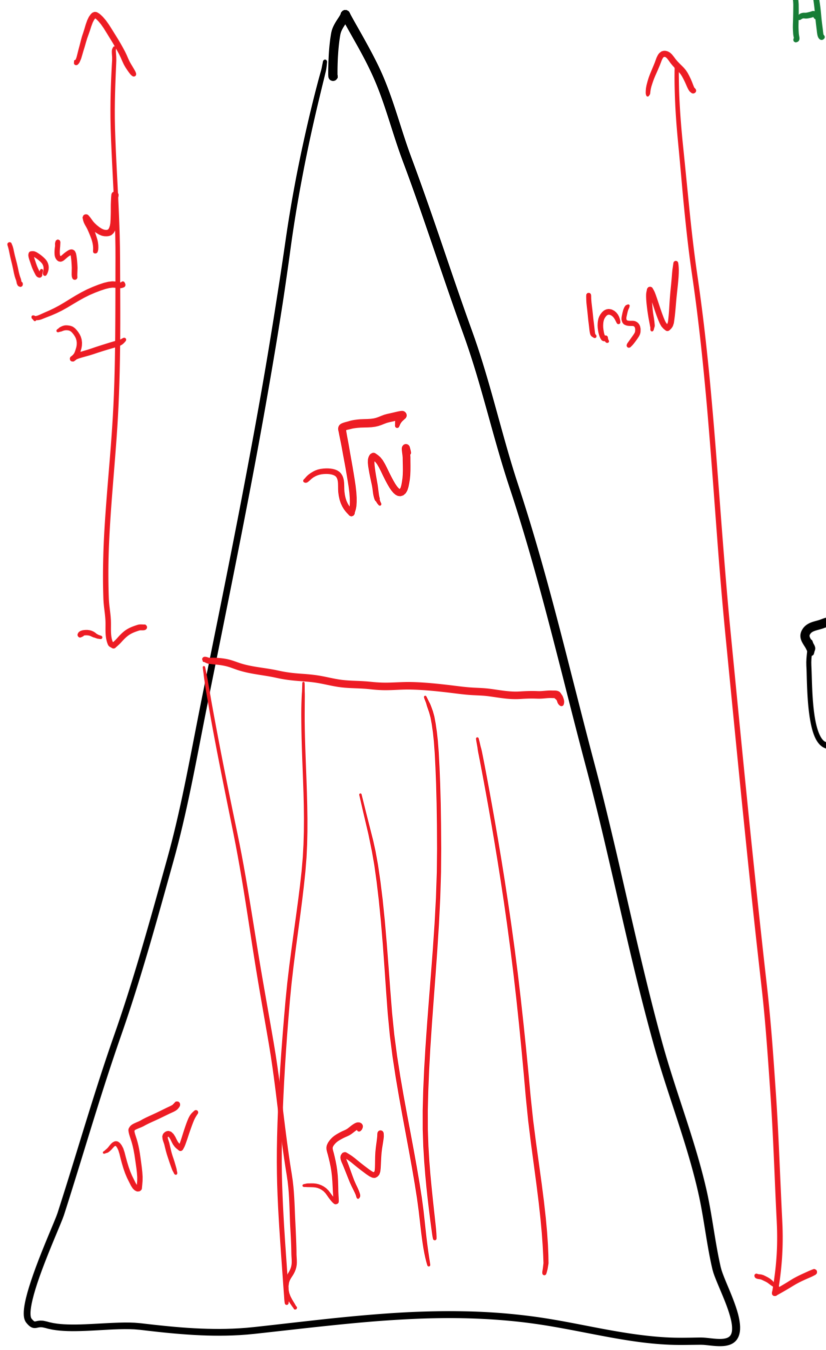
How are these blocks stored?



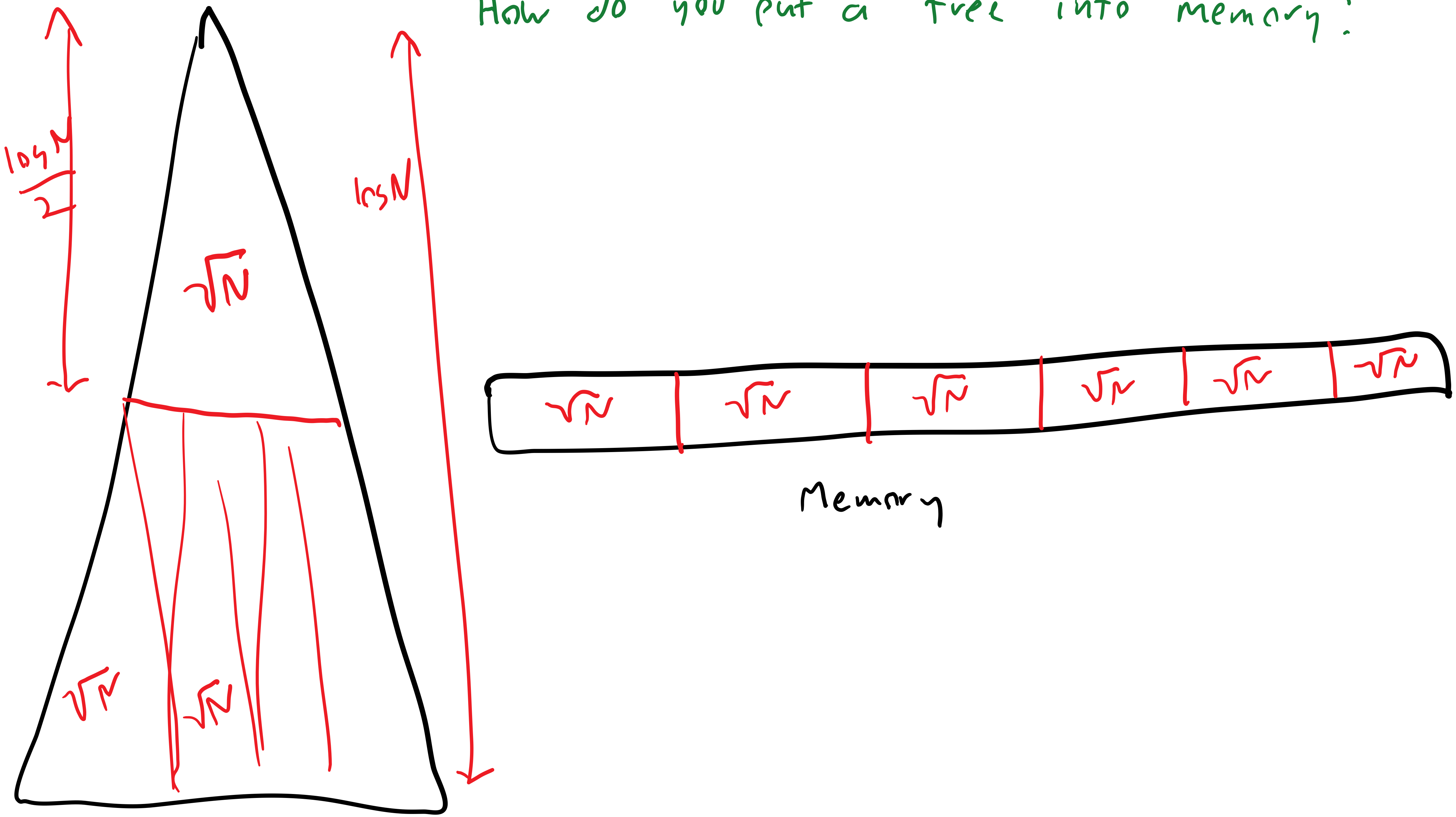
How do you put a tree into memory?



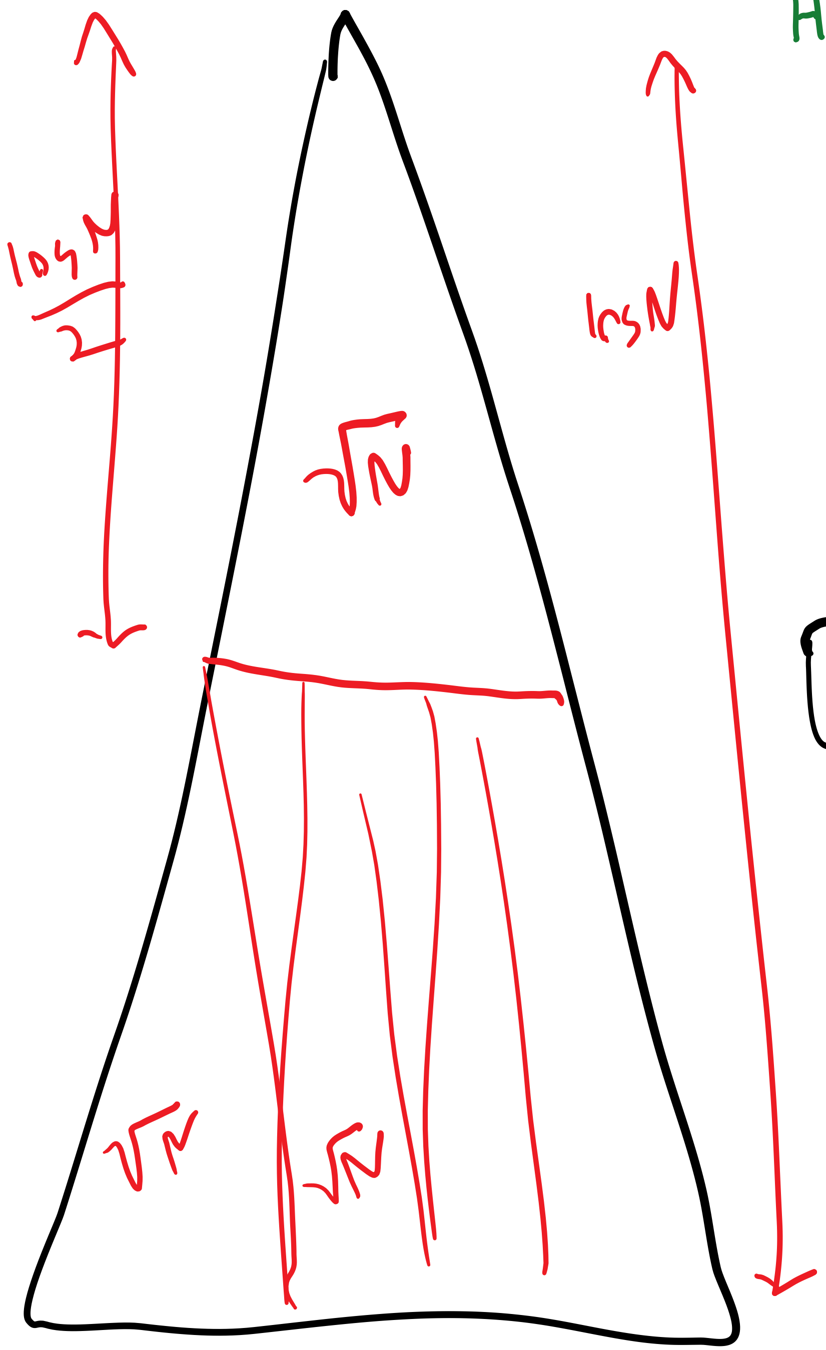
How do you put a tree into memory?

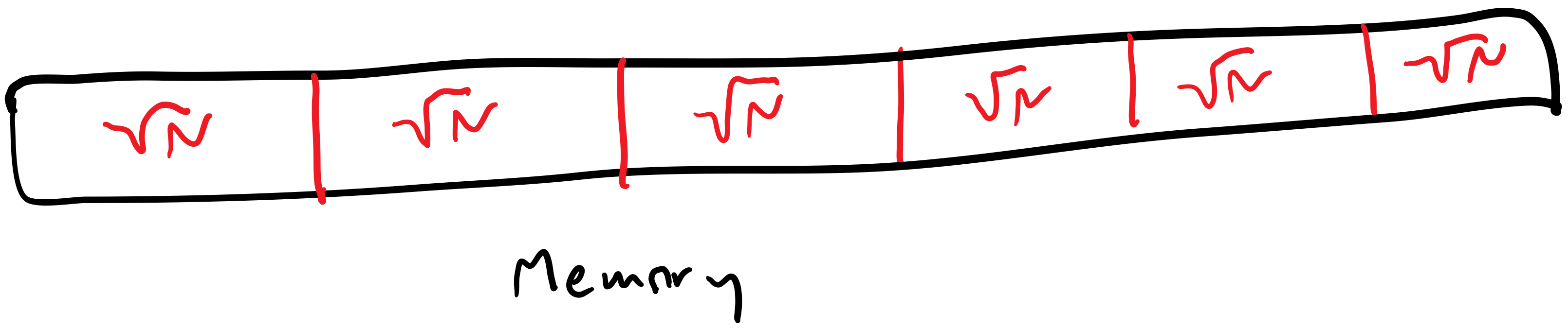


How do you put a tree into memory?



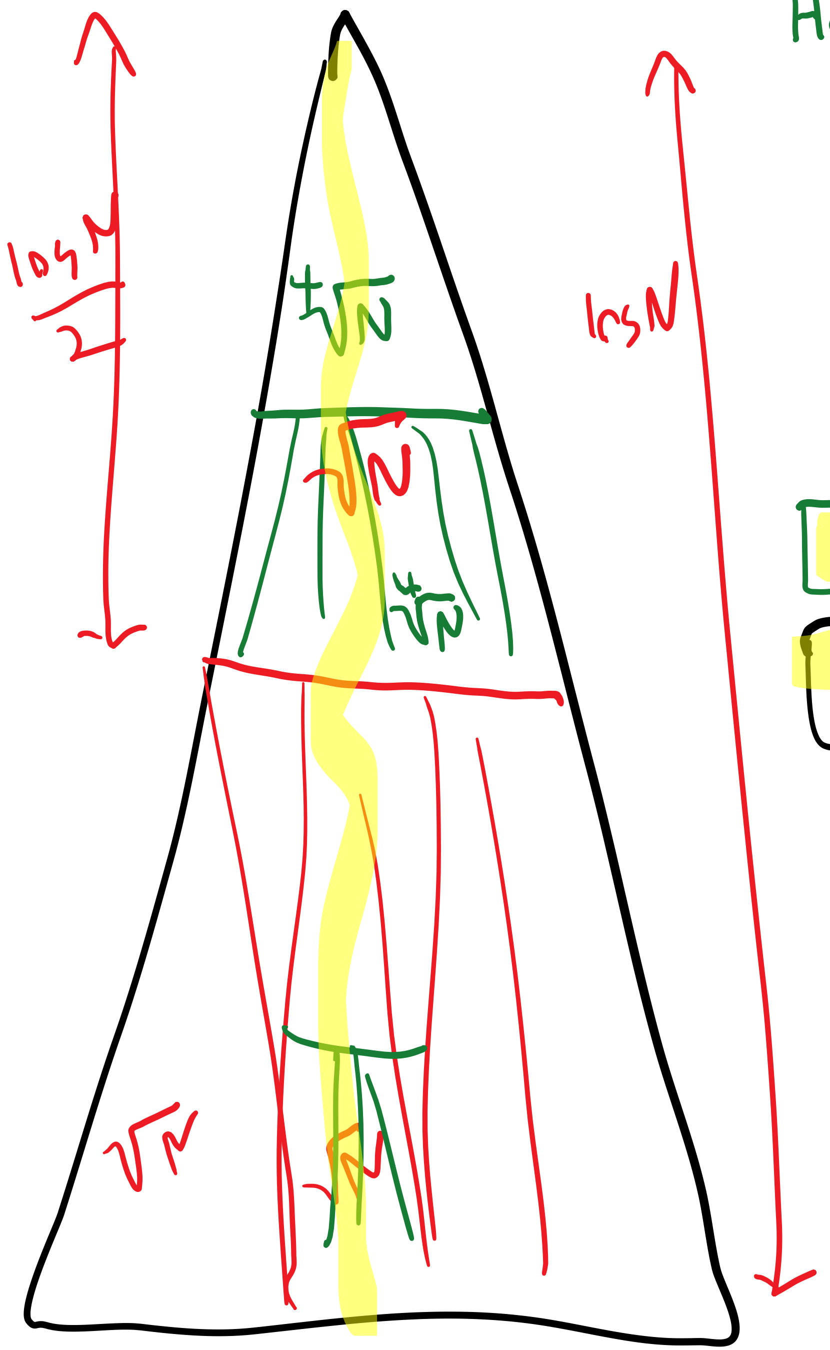
How do you put a tree into memory?

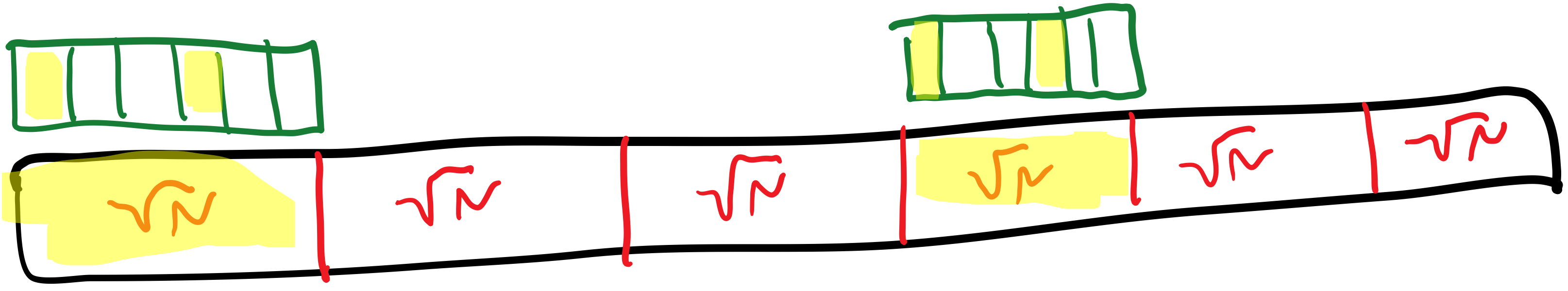




RECURSE

How do you put a tree into memory?

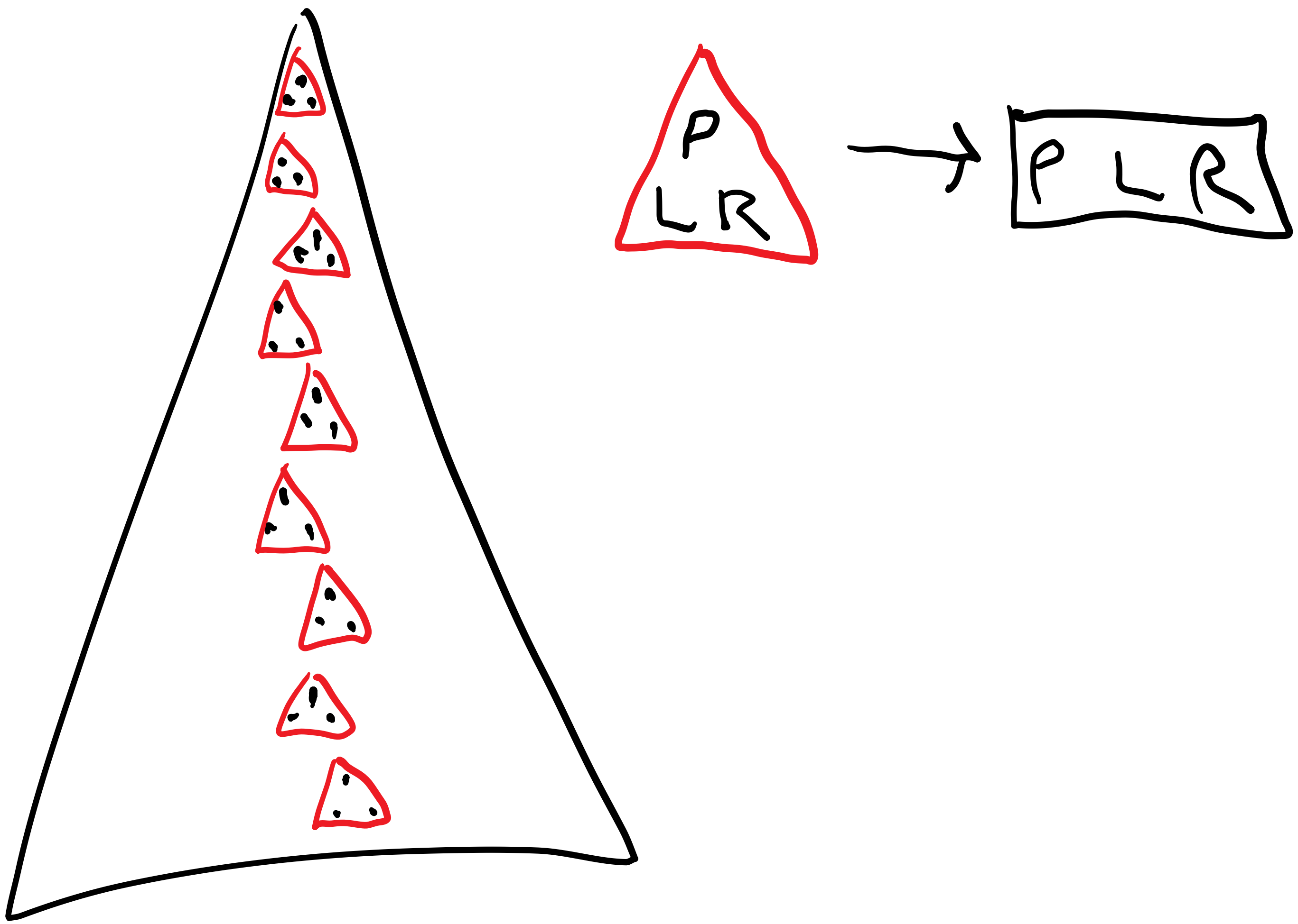




RECURSE

2	Blocks	$\sqrt{N}$
4	Blocks	$\sqrt[4]{N}$
⋮		
$\log_B N$	Blocks	B

ULTIMATE LOCALITY



Lots of results in

Cache - Oblivious Model

Persistence

Persistence

See the past

Persistence

See the past

Operation → Query — Read Only

Operation → Update — Read / Write

Types of Persistence

Partial | **Full** | **Confluent** | **Retroactive**

Types of Persistence

Partial | **Full** | **Confluent** | **Retroactive**

Update
↓
Update
↓
Update ← Query
↓
Update
↓
Update
↓
Update

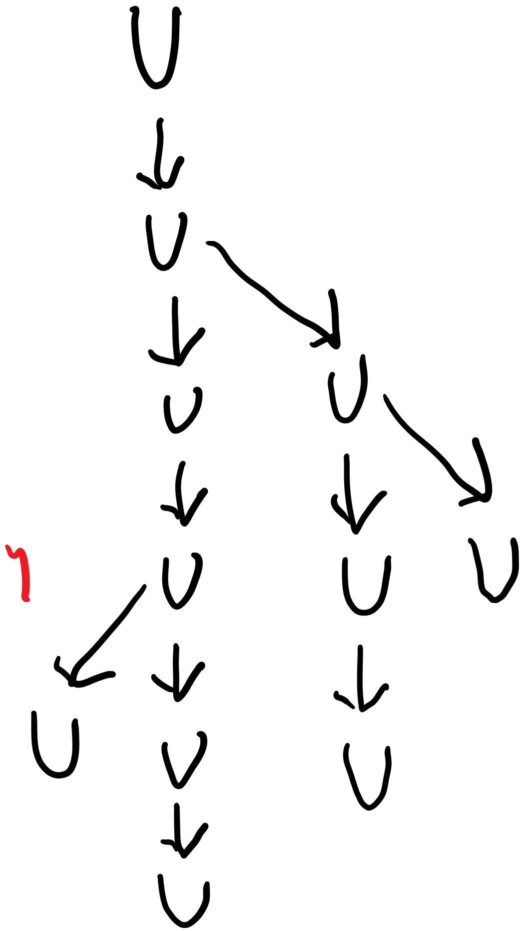
- Can Update Most recent version
- Query at any point in the past
- Time is linear

Types of Persistence

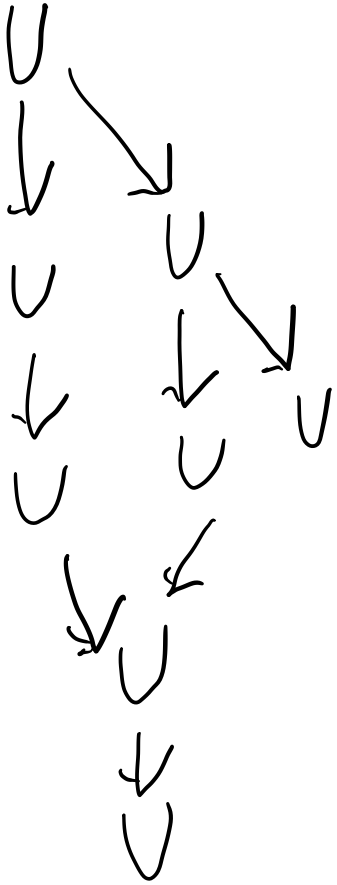
Partial

Update
↓
Update
↓
Update ← Query
↓
Update
↓
Update
↓
Update

Full



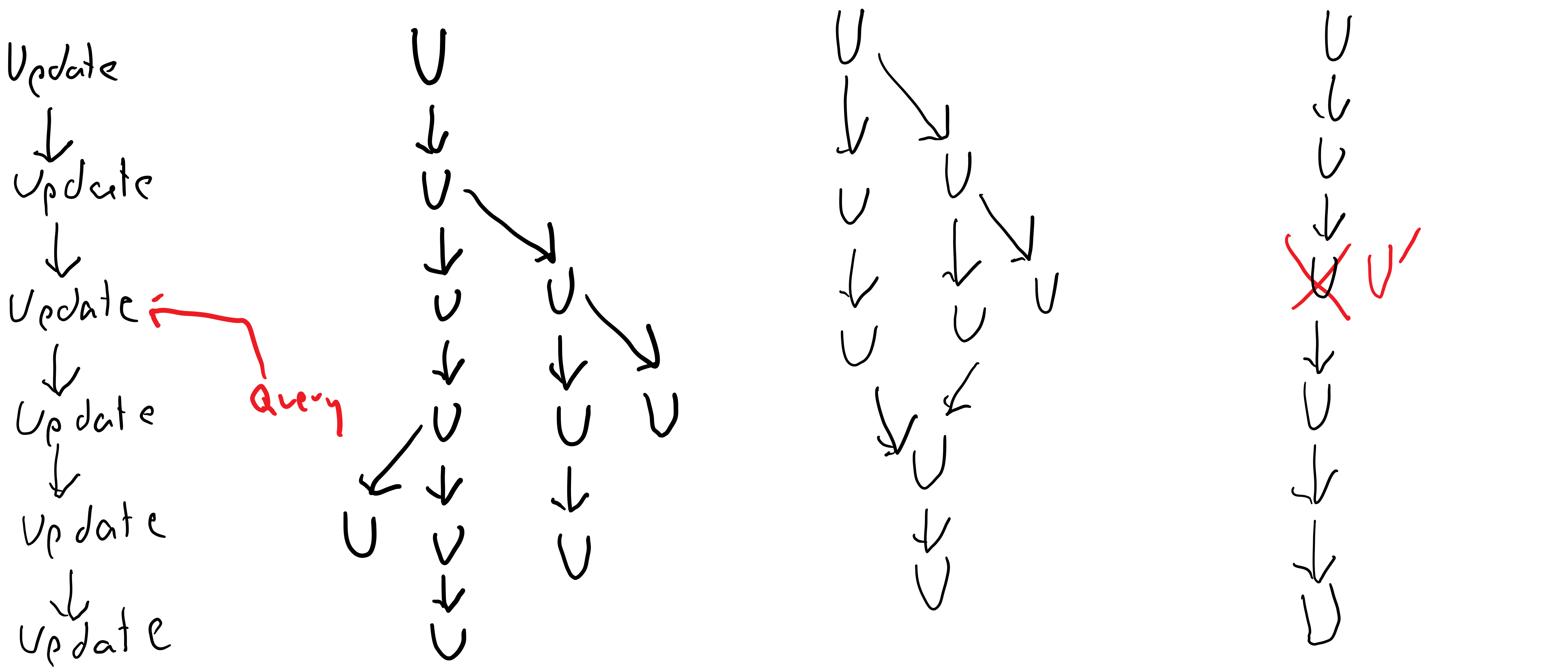
Confluent



Retroactive

Types of Persistence

[ST86]	[DSST89]	[FK03] [CIL12]	[DIL07]
Partial	**Full**	**Confluent**	**Retroactive**



Persisistence – Results

For pointer-based structures of constant indegree

- Partial, Full persistence is free
- Confluent persistence is complicated
- Retroactive is impossible

Persisistence — Results

For pointer-based structures of constant indegree

- Partial, Full persistence is free
- Confluent persistence is complicated
- Retroactive is impossible

But we care about the cache-oblivious model

C-O = locality Pointer = anti-locality

So, again, what is the C-O model?

So, again, what is the C-O model?

From point of view of an alg:

- Memory is an Array
- Read
- Write

So, again, what is the C-O model?

From point of view of an alg:

- Memory is an Array
- Read
- Write

So we need a persistent array that maintains localaty

Geometric View

time

$A[8]=4$

$A[2]=7$

$A[2]=5$

$A[15]=0$

$A[8]=5$

$A[1]=6$

$A[2]=4$

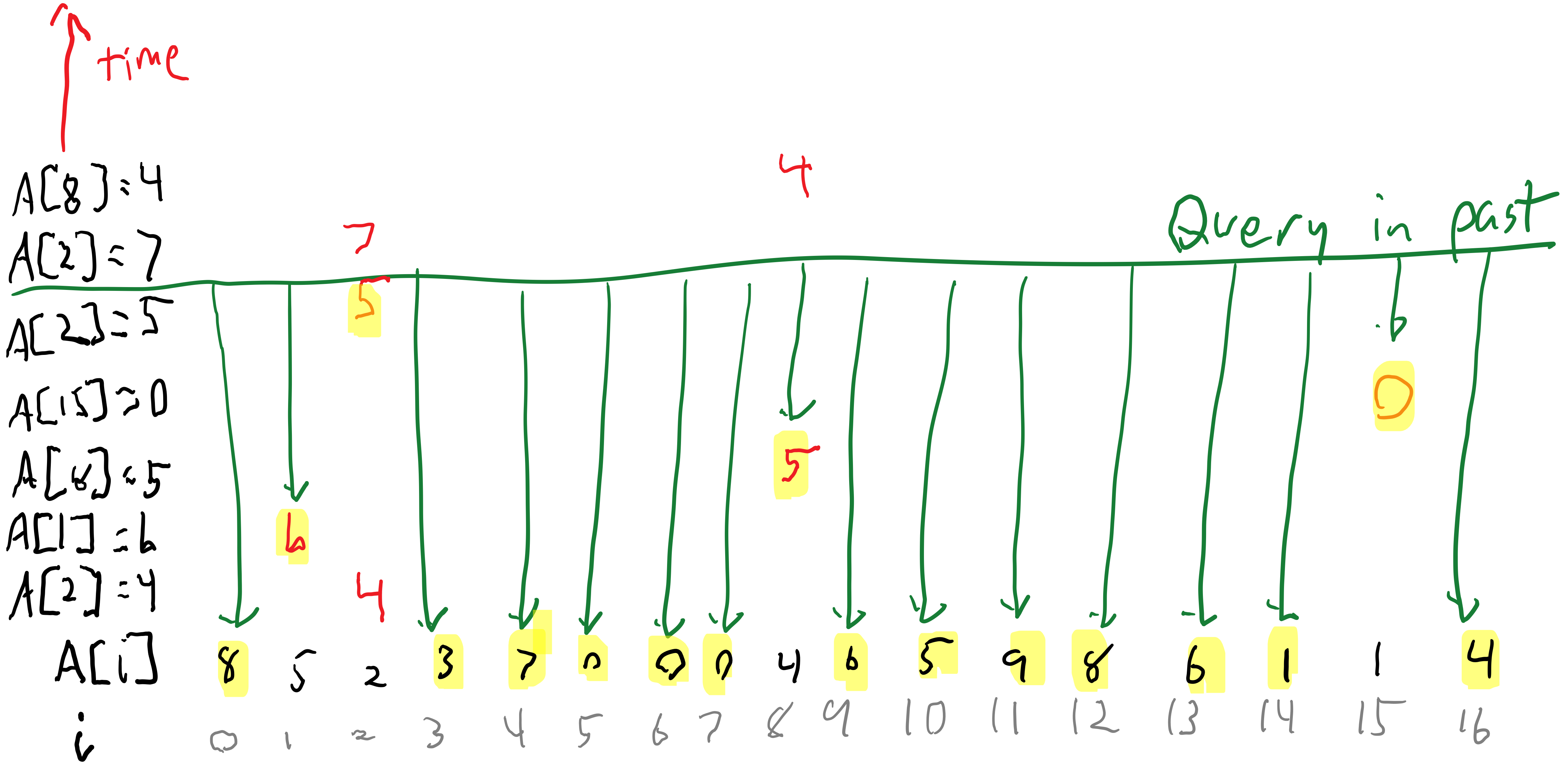
$A[i]$	8	5	2	3	7	0	0	0	4	6	5	9	8	6	1	1	4
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Geometric View

time ↑

A[8]=4									4								
A[2]=7			7														
A[2]=5			5														
A[15]=0																0	
A[8]=5									5								
A[1]=6		6															
A[2]=4			4														
A[i]	8	5	2	3	7	0	0	0	4	6	5	9	8	6	1	1	4
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Geometric View

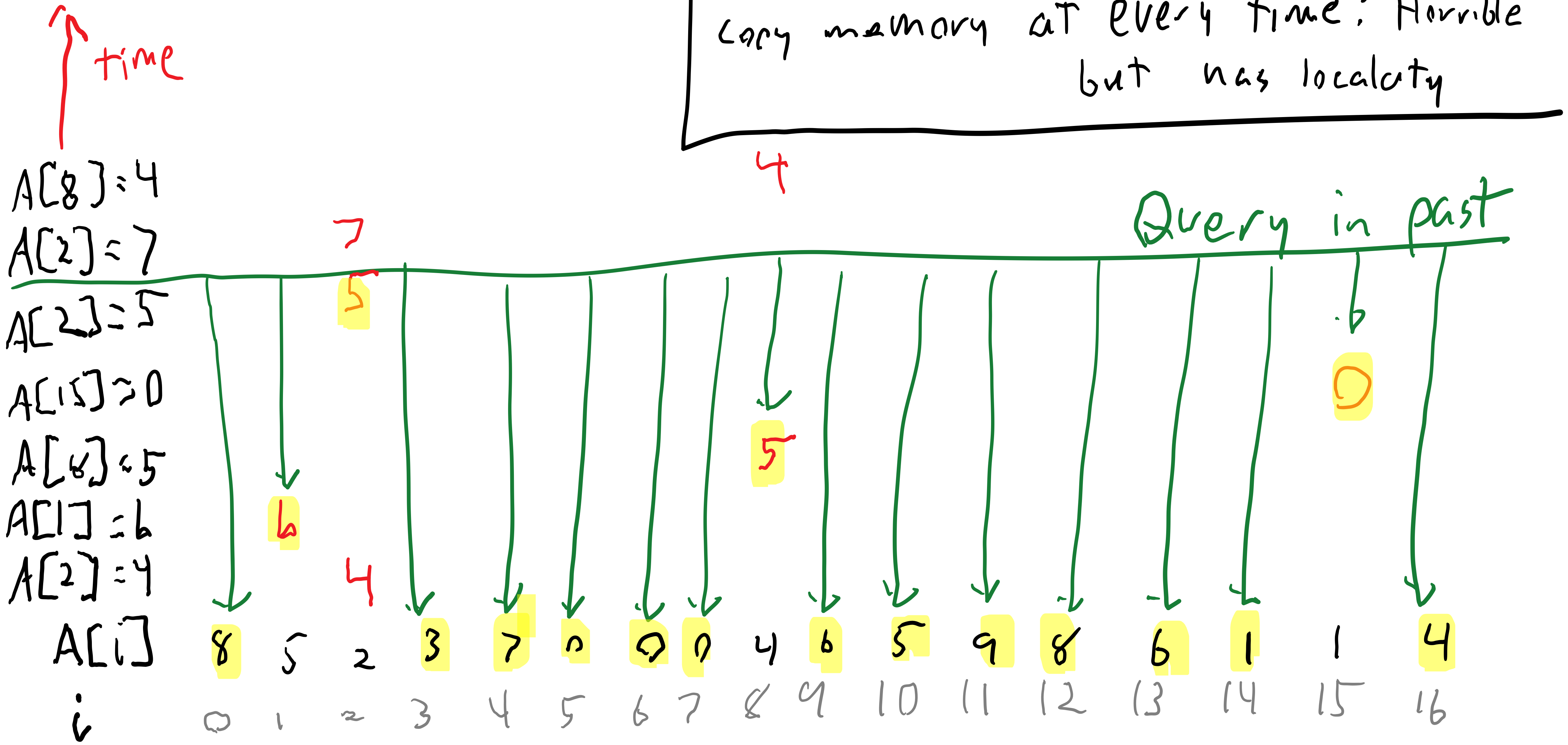


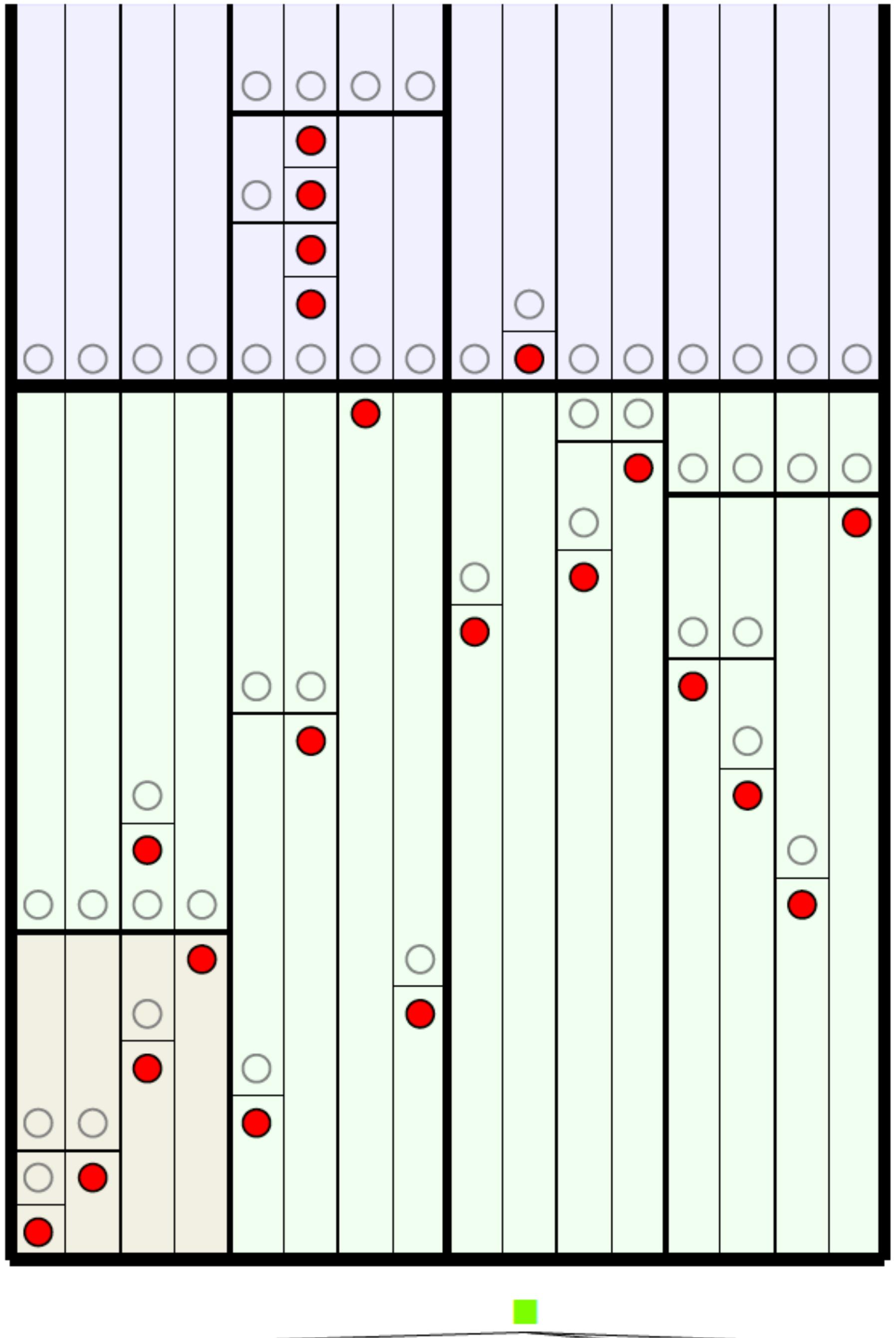
Geometric View

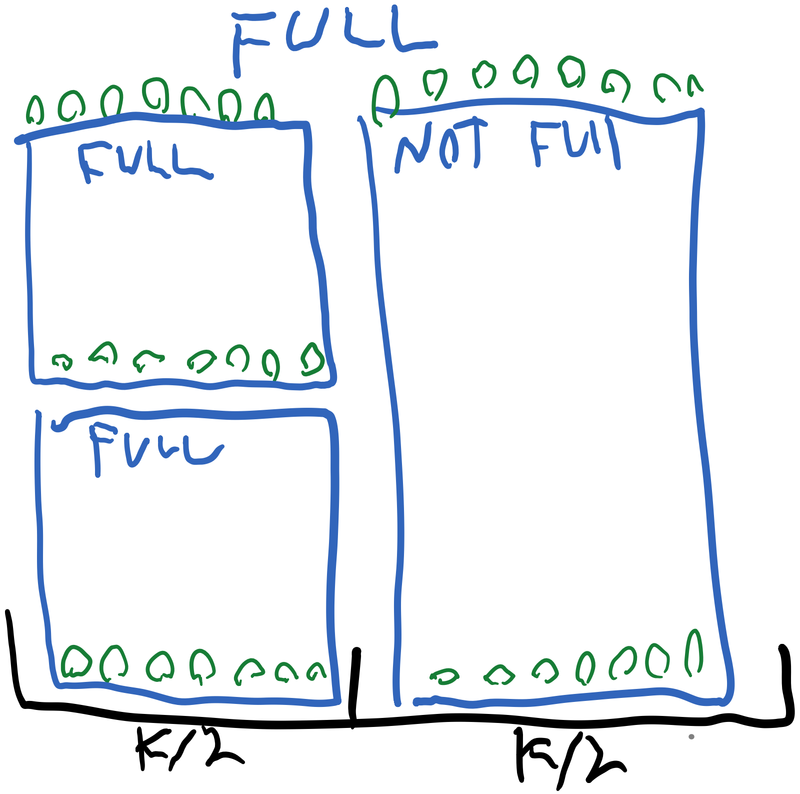
Store in obvious way: No Locality

~or~

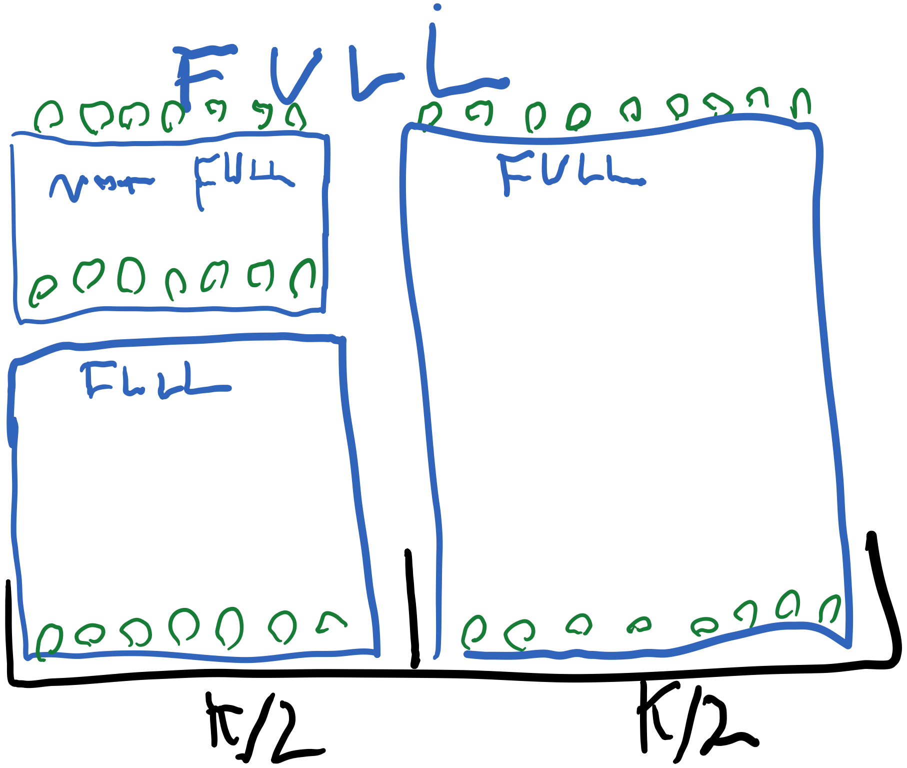
Copy memory at every time: Horrible but has locality



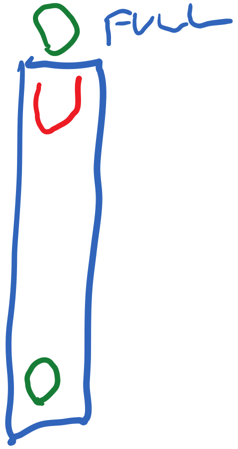




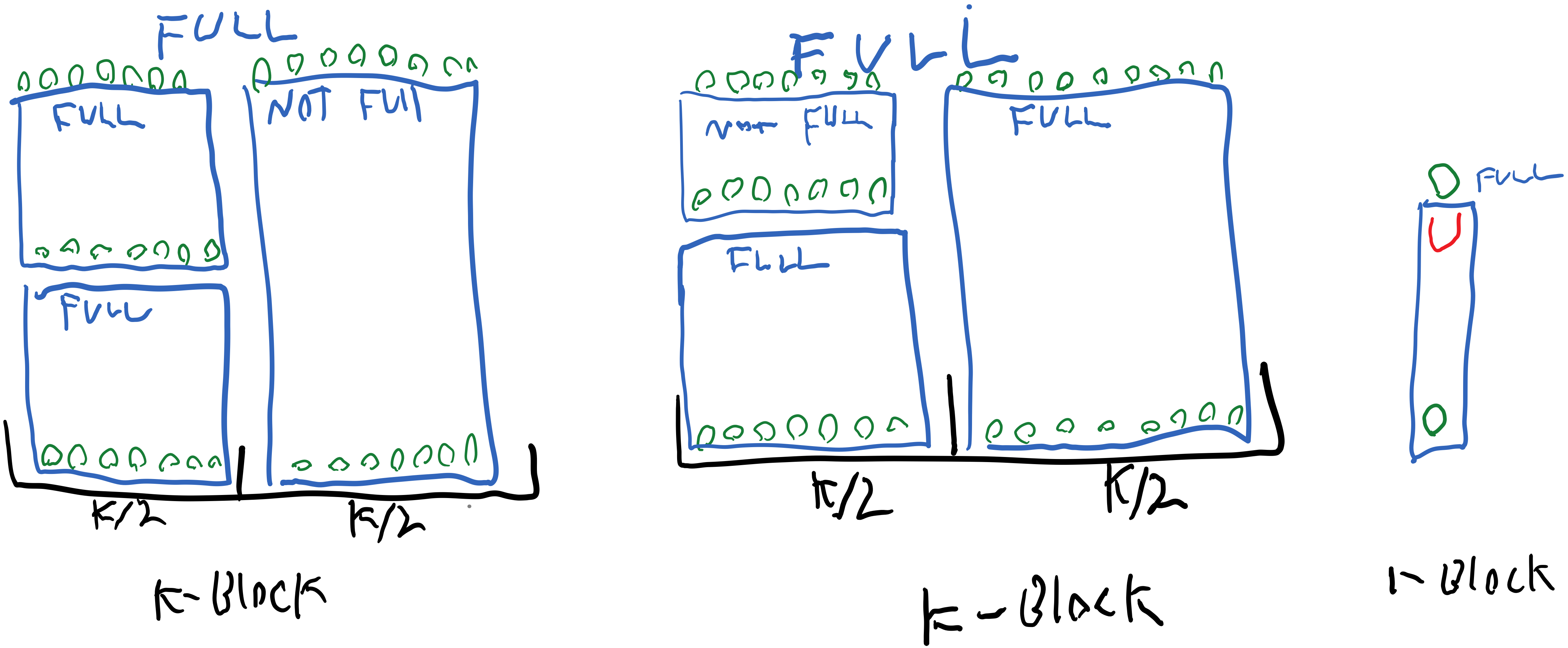
K-Block



K-Block



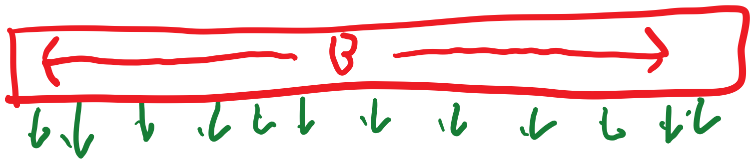
1-Block



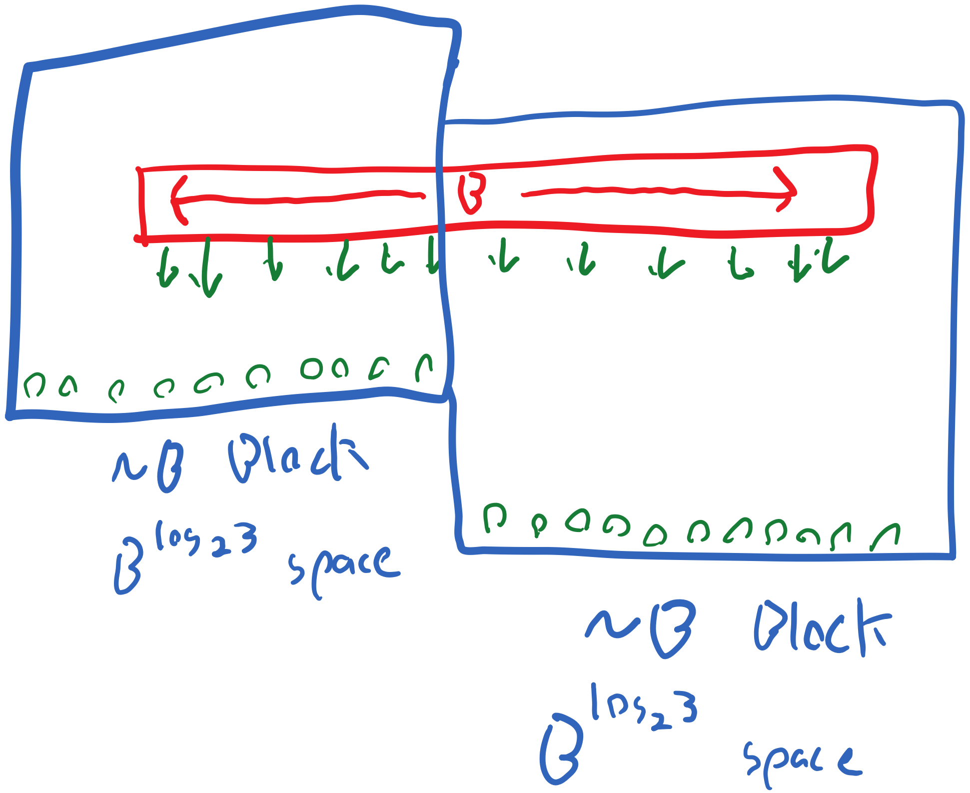
Store as a 3-ary Tree

Each K-Block is stored in $3^{\log_2 K} = K^{\log_2 3}$ space

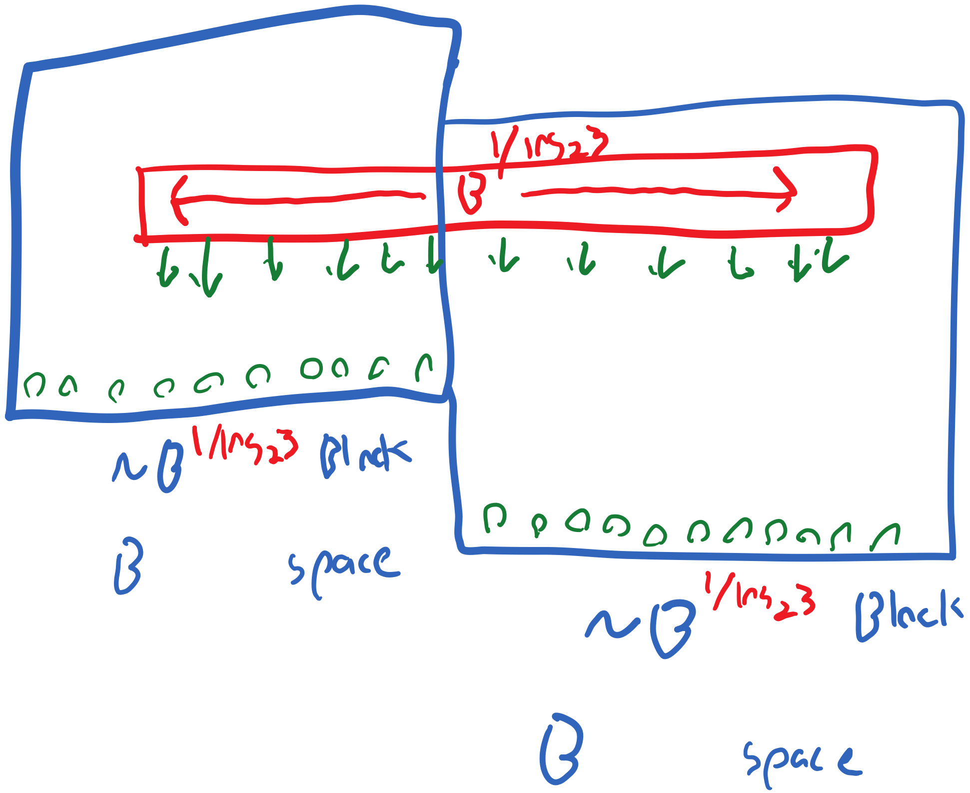
Suppose I wanted to look at 1 Block of size B in memory at some time



Suppose I wanted to look at 1 Block of size B in memory at some time



Suppose I wanted to look at 1 Block of size $B^{1/\log_2 3}$ in memory at some time



A block of size $B^{1/\log_2 3} \approx B^{0.63}$

In the past can be found in 2 ranges of size B in memory

Suppose I wanted to look at 1 Block of size $B^{1/\log_2 3}$ in memory at some time

$T(B)$ = Runtime of Alg A in C-O model with Block size B

THM: Persistent queries take time $O(T(B^{1/\log_2 3}))$

A block of size $B^{1/\log_2 3} \approx B^{0.63}$

In the past can be found in 2 ranges of size B in memory

Suppose I wanted to look at 1 Block of size $B^{1/\log_2 3}$ in memory at some time

$T(B) =$ Runtime of Alg A in C-O model with Block size B

THM: Persistent queries take time $O(T(B^{(1-\epsilon)/\log_2 3}) \log_B S)$

space

A block of size $B^{1/\log_2 3} \approx B^{0.63}$

In the past can be found in 2 ranges of size B in memory

Suppose I wanted to look at 1 Block of size $B^{1/\log_2 3}$ in memory at some time

$T(B)$ = Runtime of Alg A in C-O model with Block size B

A block of size $B^{1/\log_2 3} \approx B^{0.63}$

In the past can be found in 2 ranges of size B in memory

THM: Persistent queries take time $O(T(B^{(1-\epsilon)/\log_2 3})\log_B S)$

space

$O(\log_B N) \to O(\log_B^2 N)$ [worse than BCR 02]

$O(N/B) \to O(N \log_B N / B^{0.63})$.

Notes

- I've ignored M
- Log S space blowup
- Full persistence probably possible

Thoughts

The C-O model is a torturous way to get algs with great locality.

Better way?

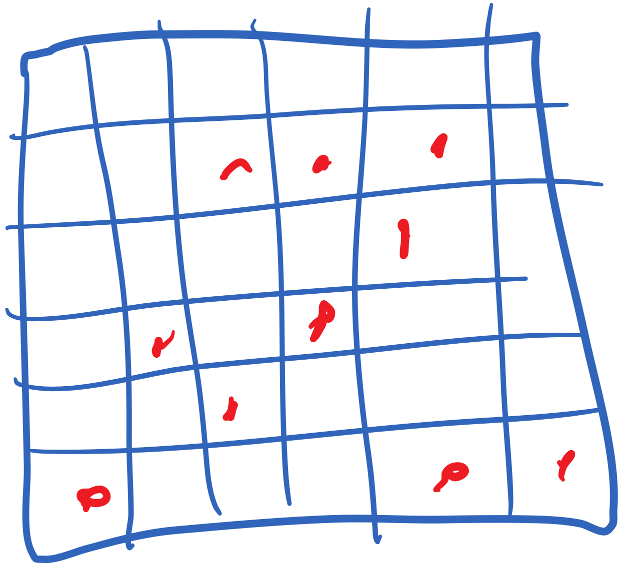
Arxiv
[IKV19]

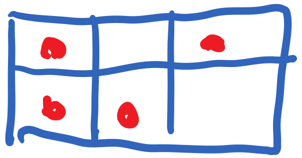
Geometry?

Part III: Forbidden Matricies

Idea: Use Results on extremal functions of 0-1 Matricies with forbidden submatricies

Idea: Use Results on extremal functions of 0-1 Matricies with forbidden submatricies



with NO  has $O(n \log n)$ •'s

[G. Tardos 05]

Method

Data Structure

Method

Data Structure

Geometric View

Find Forbidden Submatricies

Method

Data Structure

↓

Geometric View

↓

Find Forbidden Submatricies

Method

Data Structure

Geometric View

Find Forbidden Submatricies

Compute / look up extremal Function

Method

[Pettie 2010]

Data Structure

Geometric View

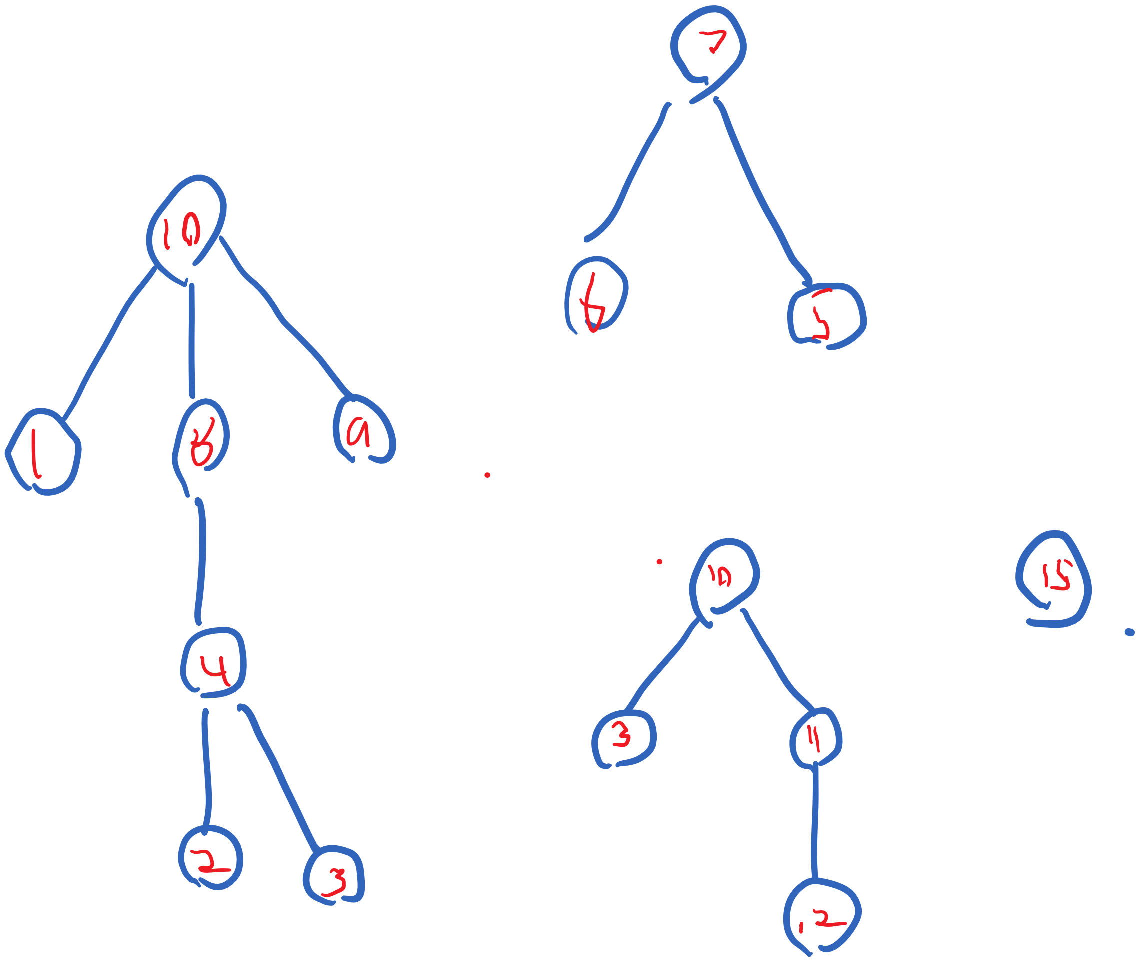
Find Forbidden Submatricies

Compute/look up extremal Function

Runtime Bound!

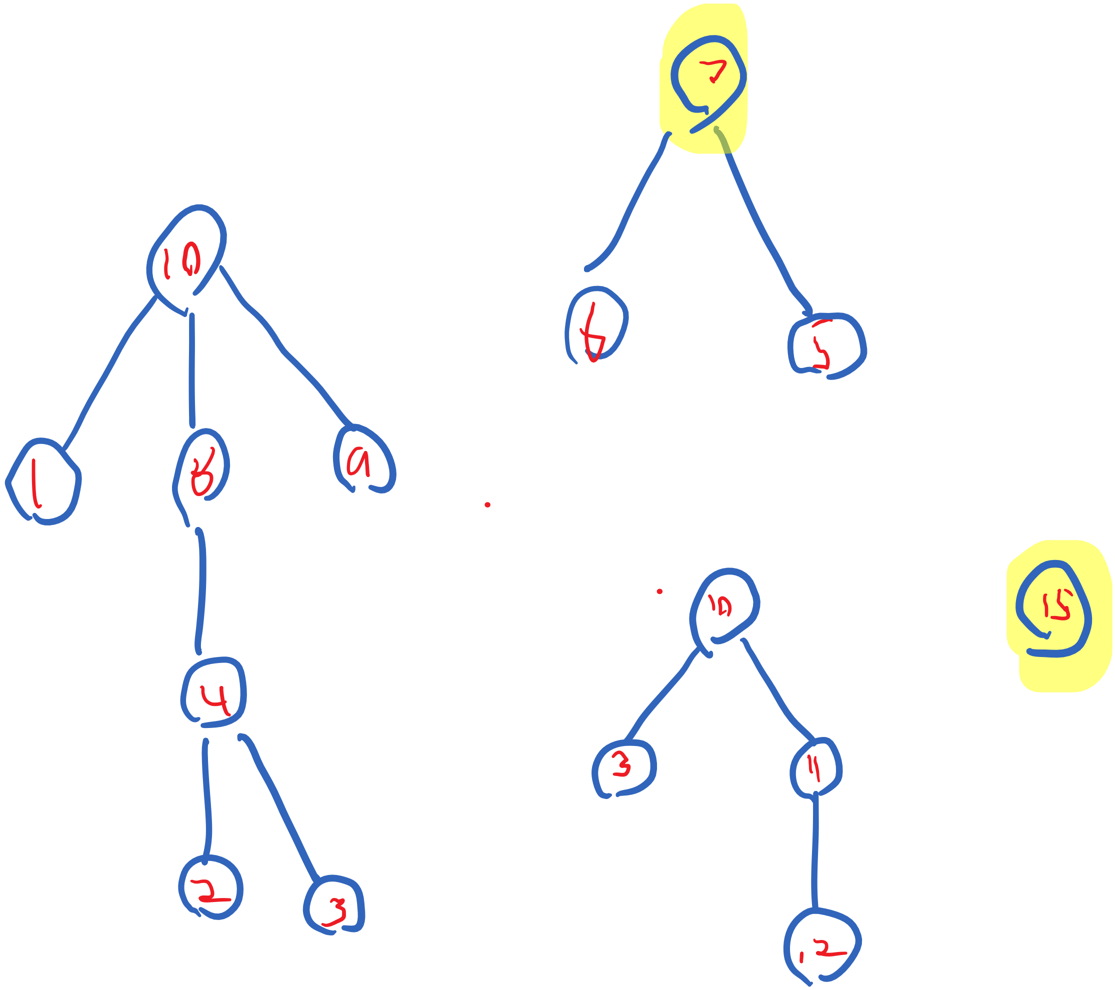
Example: Union-Find w Path Compression.

[GF64, HU73, T75]



- Who is my root
- Union

Example: Union-Find w Path Compression.

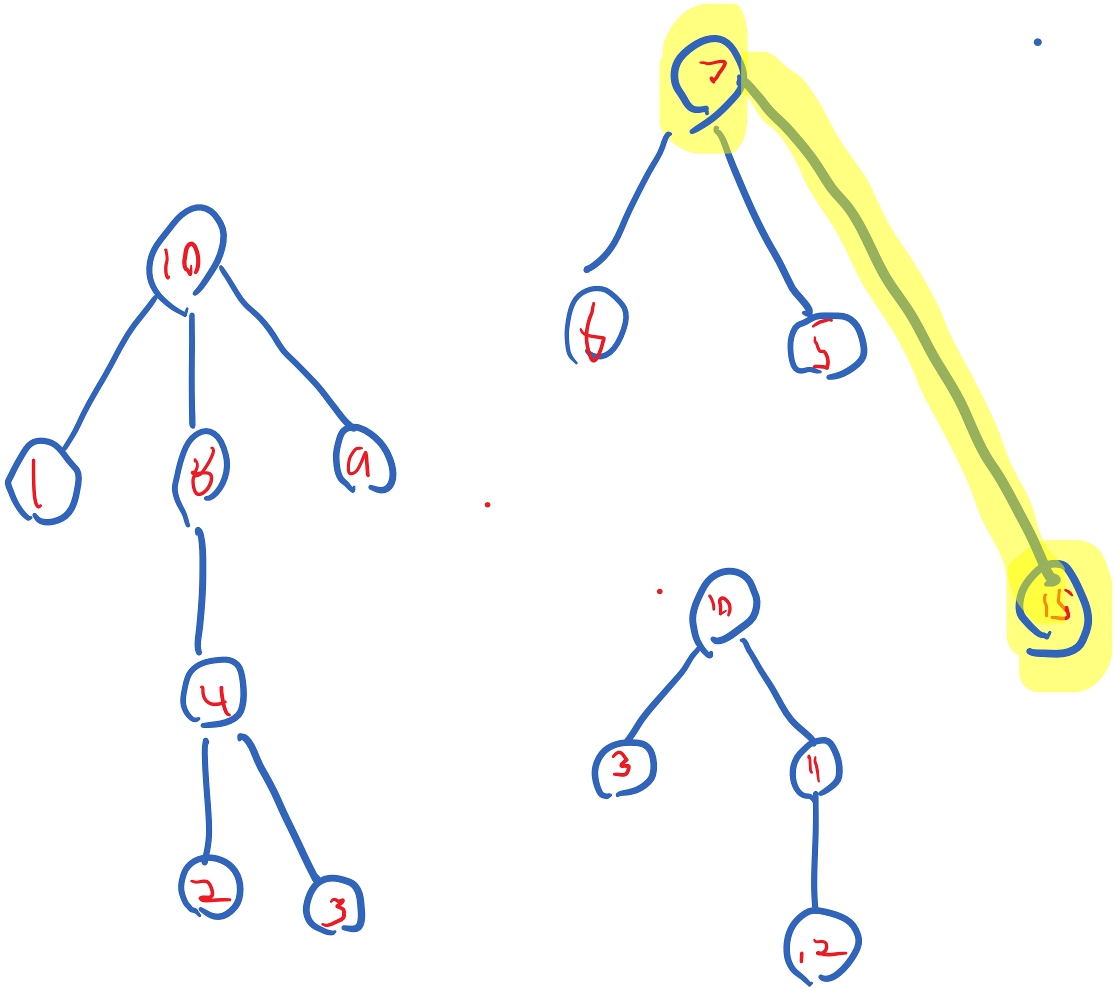


- Who is my root

- Combine

Union

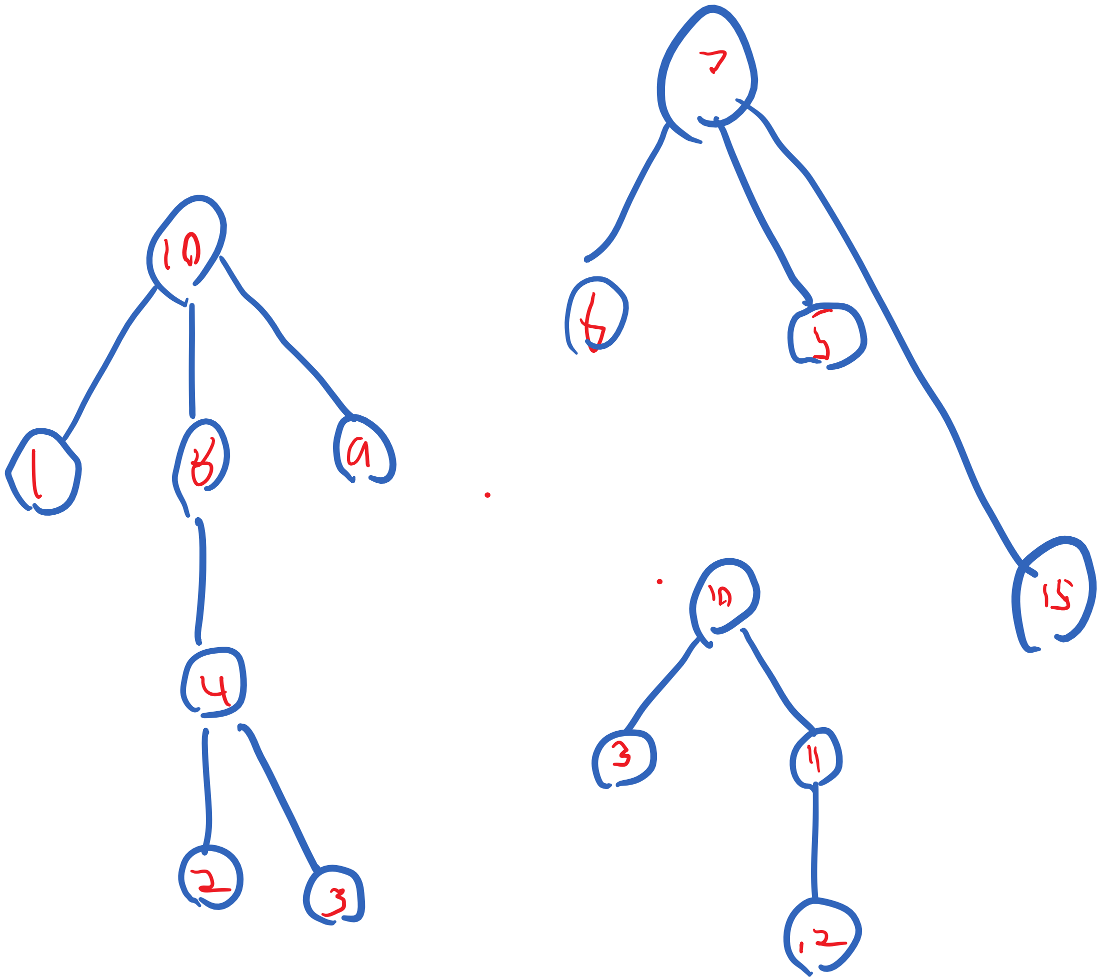
Example: Union-Find w Path Compression.



- Who is my root
- Combine

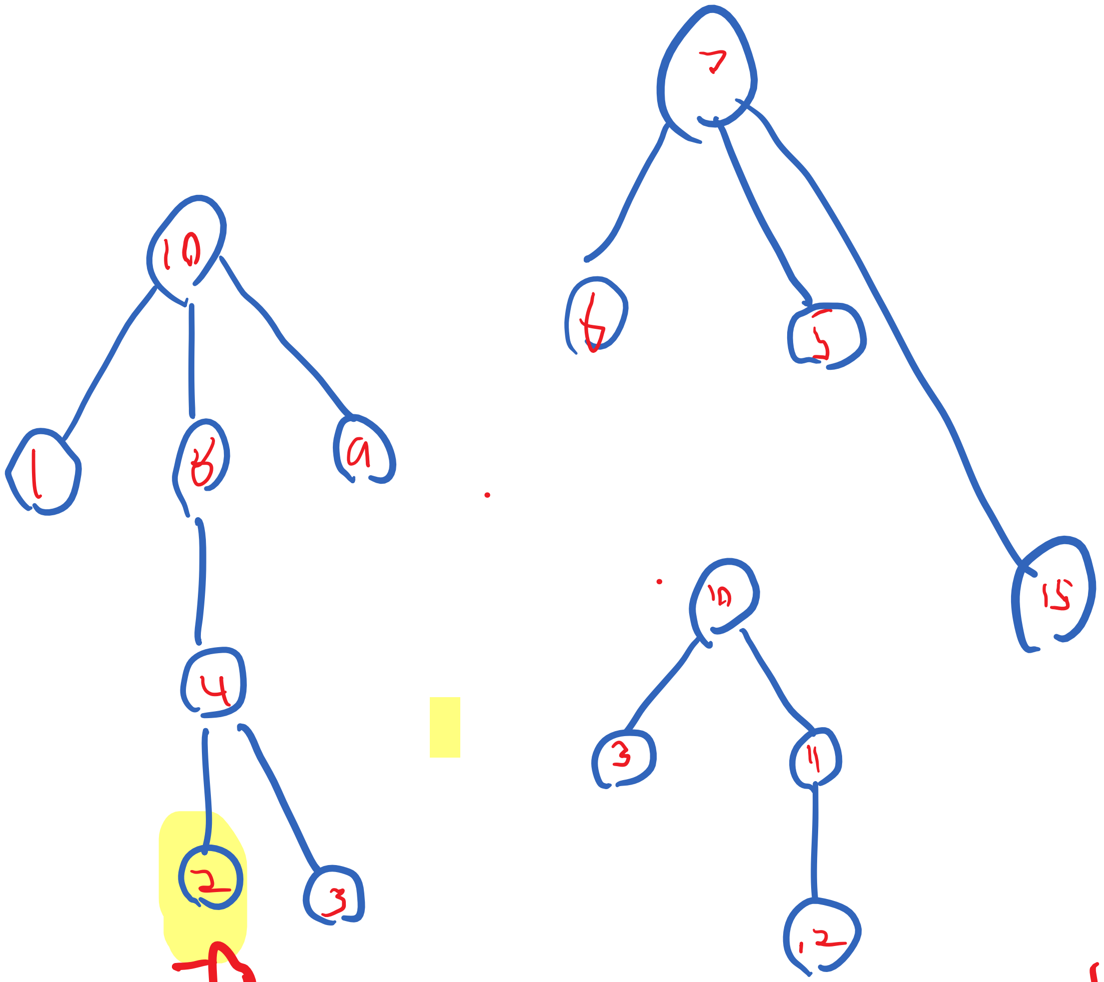
Union

Example: Union-Find w Path Compression.



- Who is my root
- Combine

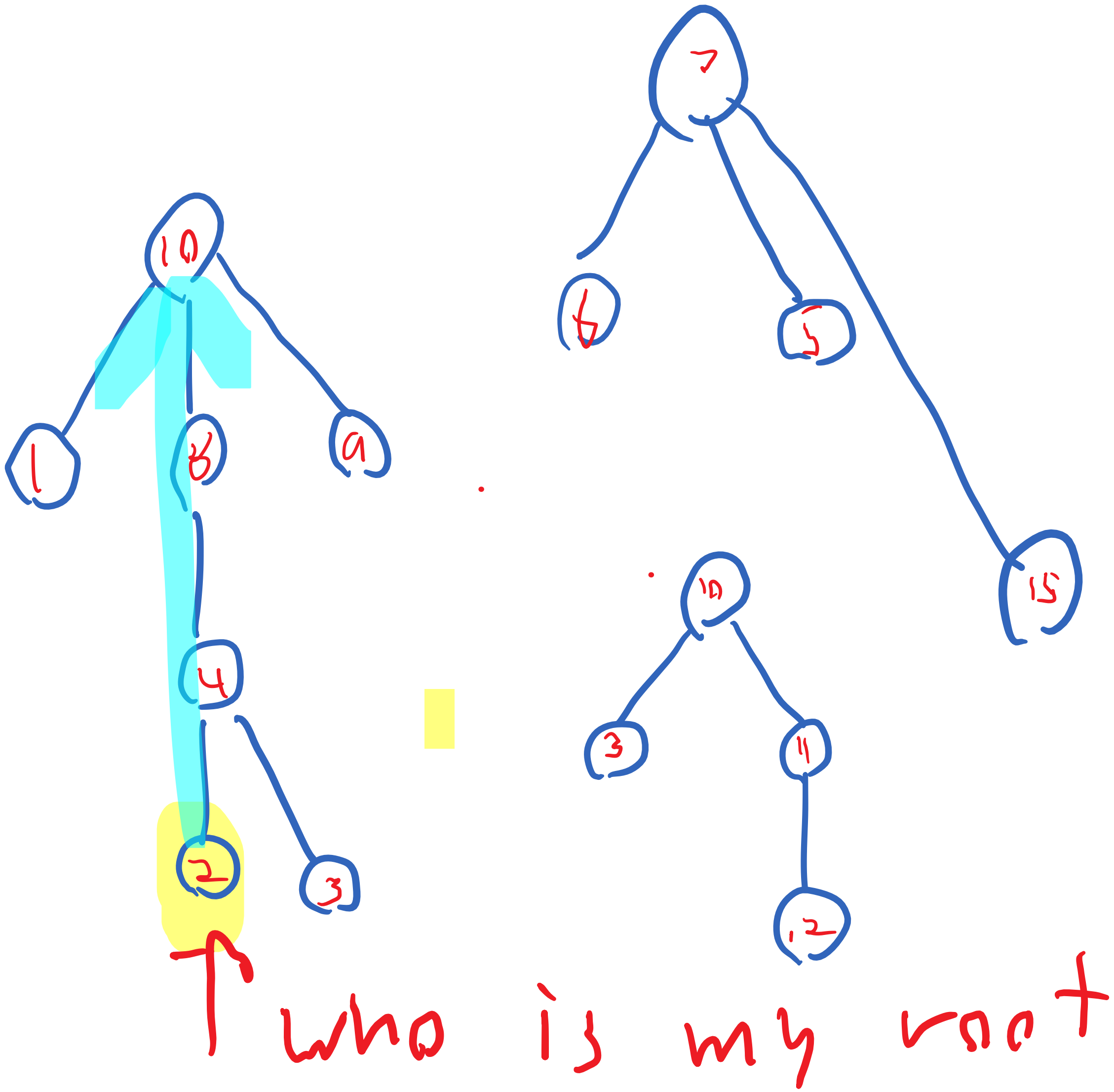
Example: Union-Find w Path Compression.



↑ who is my root

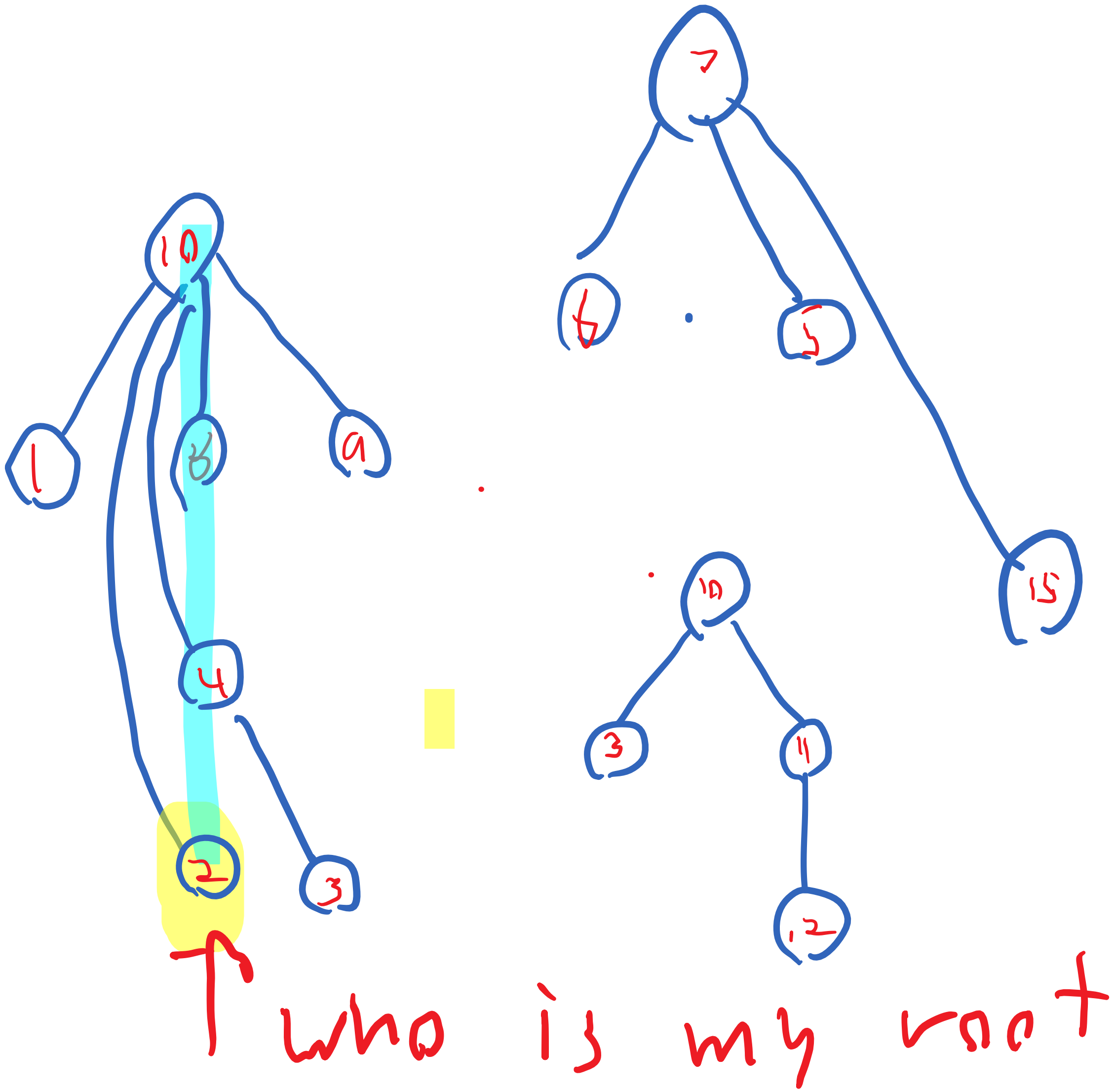
- Who is my root
- Combine

Example: Union-Find w Path Compression.



- Who is my root
- Combine

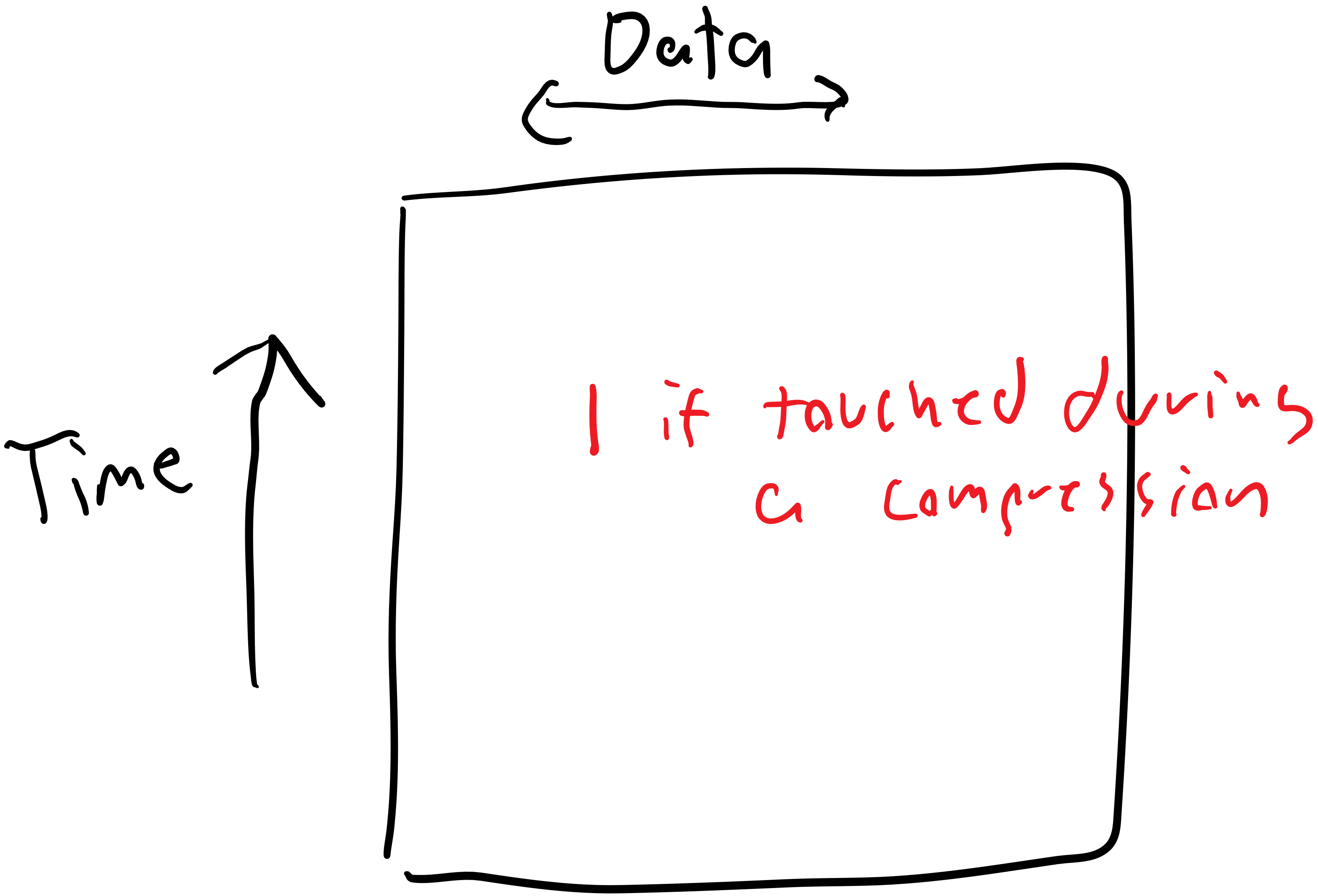
Example: Union-Find w Path Compression.



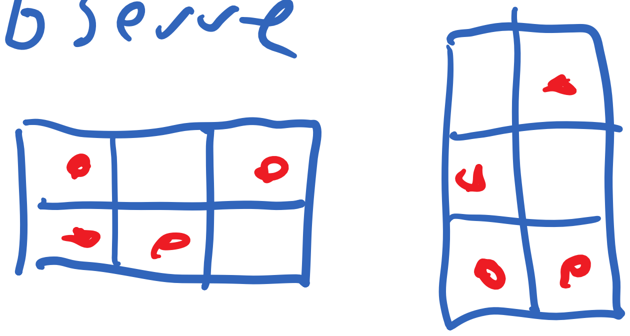
- Who is my root
- Combine

COMPRESS!

Plot: Geometric View



Observe



Forbidden

Thus

$O(n \log n)$

Technique seems powerfull

Technique seems powerfull

Yet most results are simplifications

Another Example

Incremental Restricted Voronoi

[Aronov
Bose
Demaine
Gudmundsson
I
Langerman
Smid
00]

→ A problem Having to do with Binary Search Trees

Flarb/Dilbags

[Pettie 10]

→ Geometric View

↓

Forbidden Martrix

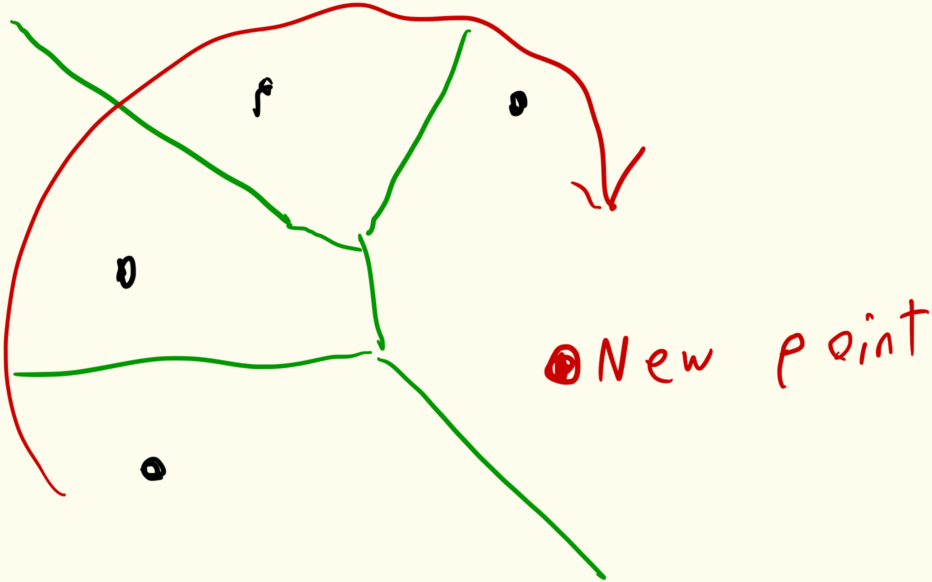
↓

Bound!

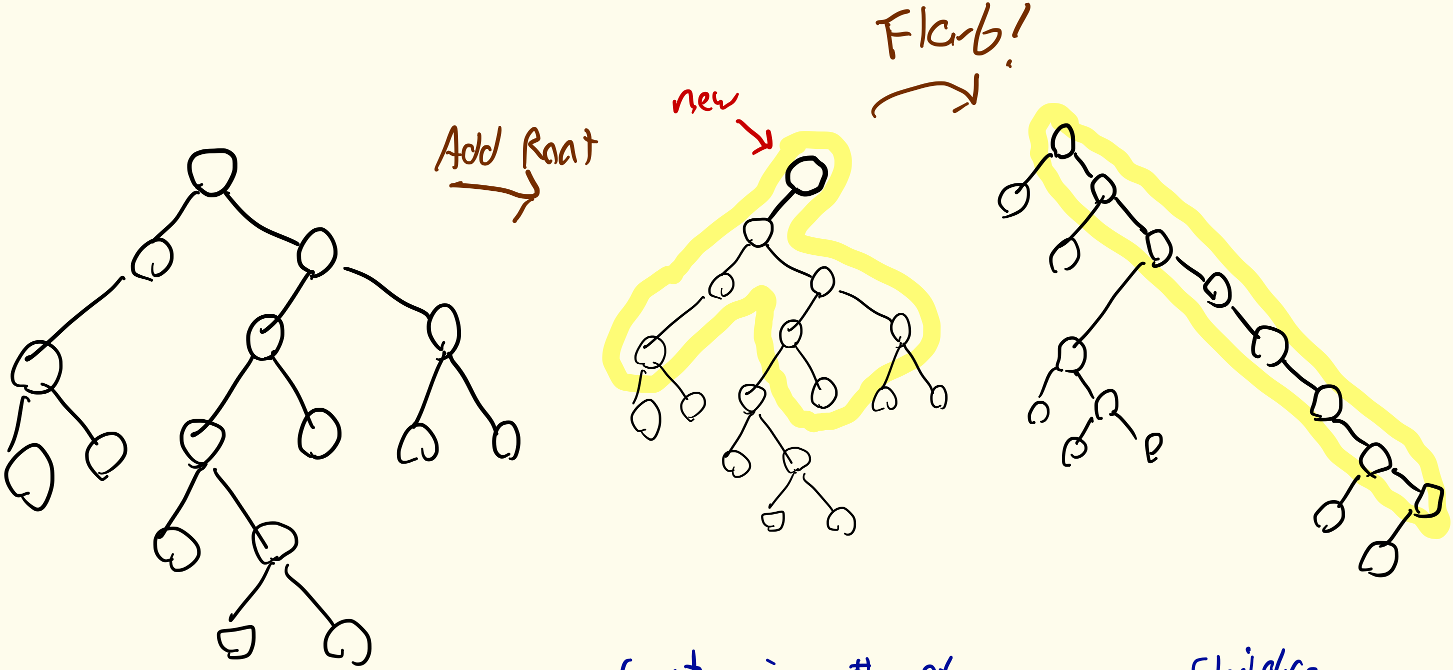
complicated →

Voronoi Diagrams

- Incremental
- Add hull points in clockwise order

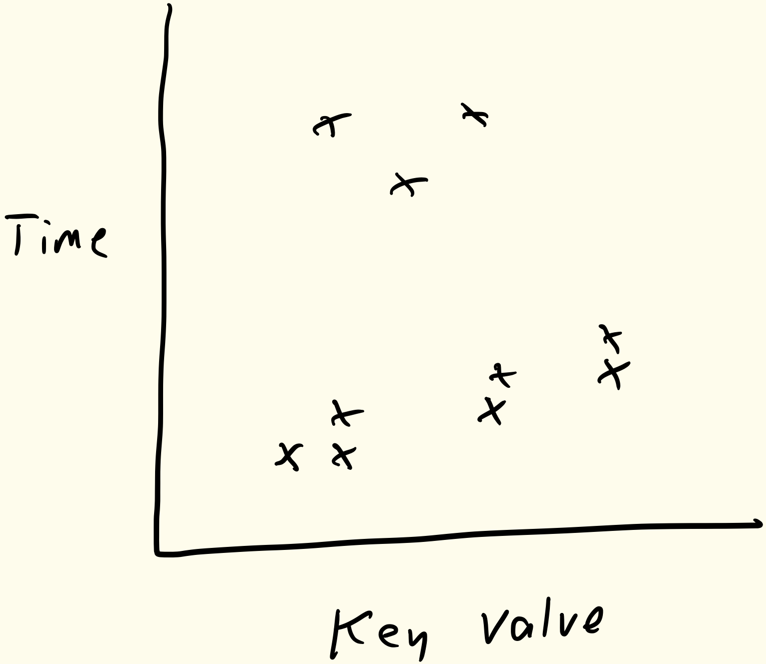


Flarb / Oilbag



Cost is # of children who have had their parent change

Geometric View



X = parent changed

Pettie's Observation

No

	X	X
		X
X		

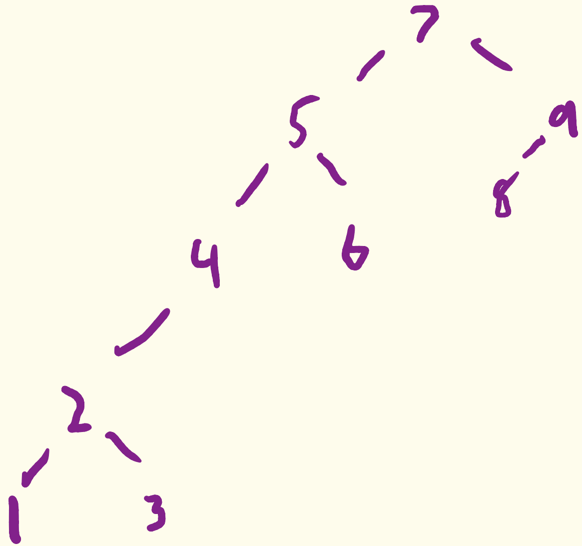
as a submatrix

$\rightarrow$ # X's = $O(N \log N)$

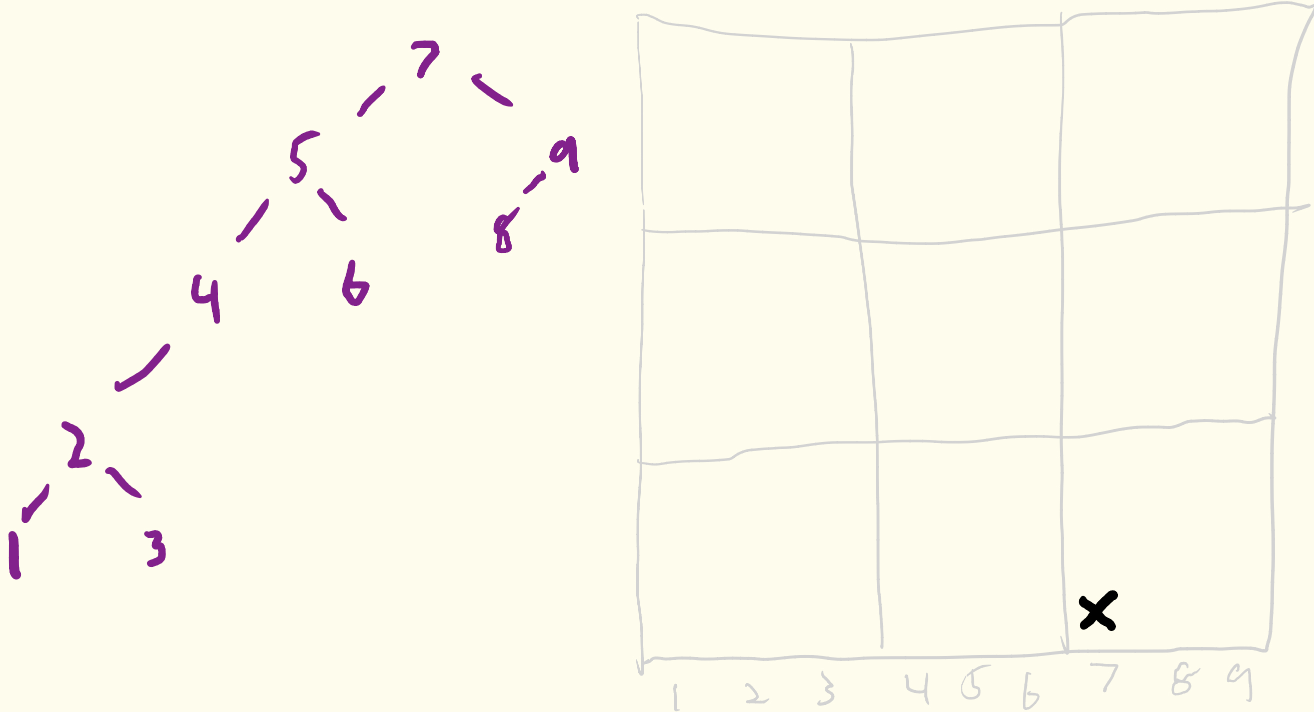
Pattern Avoidence in BSTs

Forbidden Pattern in the search set $\longrightarrow$ Bound on algorithm to make a satisfied set

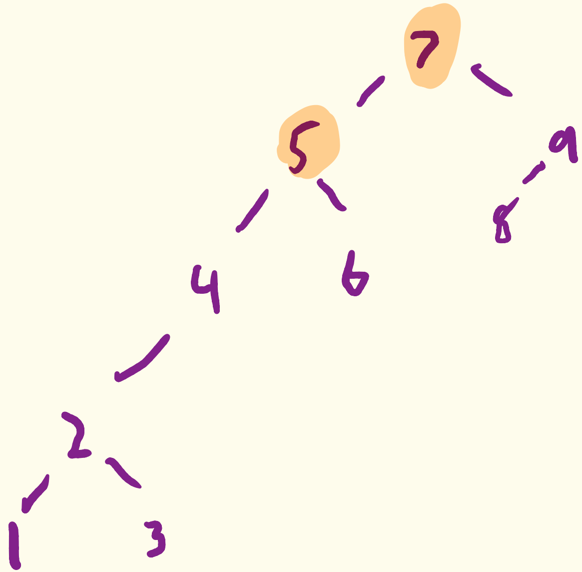
Preorder conjecture

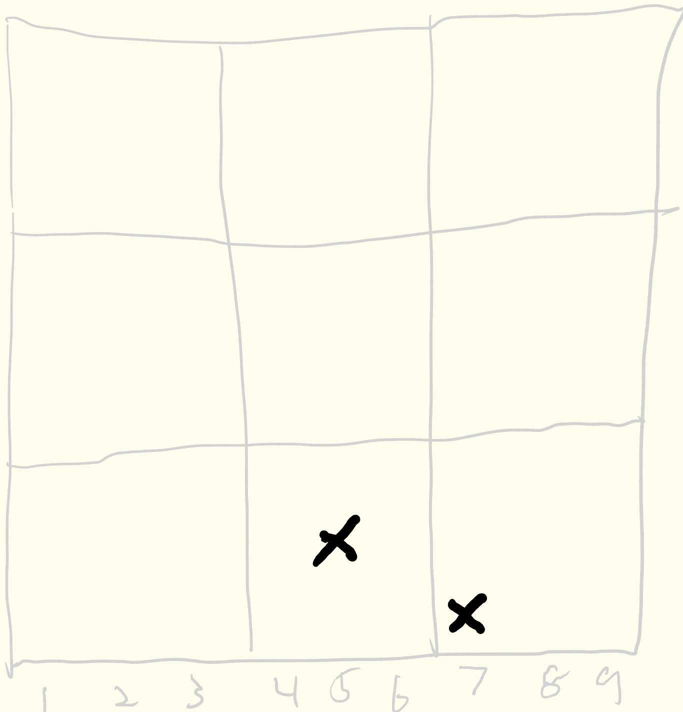


Preorder Conjecture

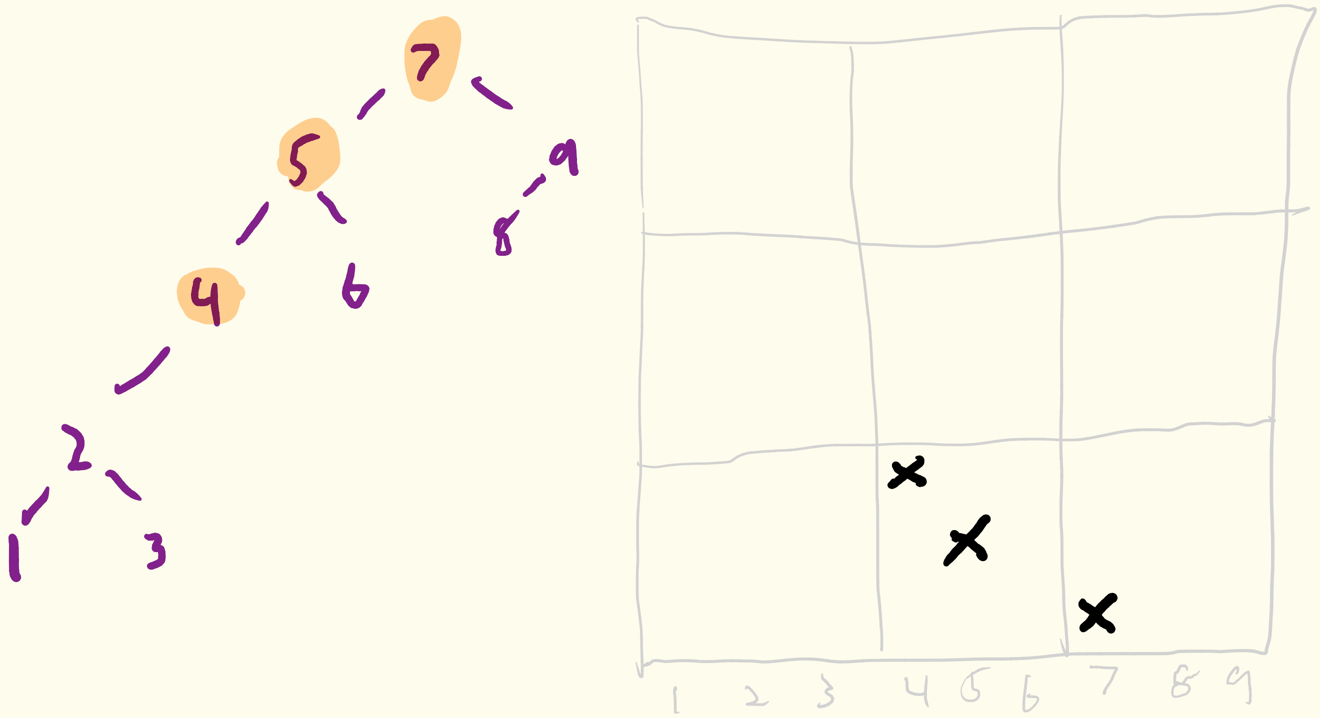


Preorder Conjecture

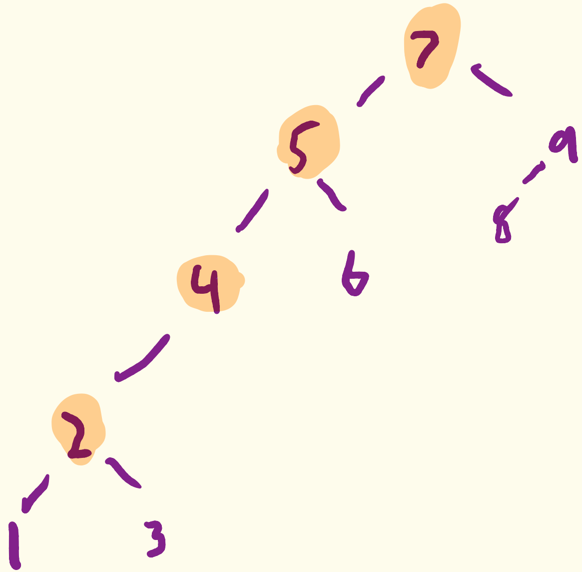


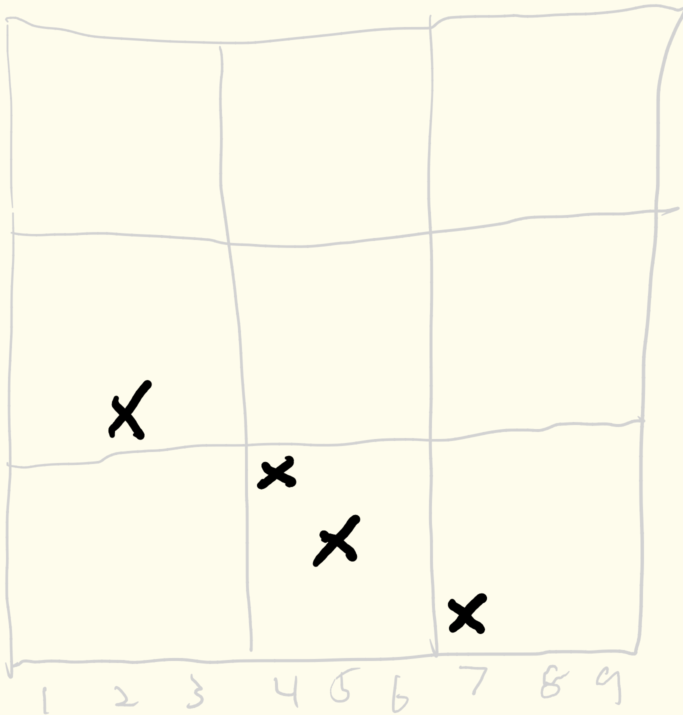


Preorder Conjecture

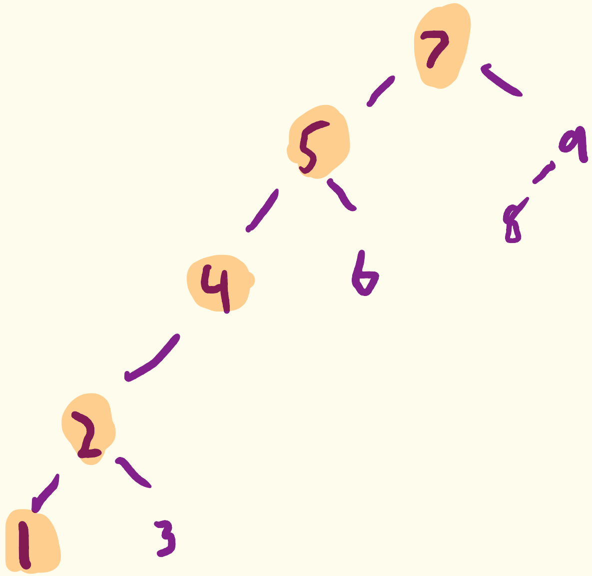


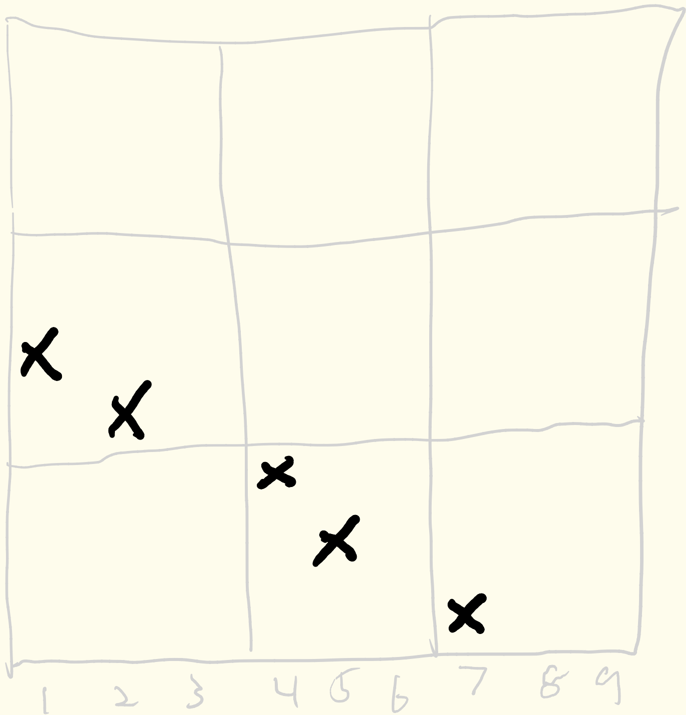
Preorder Conjecture



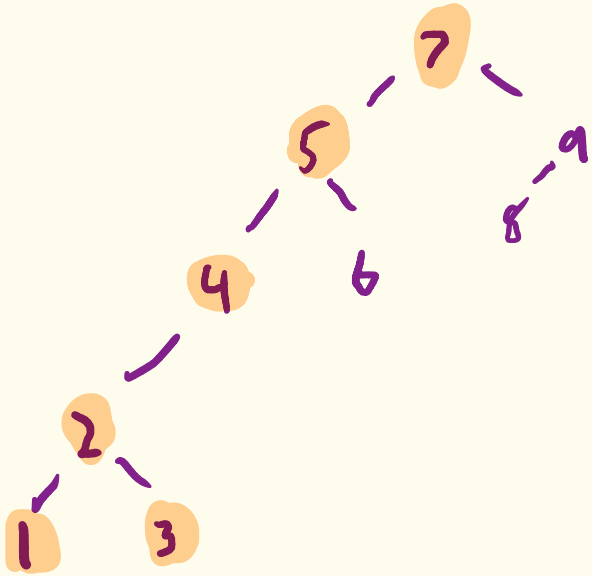


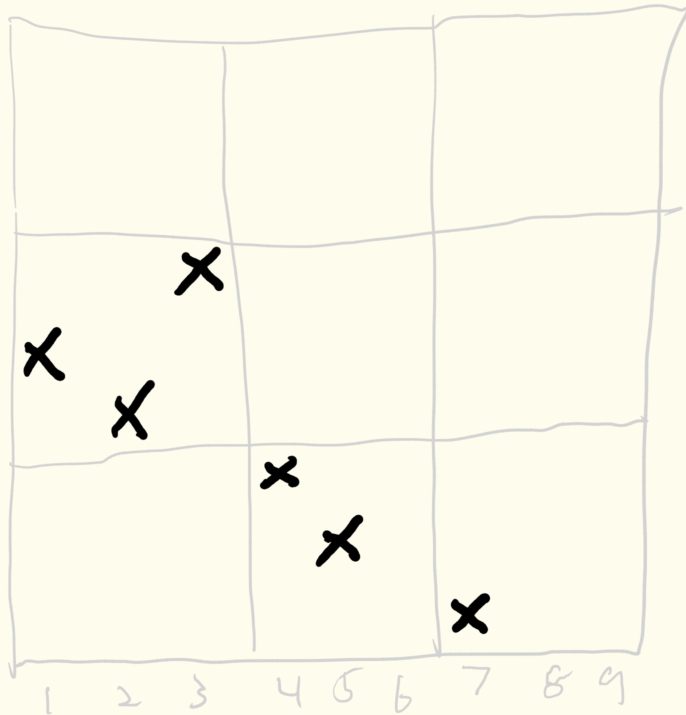
Preorder Conjecture



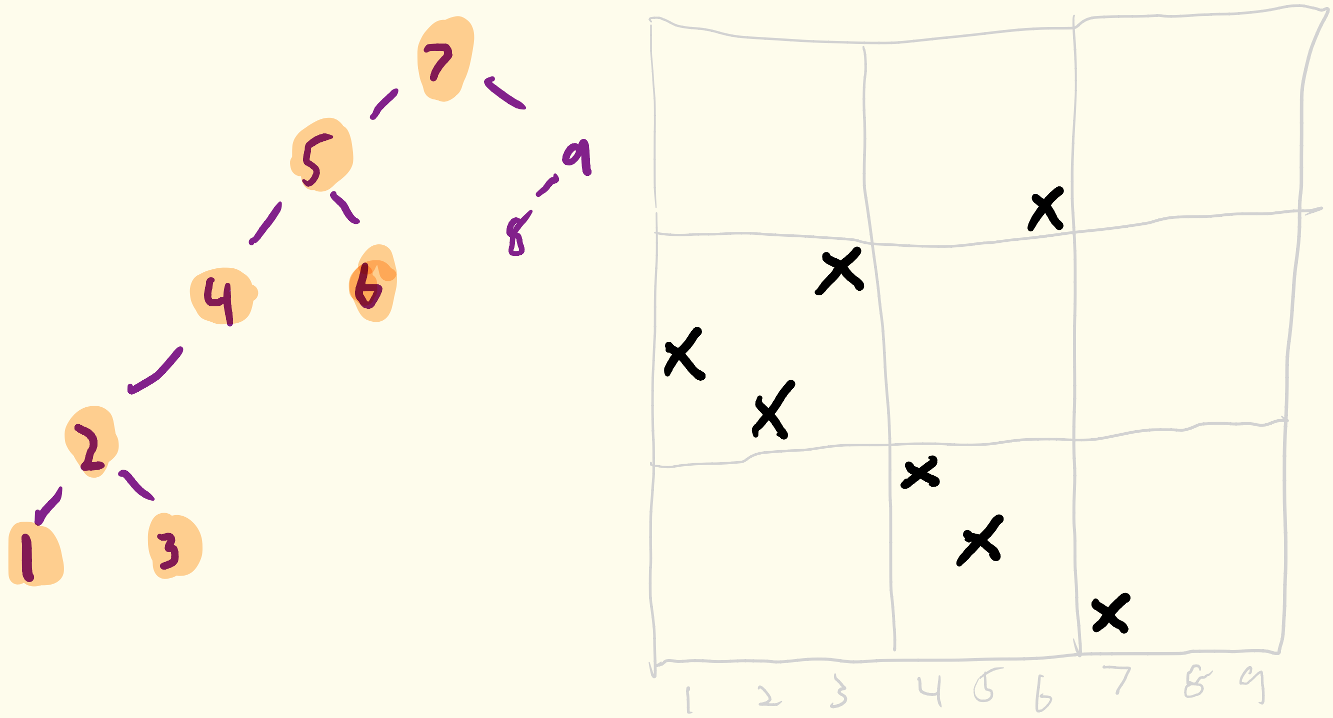


Preorder Conjecture

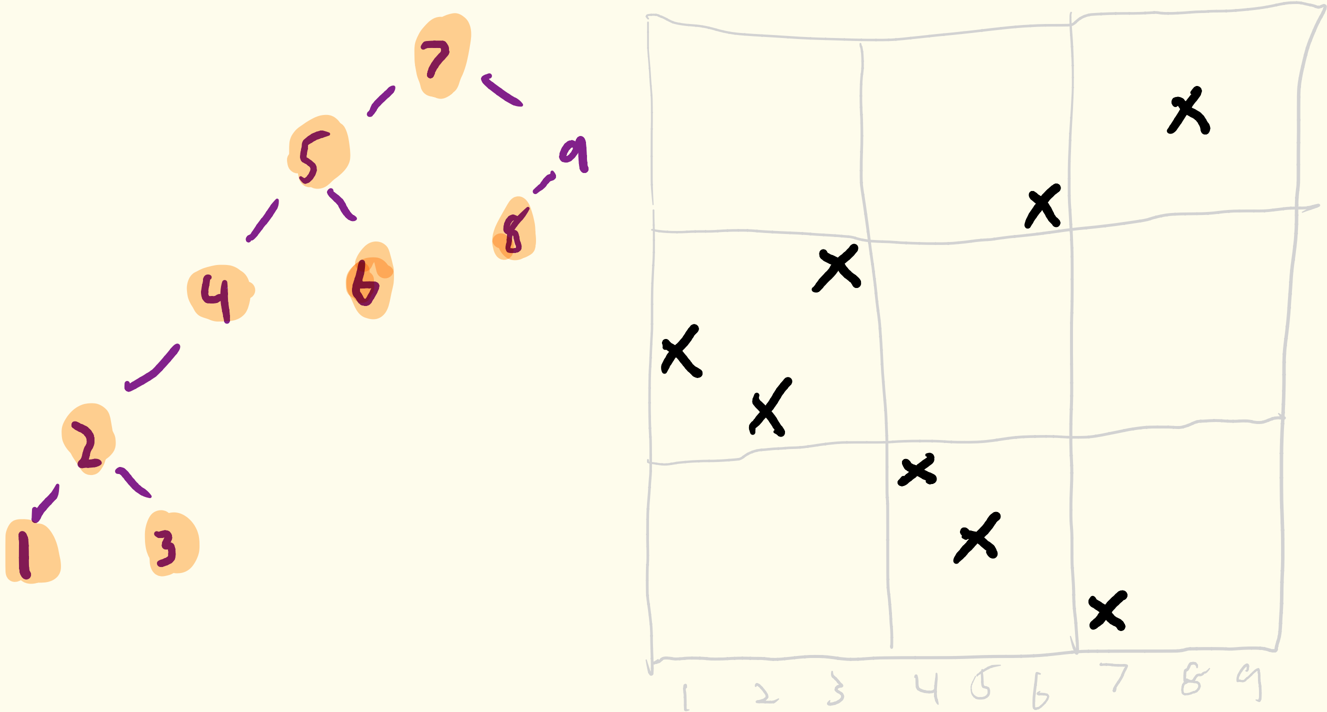




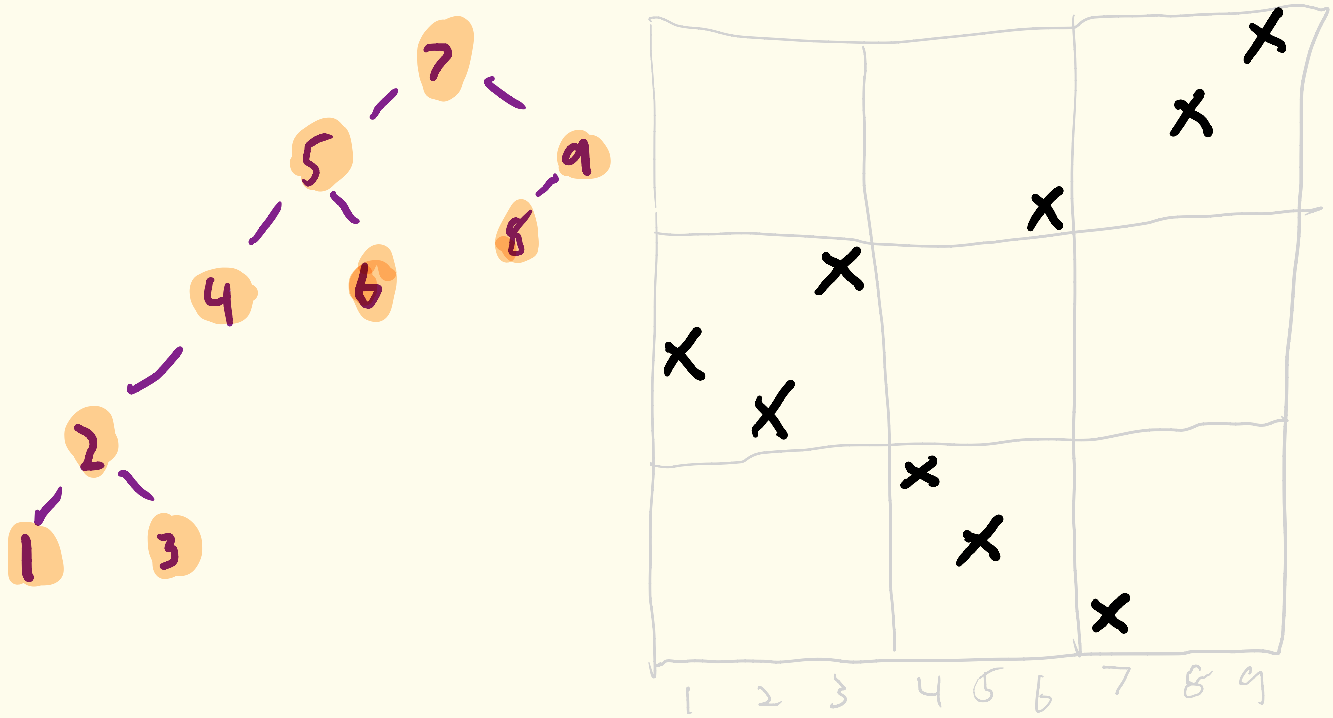
Preorder Conjecture



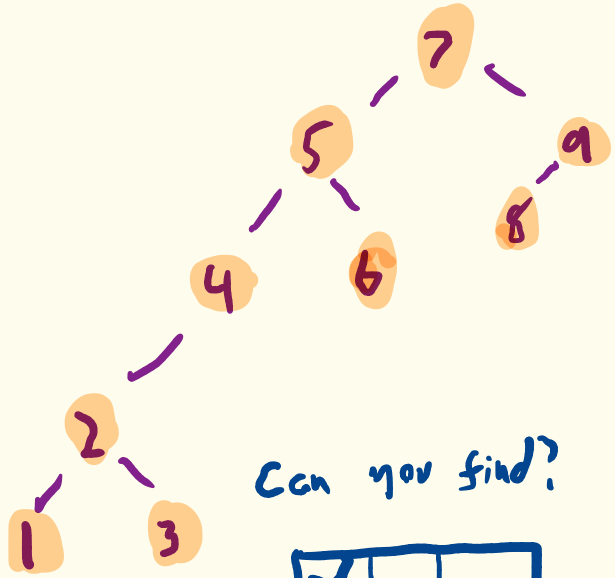
Preorder Conjecture



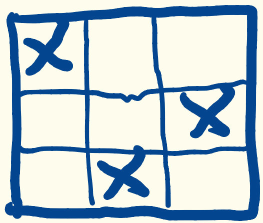
Preorder Conjecture



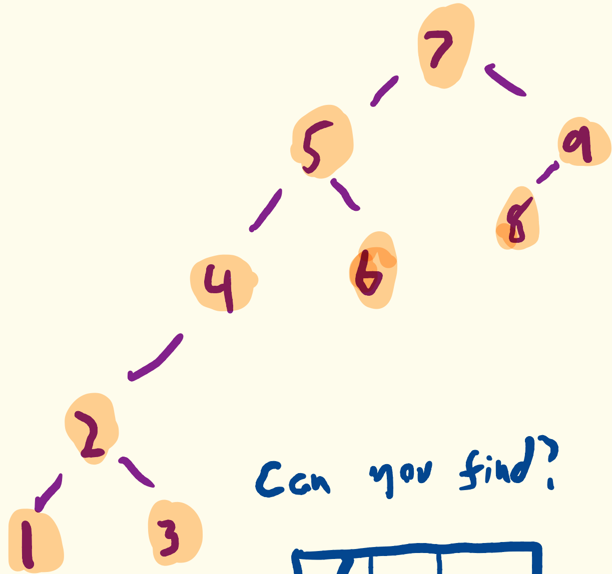
Preorder Conjecture



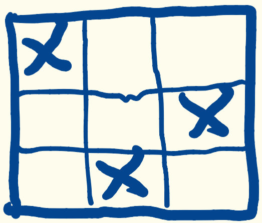
Can you find?

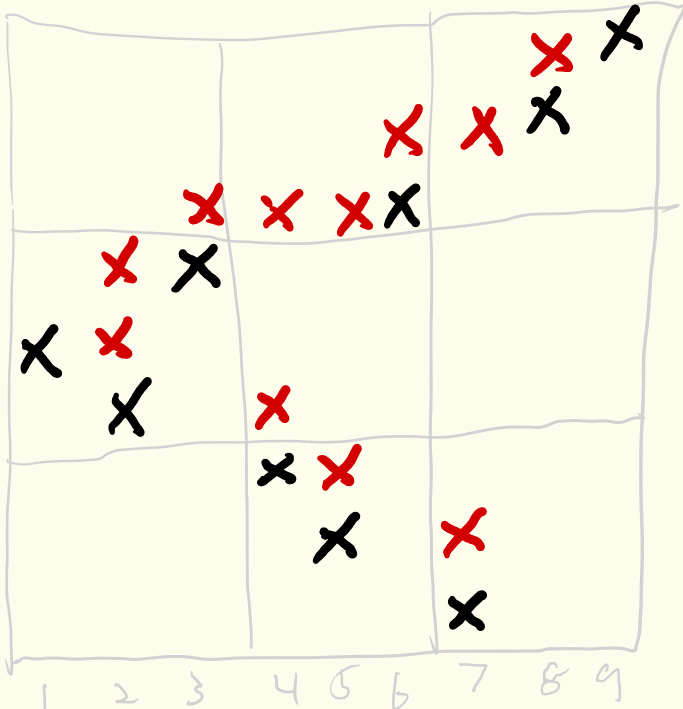


Preorder Conjecture

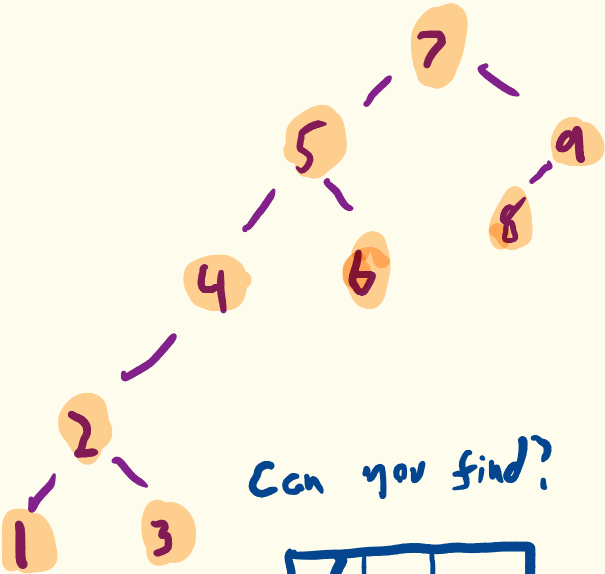


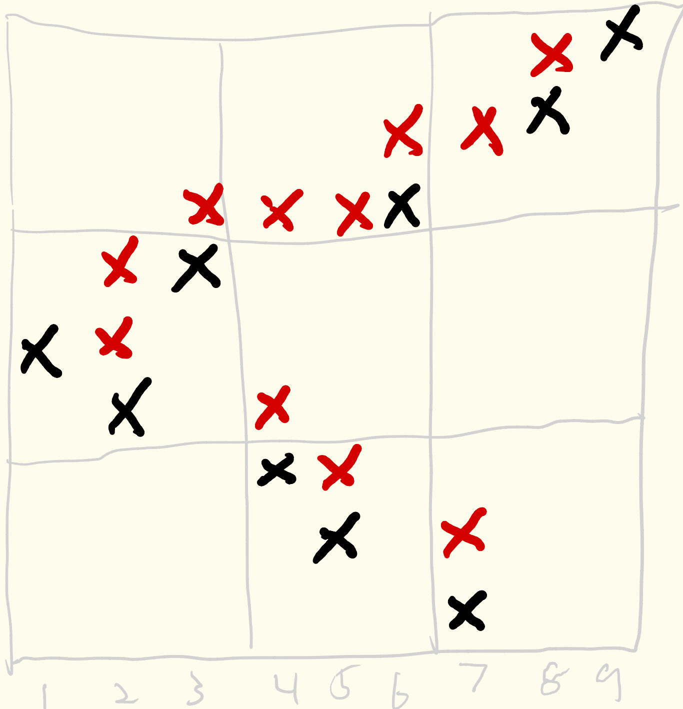
Can you find?



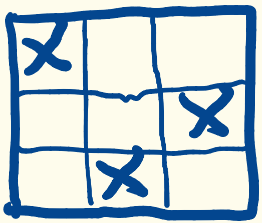


Dynamic Opt → Preorder takes O(N) time

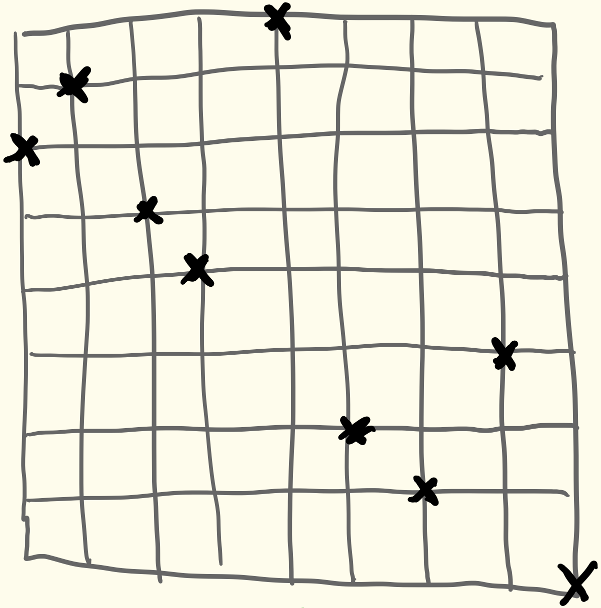




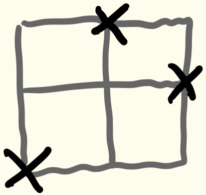
Can you find?



Pattern Avoidance in BST



No submatrix



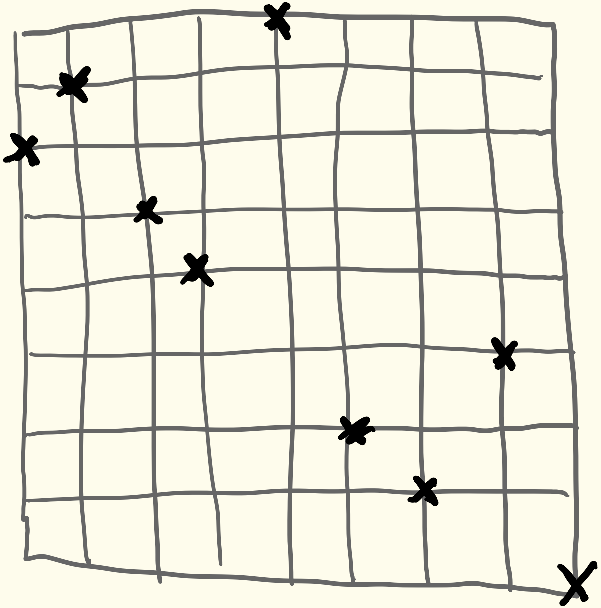
→ Greedy's Runtime is

$$\leq n 2^{\alpha(n)^{O(1)}}$$

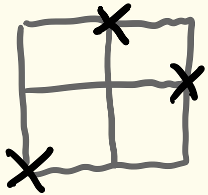
[Chalermsook
Goswami
Kozma
Mehlhorn
Saranurak 2015]

Pattern Avoidance in BST

Generalize!



No submatrix

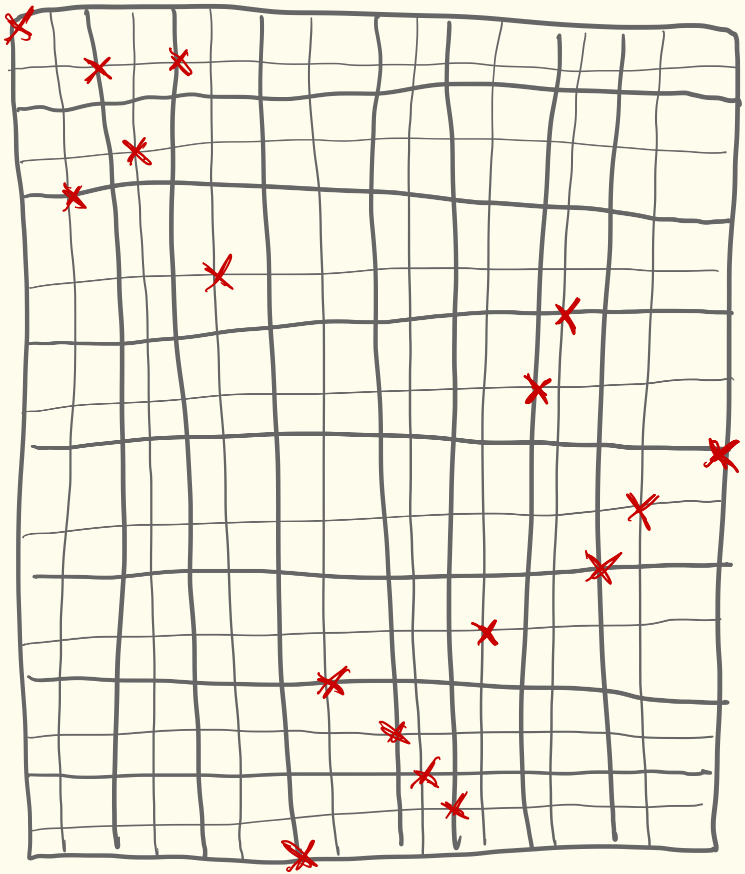


any $k \times k$ permutation

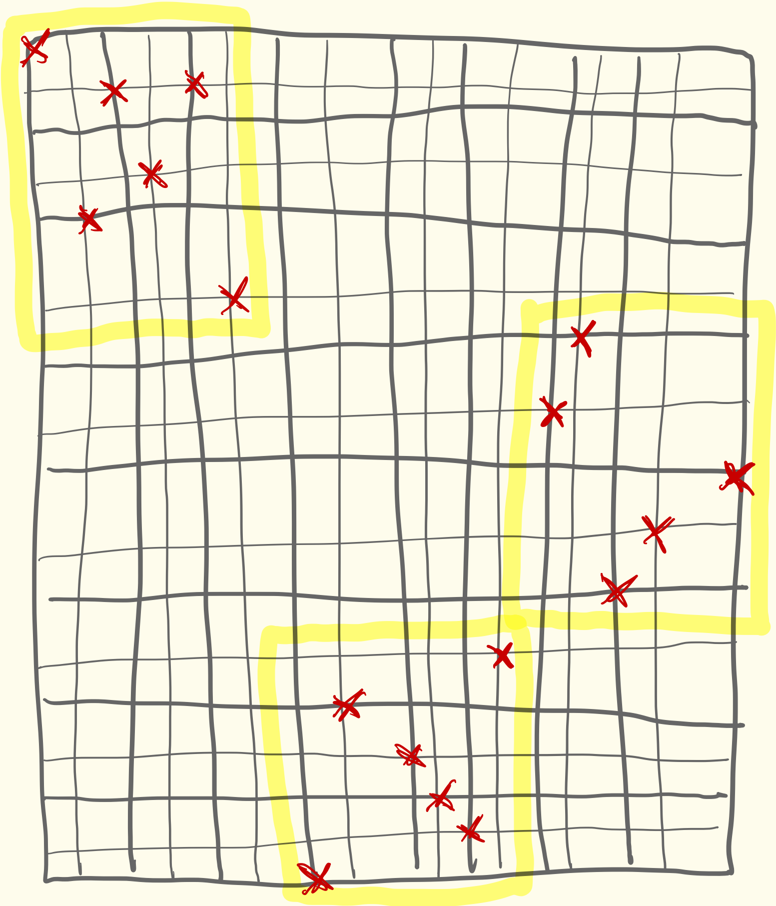
→ Greedy's runtime is

$$n2^{\alpha(n)^{\cancel{O(1)}\ O(k)}}$$

Recursive Decompositions

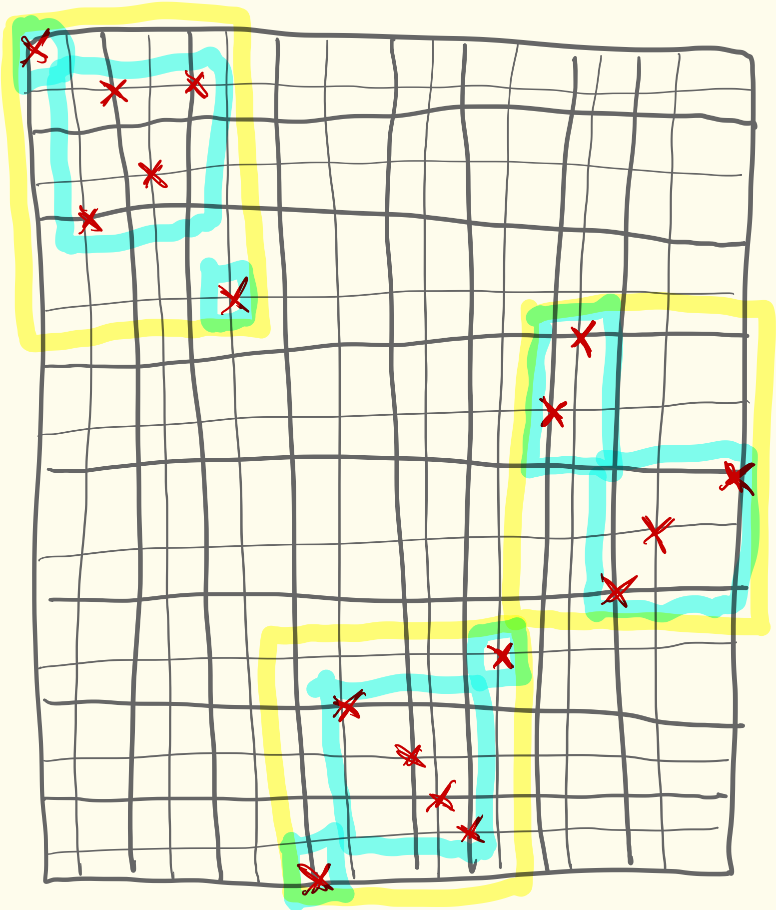


Recursive Decompositions



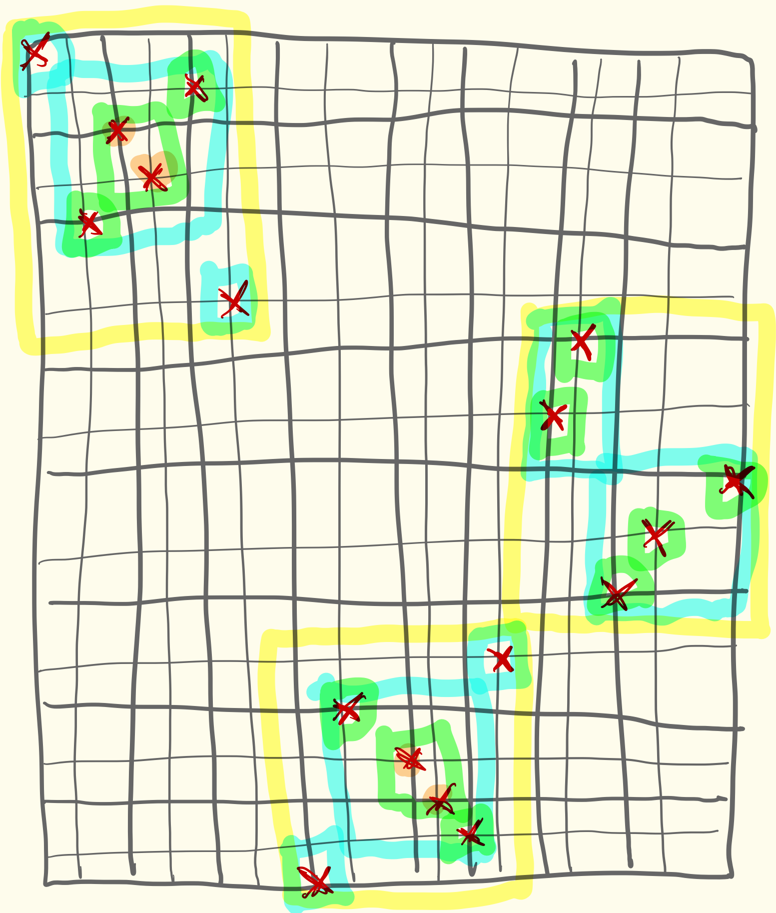
3 - decomposable

Recursive Decompositions



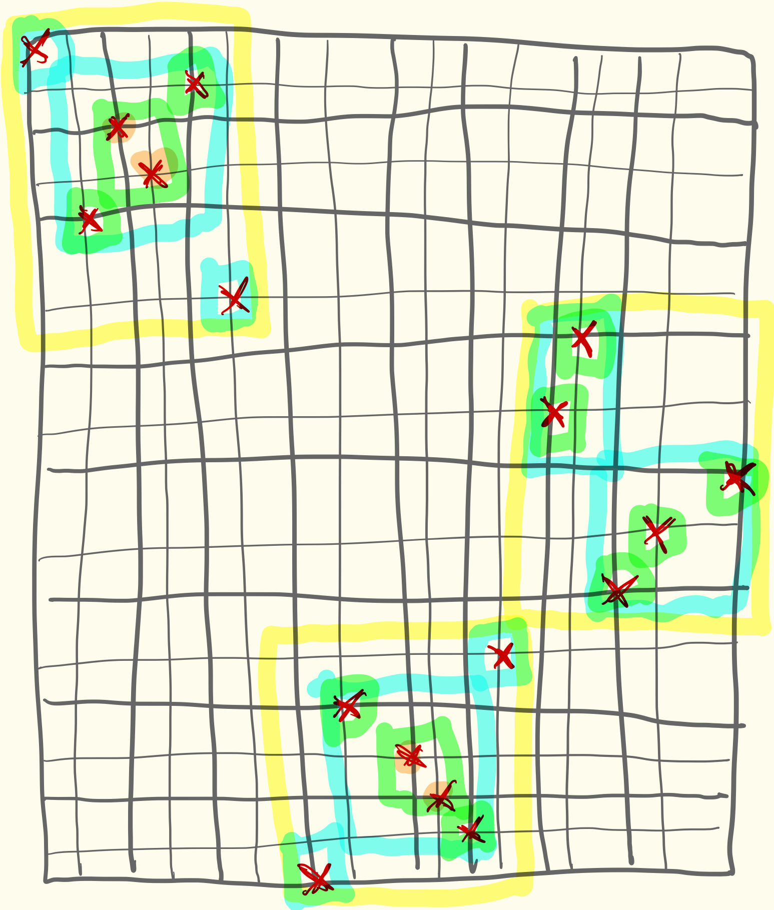
3 - decomposable

Recursive Decompositions



recursively
3 - decomposable

Recursive Decompositions



recursively 3 - decomposable

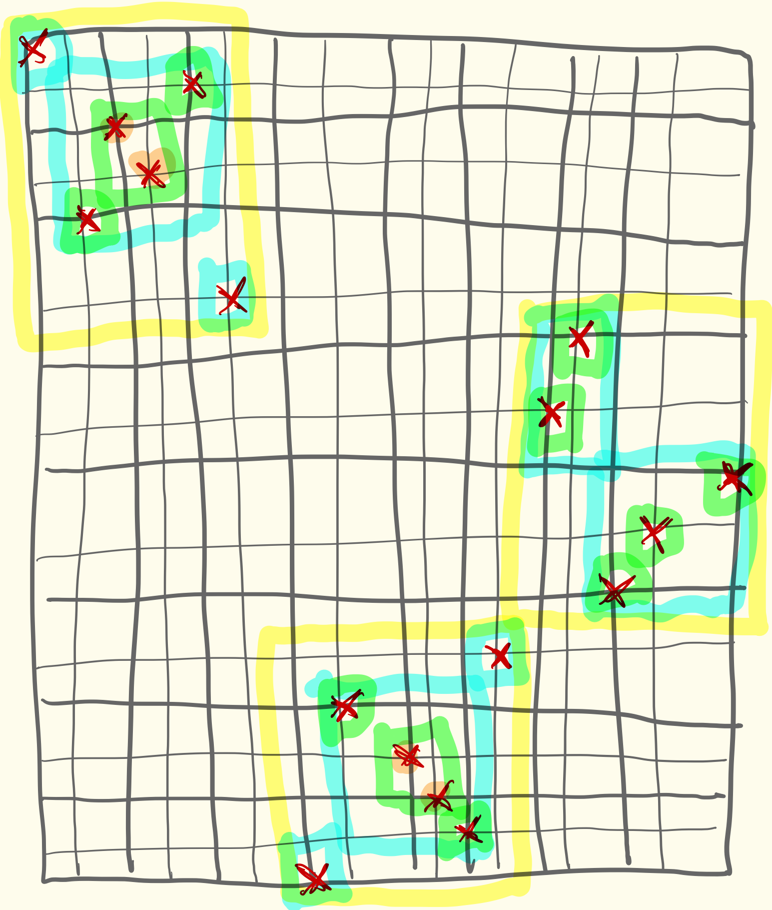
Recursively k-decomposable

→ Greedy* adds:

$$n\,2^{O(k^2)}$$

[Chalermsook
Goswami
Kozma
Melhorn
Saranurak 2015]

Recursive Decompositions



recursively 3-decomposable

Recursively k-decomposable

→ Greedy* adds:

$\cancel{n^2}$ $\cancel{O(k^2)}$

$O(n \log k)$

[Chalermsook
Goswami
Kozma
Melhorn
Saranurak 2015]

← Same 2017
using
[Iacono Langerman
SODA 2017]

More Applications?

Possible Future Directions

- Geometry of other fundamental DS's. E.G. Heaps
- Modeling other models Geometrically?
- Other forbidden matrix applications e.g. compact encoding of geometric objects.

The End

ULB Algorithms Group Has Multiple Postdocs!

Gwen Joret

Jean Cardinal

Elena Khramtcova

John Iacono

Aurélien Ooms

Sam Fiorini

Till Miltzow

Stefan Langerman