# The analysis of self-adjusting data-structures
# Results, Conjectures, Alternatives

John Iacono

Polytechnic University

Brooklyn
NYC
NY
USA

# Outline

- Trees
- Heaps

# Binary Search Trees - Prehistory

- AVL Trees
- "Optimal" trees
- Level-linked trees
- Finger trees

# AVL Trees [A-V, L 62]

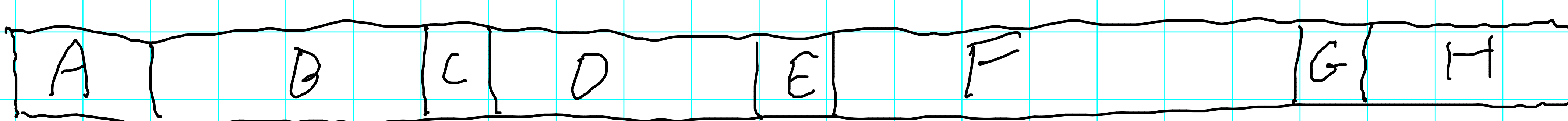
- $O(\log n)$ Ins/Del/Search
- One bit per node of structural info
- Worst-case asymptotically optimal
- Many many alternatives (e.g. left-leaning red-black trees) [Sedgewick 08]

# "Optimal trees" [e.g. Knuth 85]

- Given $n$ items each with a weight $w_i$, $\sum w_i = 1$, in an optimal tree the cost to search for item $i$ is $O(\log \frac{1}{w_i})$
- Optimal if searches are drawn independently at random with probability equal to the weights
- Entropy

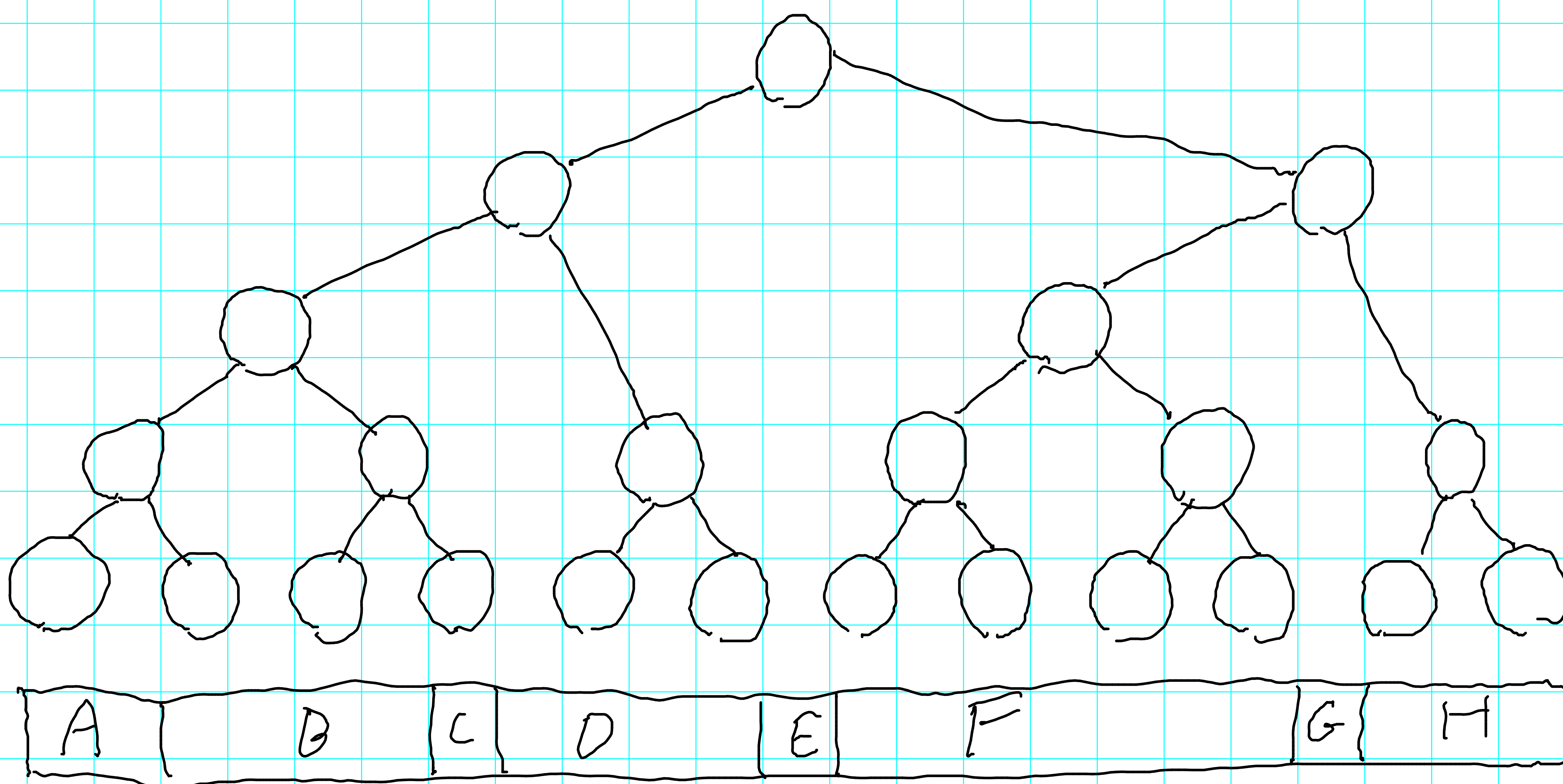
# Optimal Trees

## Simple Construction



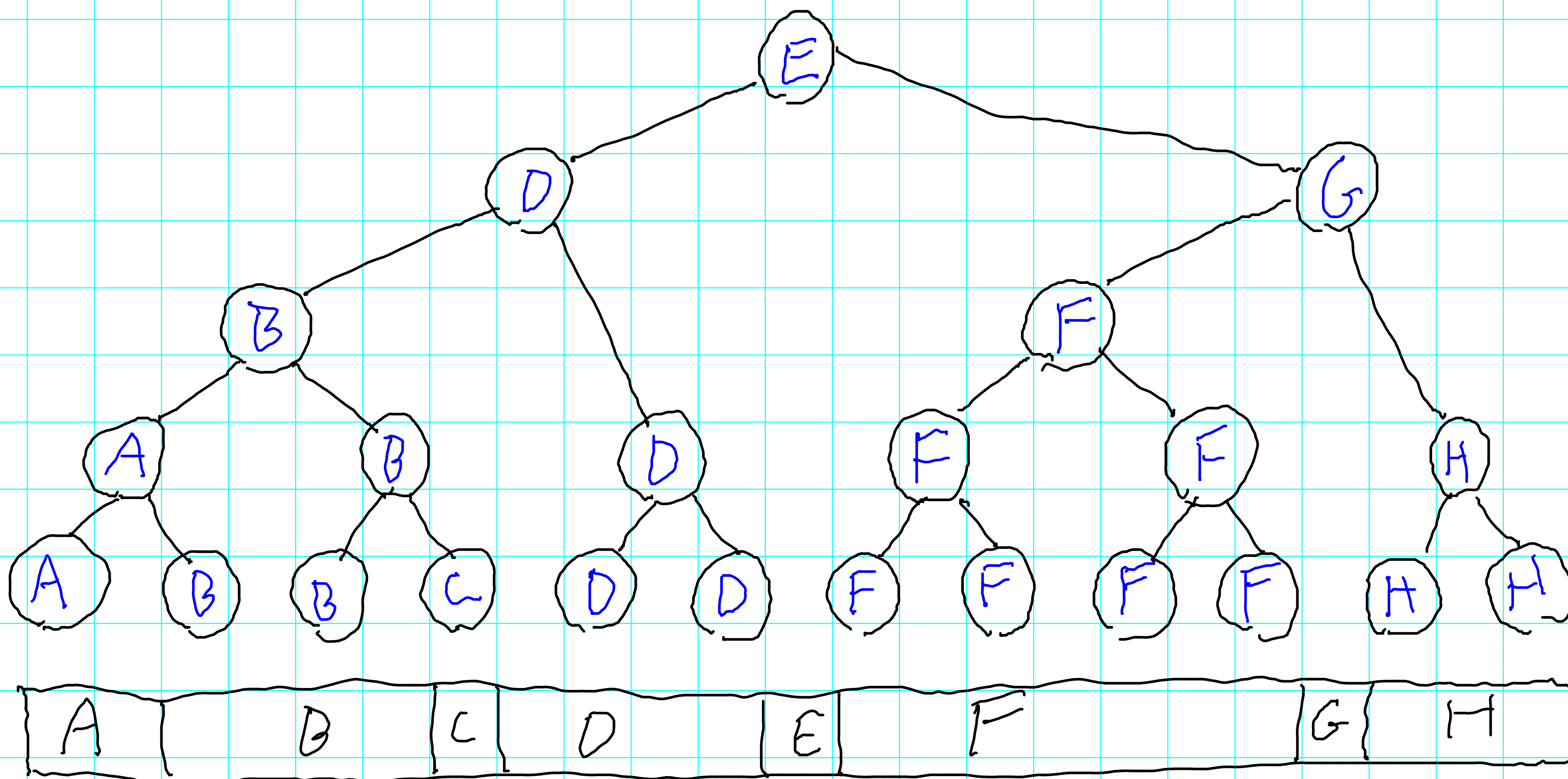
# Optimal Trees

## Simple Construction



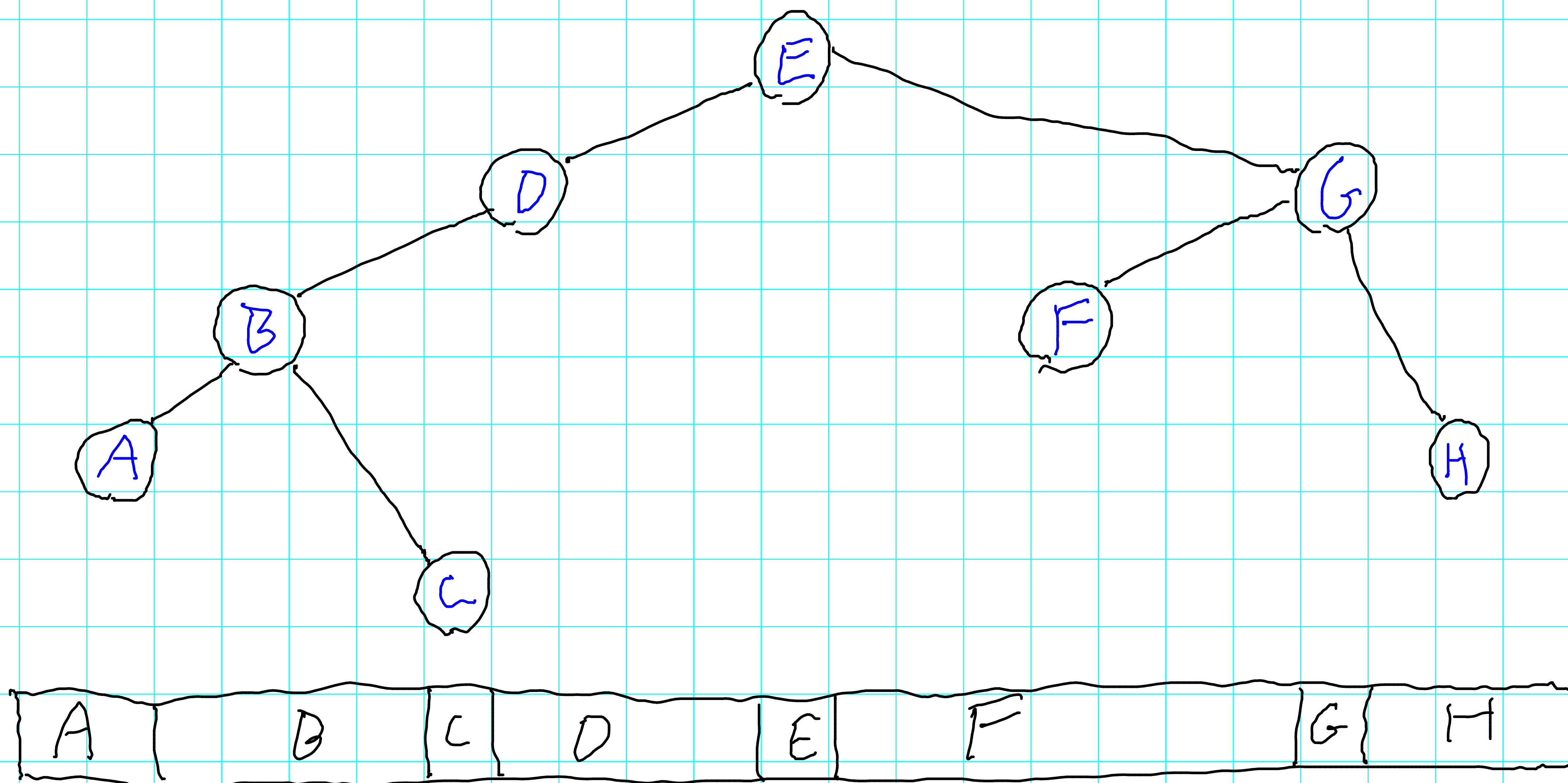
# Optimal Trees

## Simple Construction



# Optimal Trees

## Simple Construction



# Level-linked Trees

[Brown Tarjan 80]

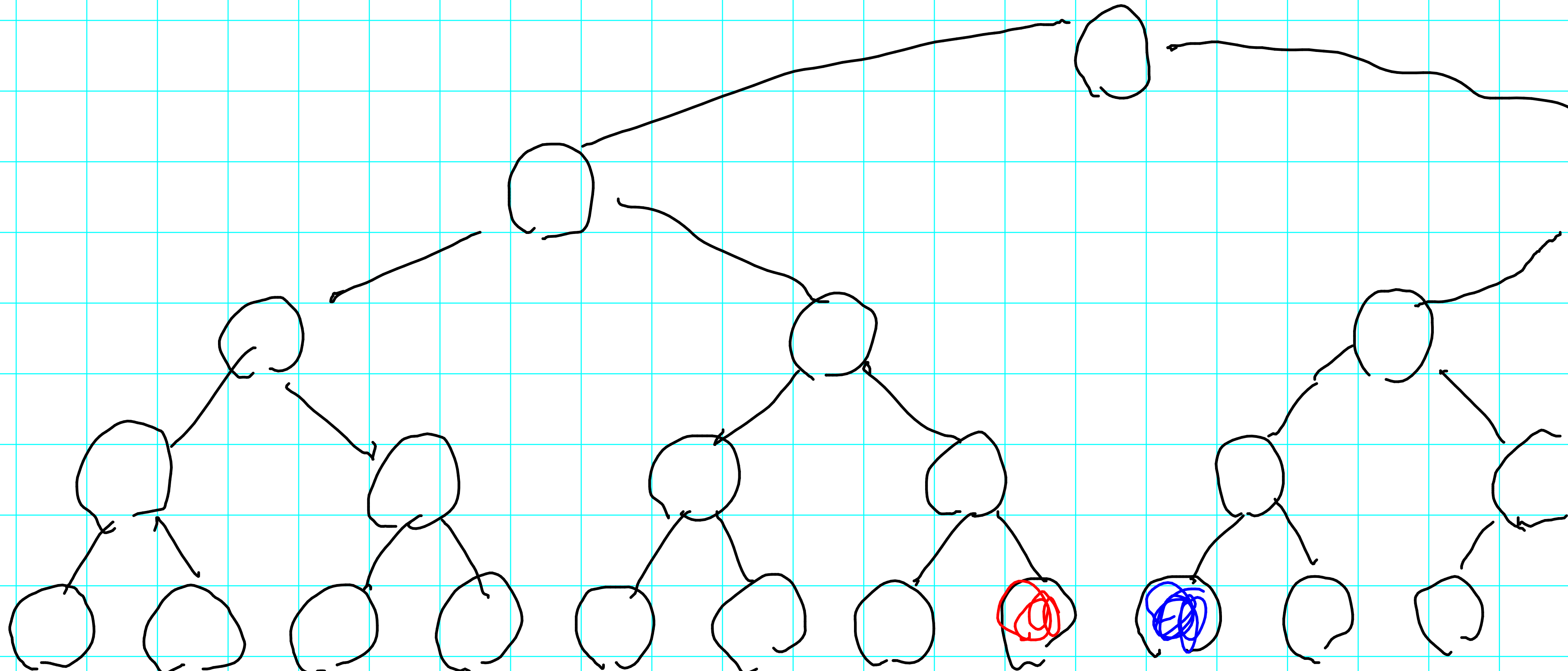
- Idea: Searches should be fast if the key being searched is "close" to the previous search

# Level-linked Trees

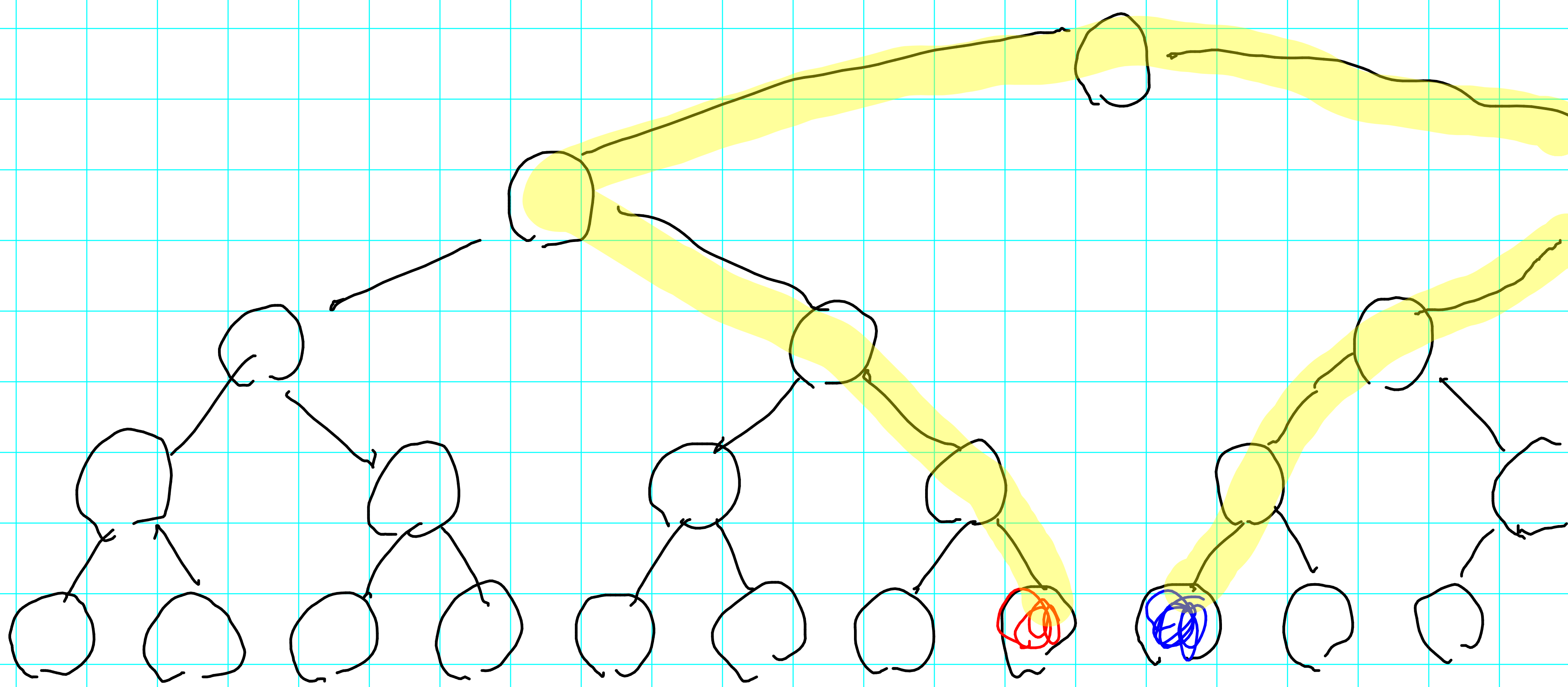
- Idea: Searches should be fast if the key being searched is "close" to the previous search

- Search takes time $O(\lg K)$
  $K$ = number of keys from last search to current search

# Level-linked Trees

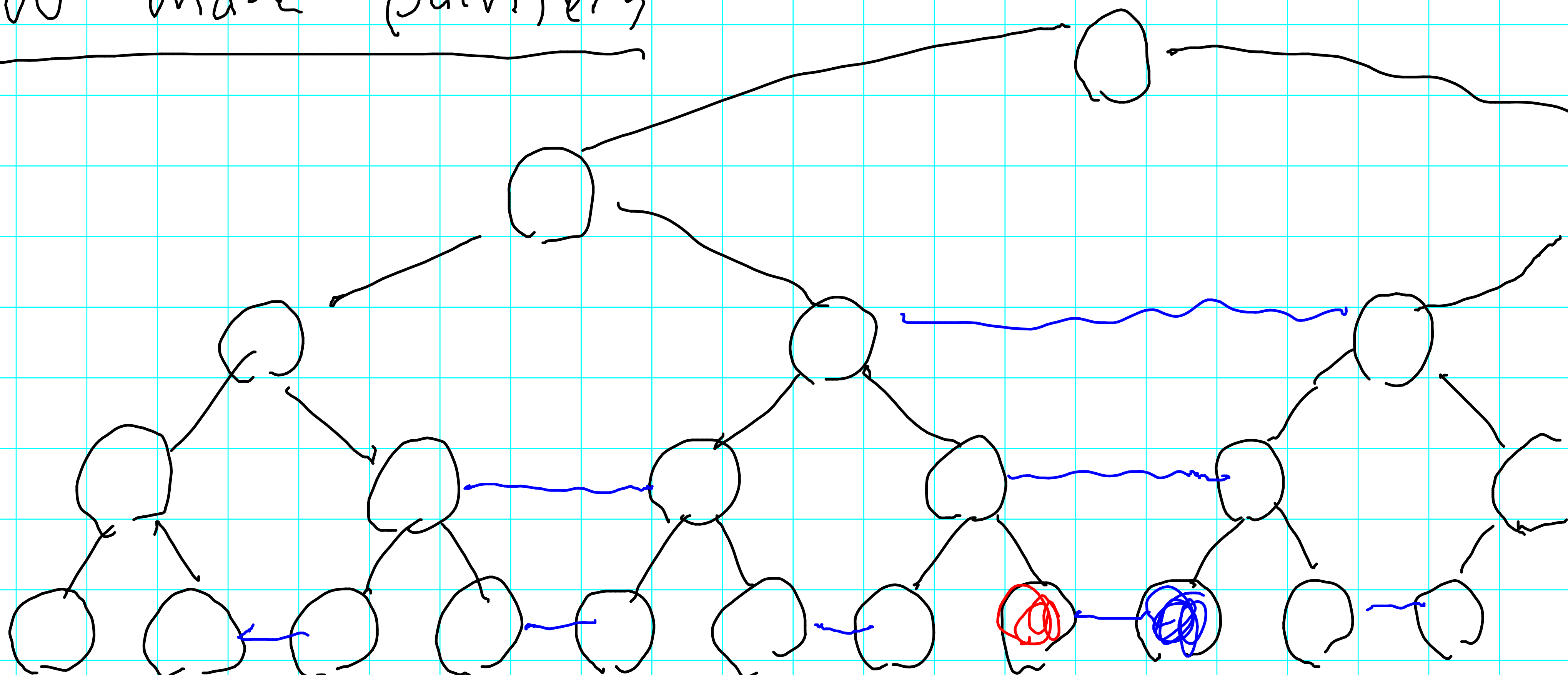


# Level-linked Trees



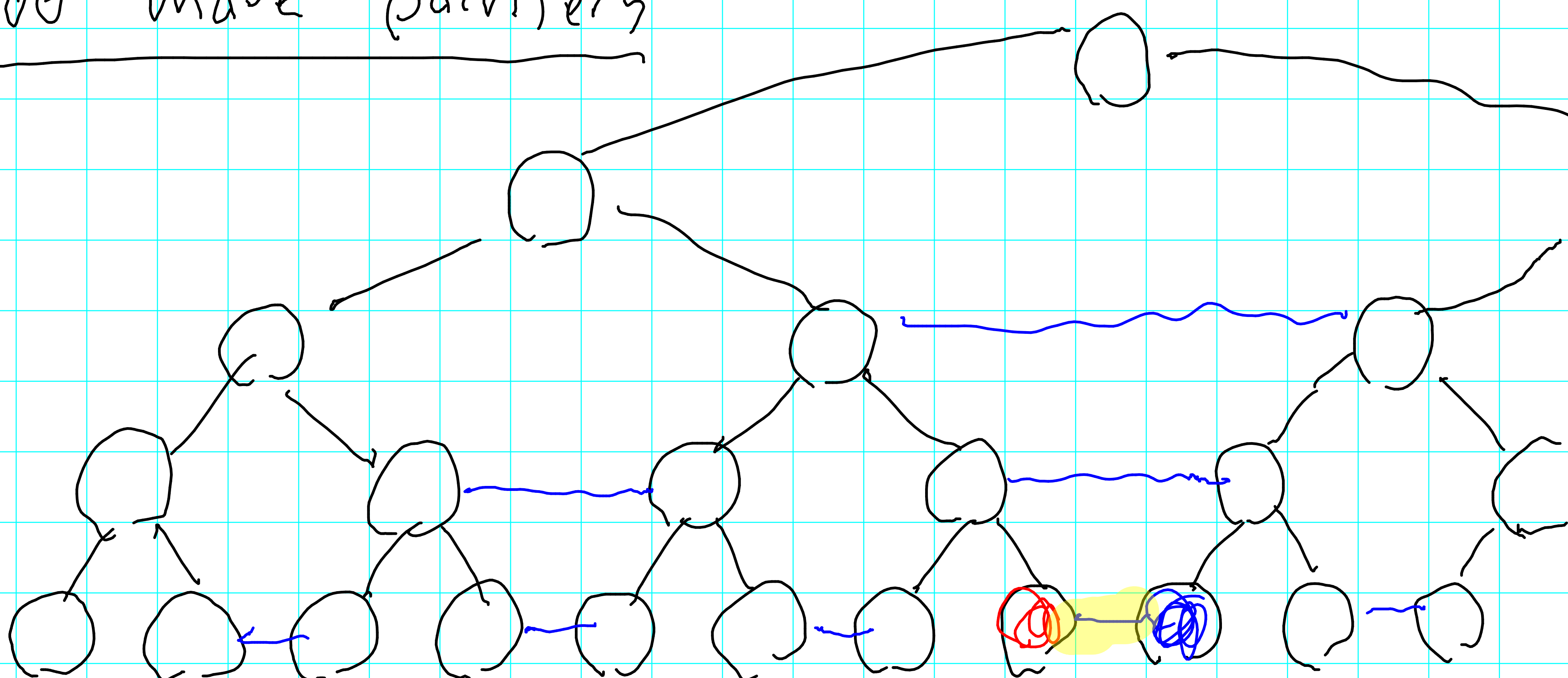
# Level-linked Trees

Add more pointers



# Level-linked Trees

Add more pointers



# Finger Trees

Guibas
McCreight
Plass
Roberts
77

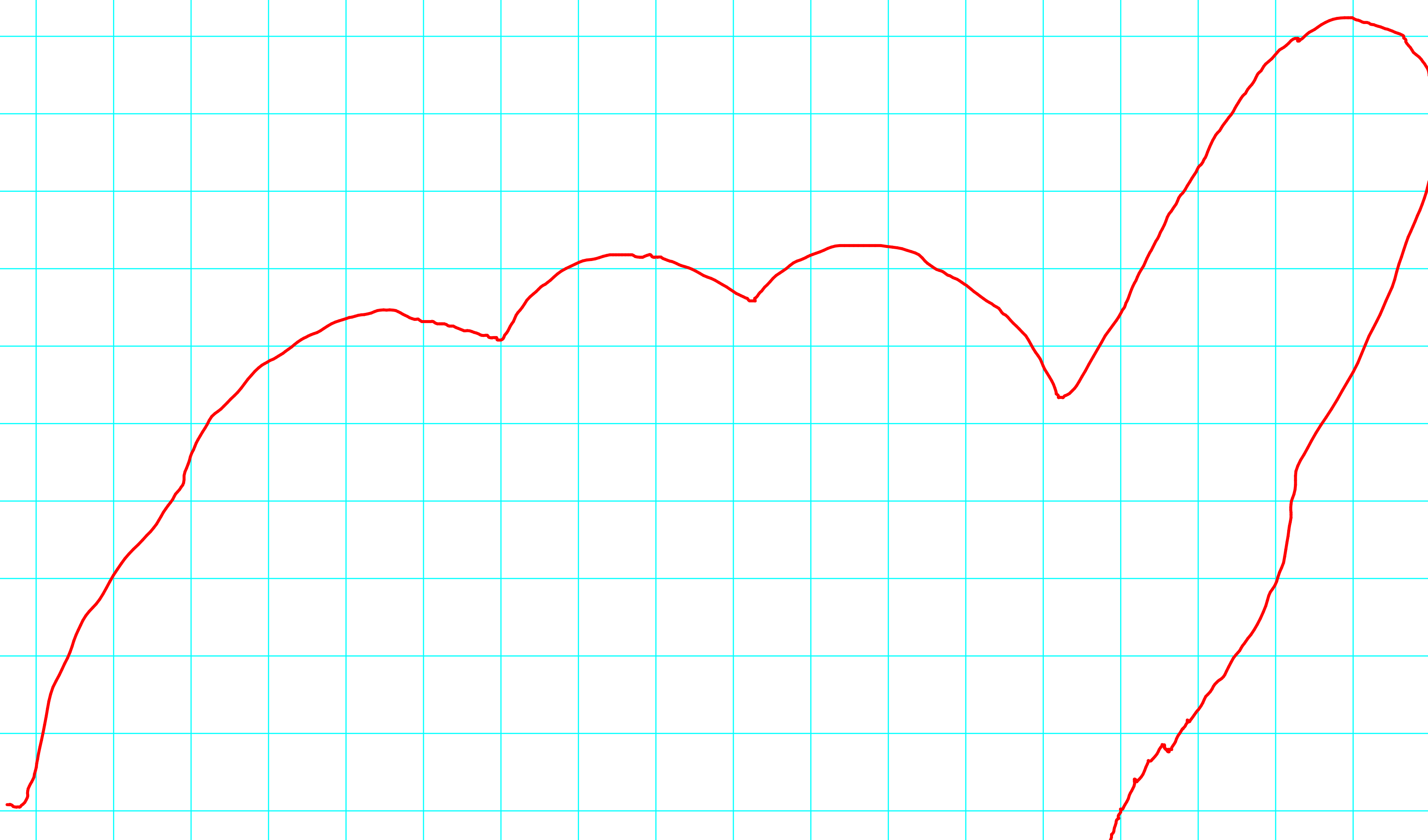
Idea:

1 3 10 12 17 31 54 63 64 65 77 81

# Finger Trees

Idea:

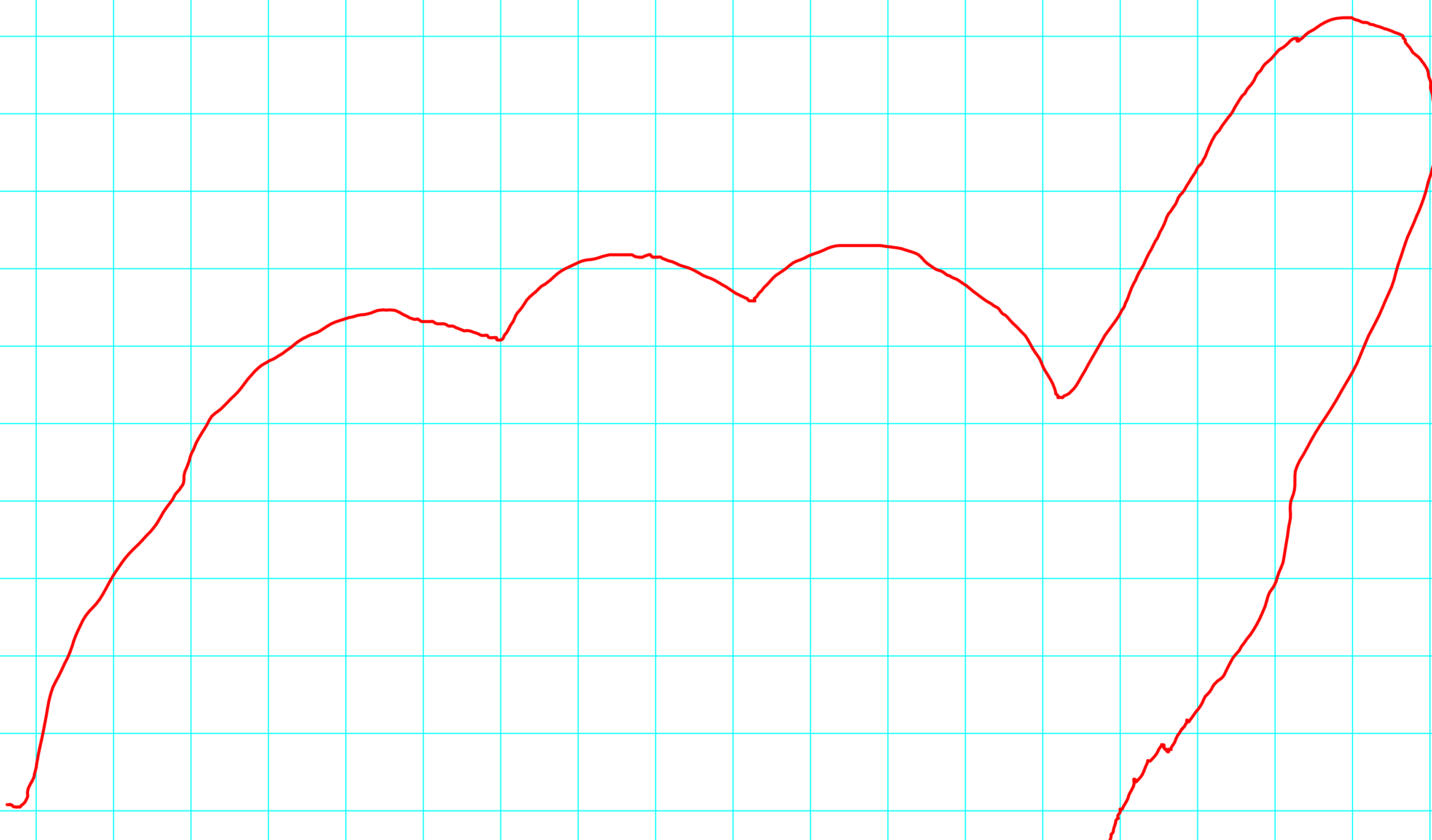
1 3 10 12 17 31 54 63 64 65 77 81



# Finger Trees

Idea:

$$1 \quad 3 \quad \overbrace{10 \quad 12 \quad 17 \quad 31 \quad 54 \quad 63 \quad 64}^{O(\log 7)} \quad 65 \quad 77 \quad 81$$



Enough Prehistory

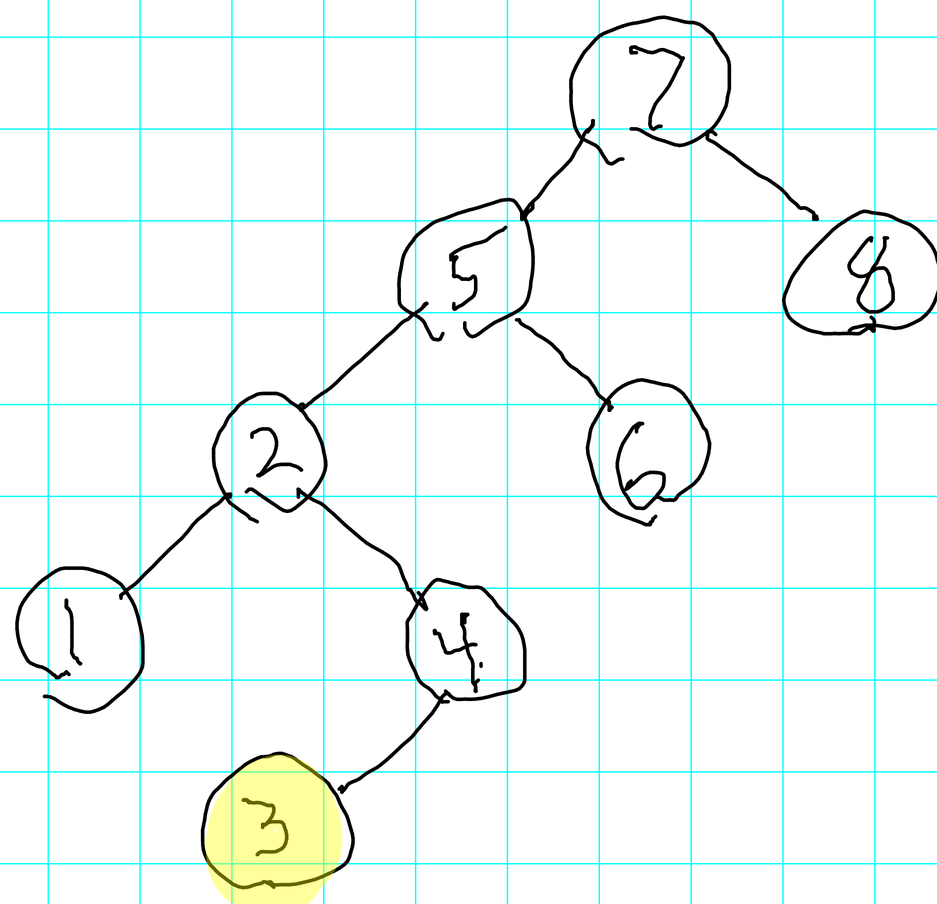
# Splay Trees

[sleator + tarjan 85]

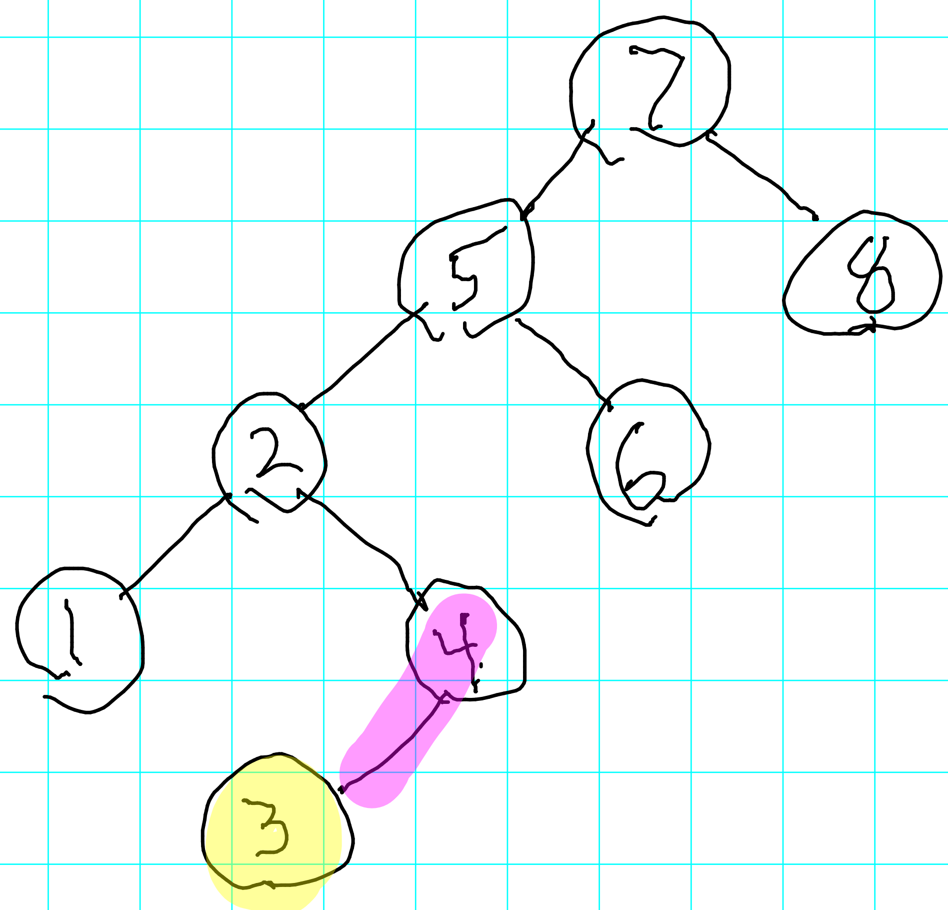
# Splay Trees

- No balance information stored
- No structural constraints
- Runtimes are amortized
- IDEA: Bring the item you search to the root

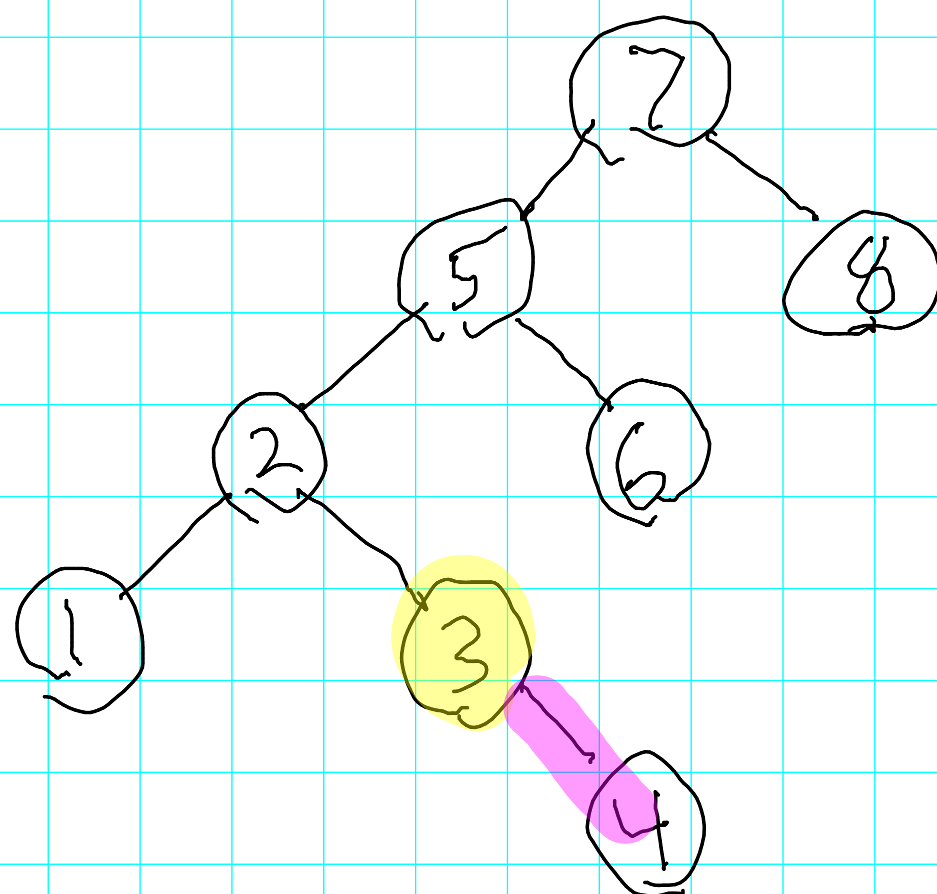
# Move-to-Root: First Idea



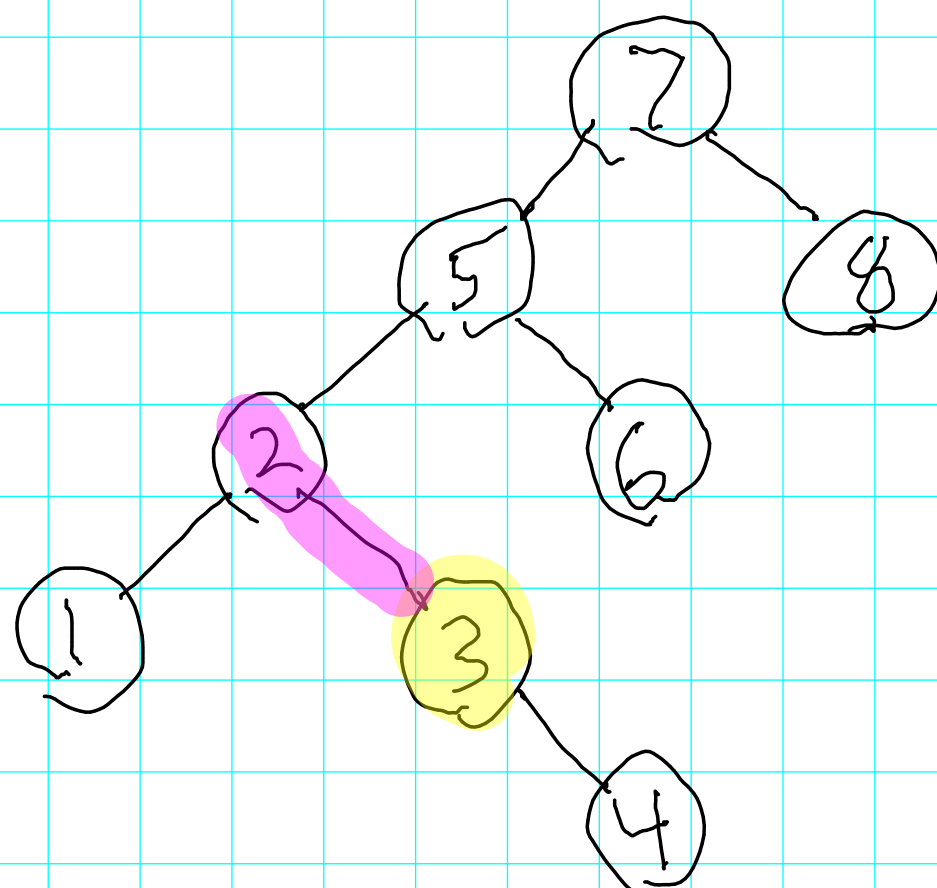
# Move-to-Root: First Idea



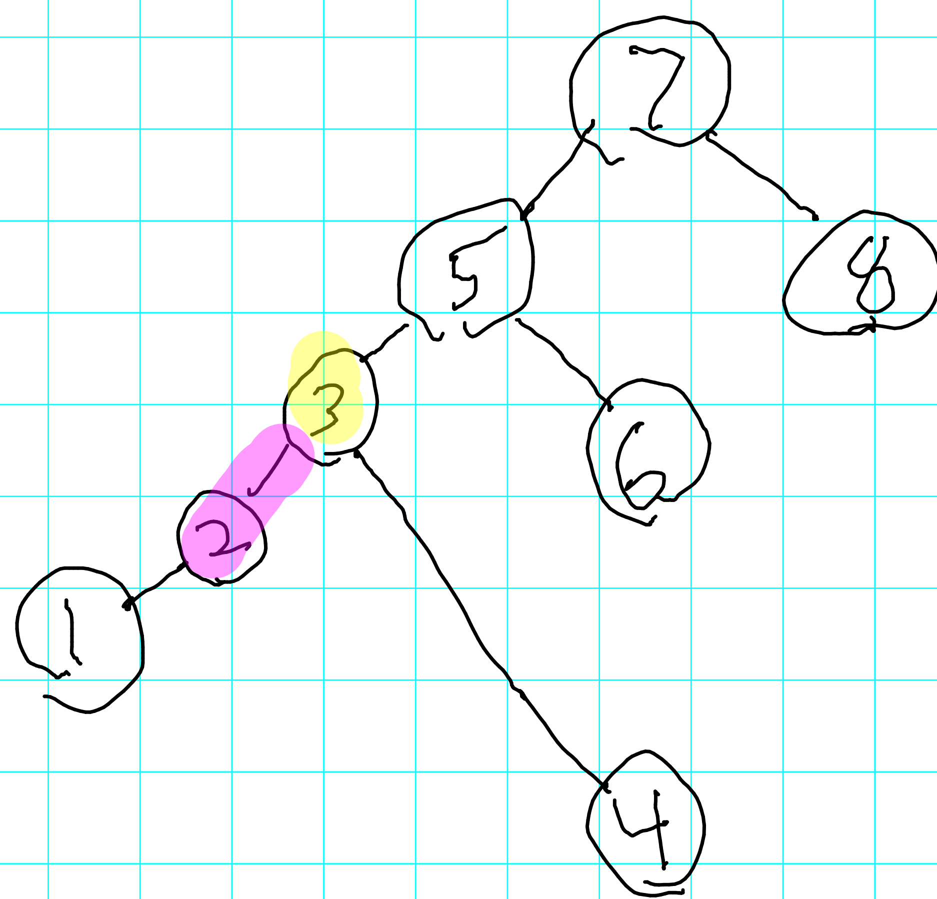
# Move-to-Root: First Idea



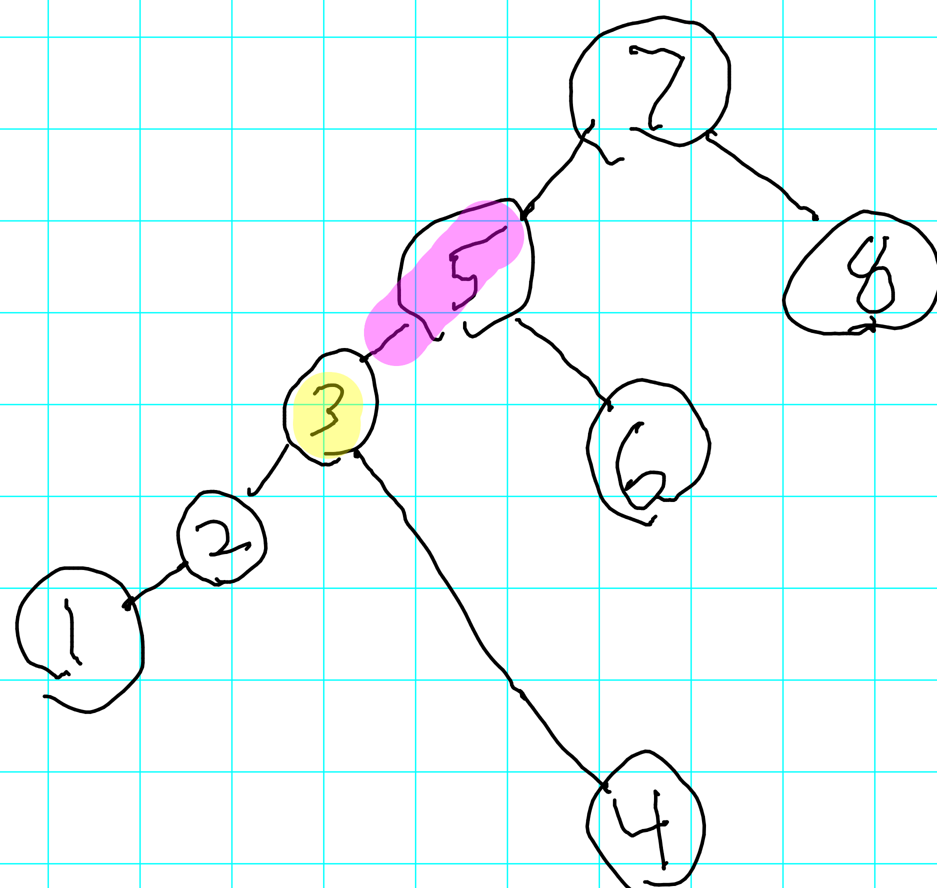
# Move-to-Root: First Idea



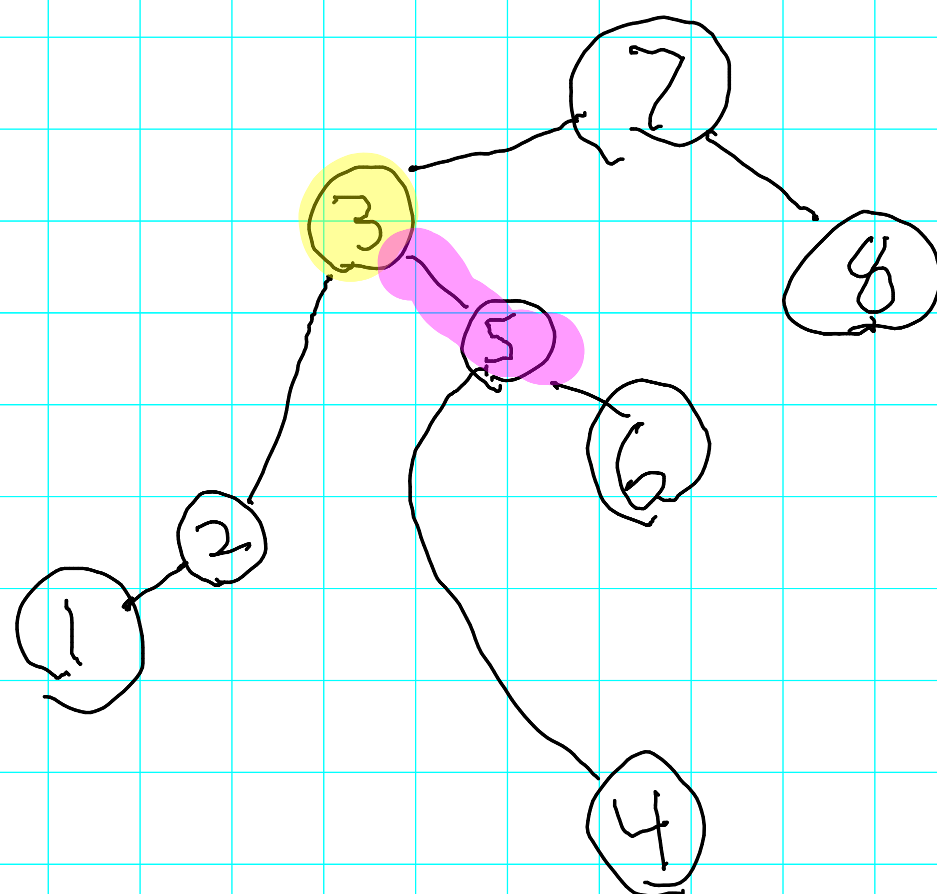
# Move-to-Root: First Idea



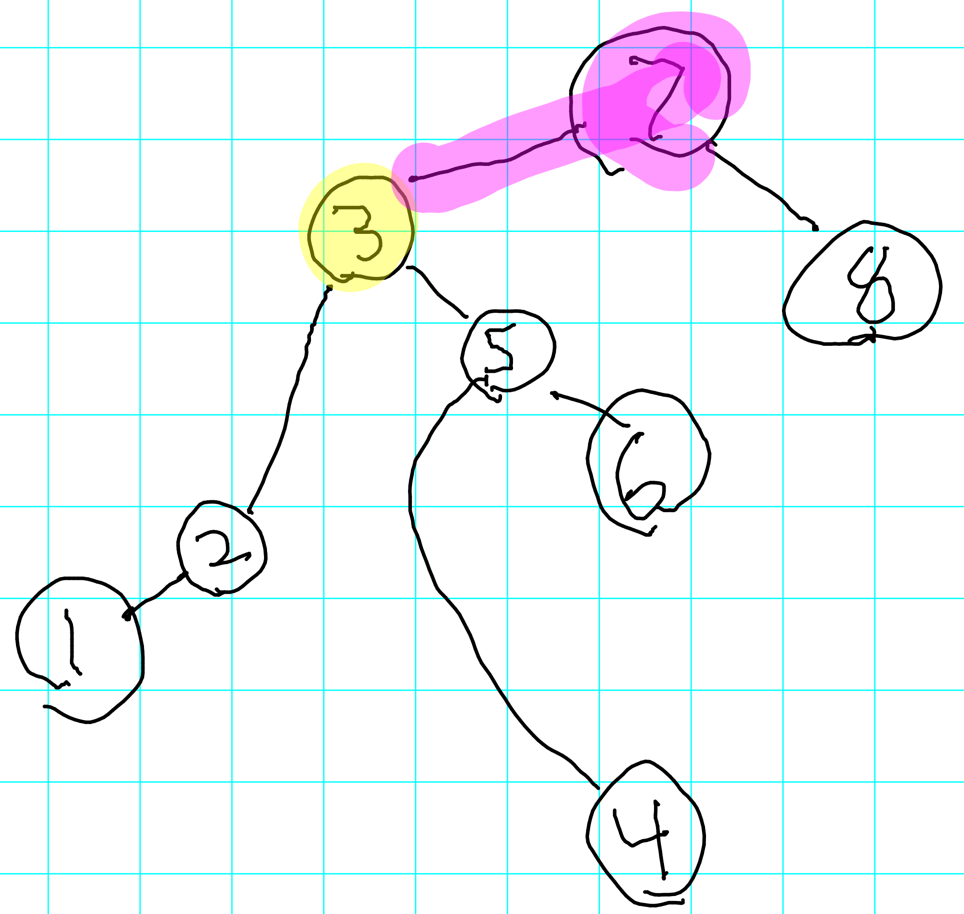
# Move-to-Root: First Idea



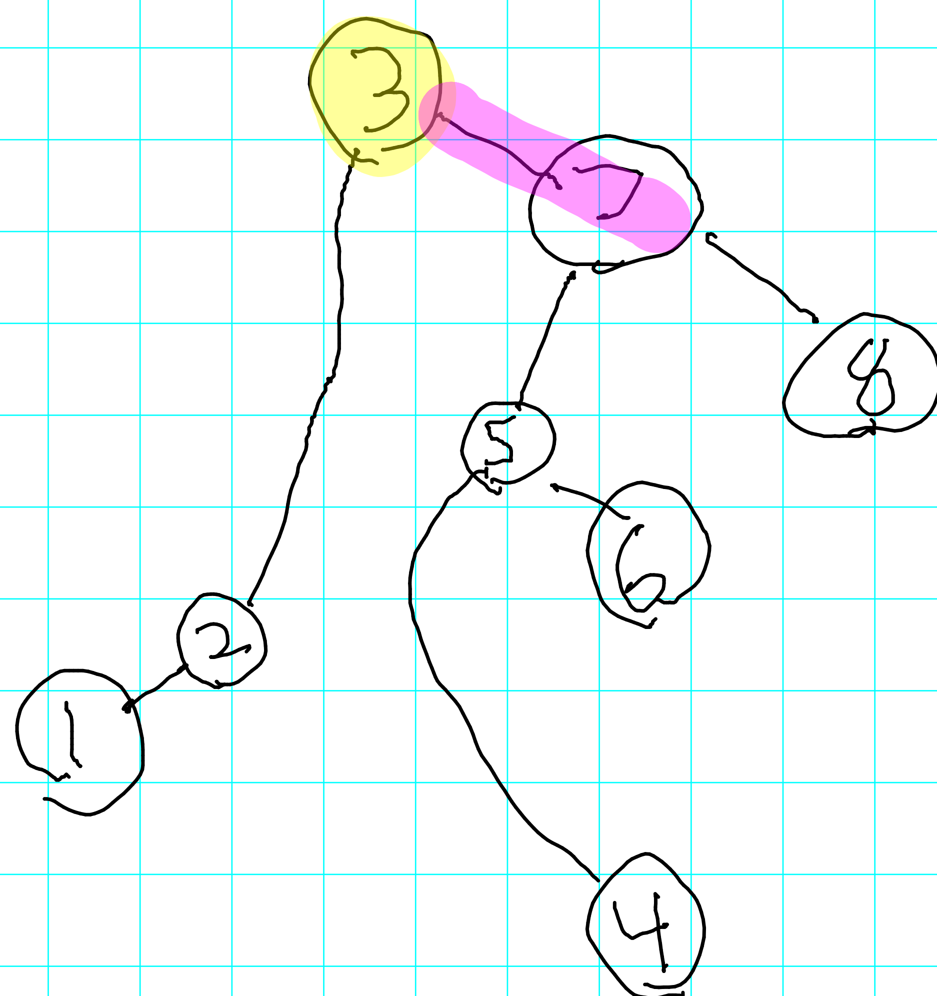
# Move-to-Root: First Idea



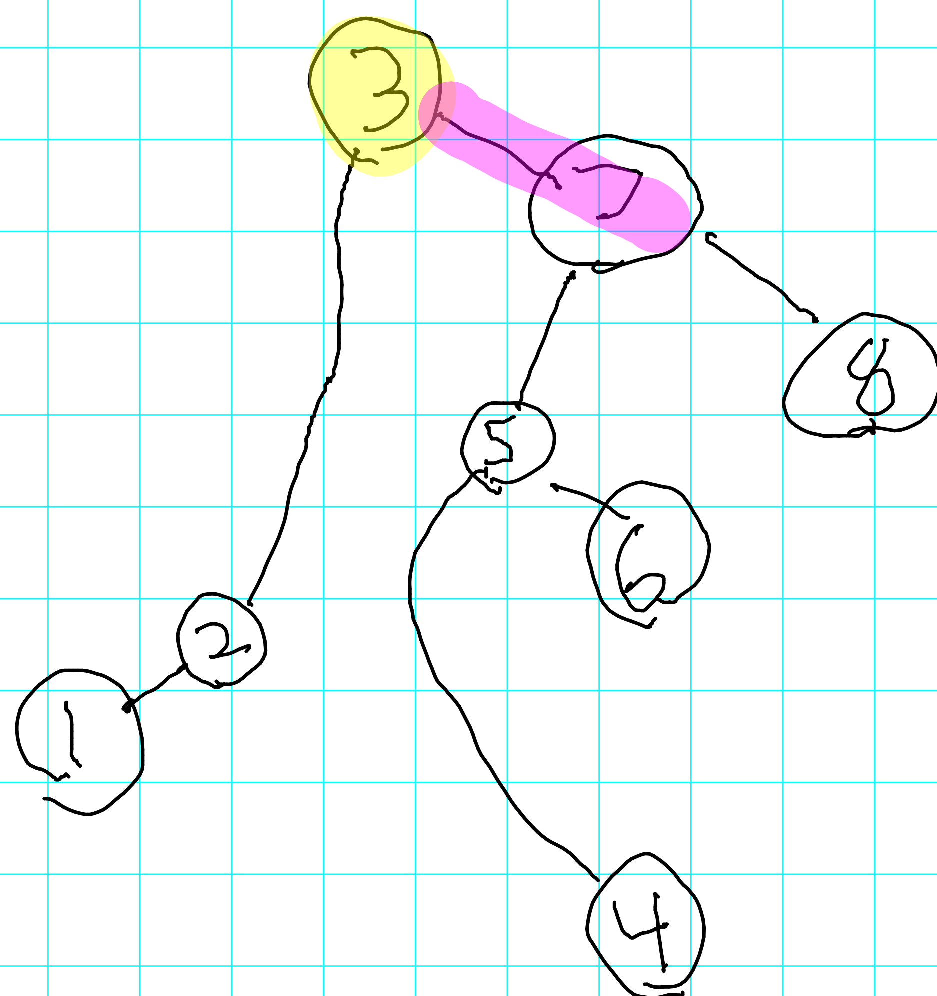
# Move-to-Root: First Idea



# Move-to-Root: First Idea

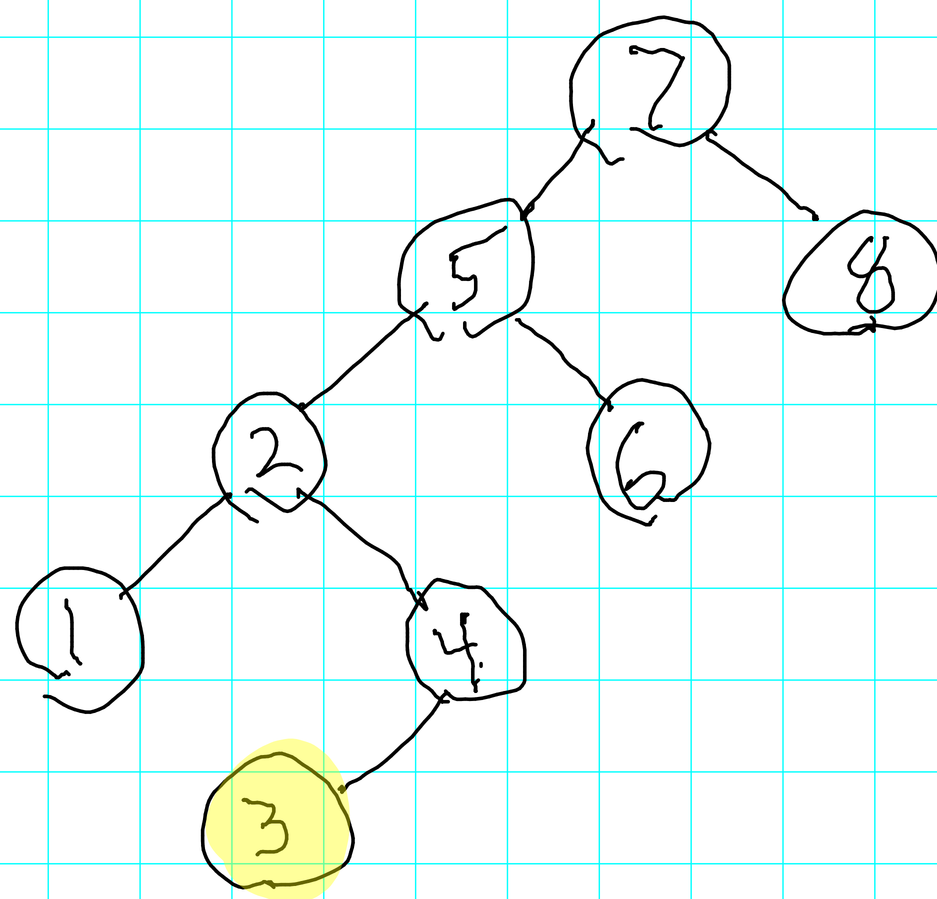


# Move-to-Root: First Idea

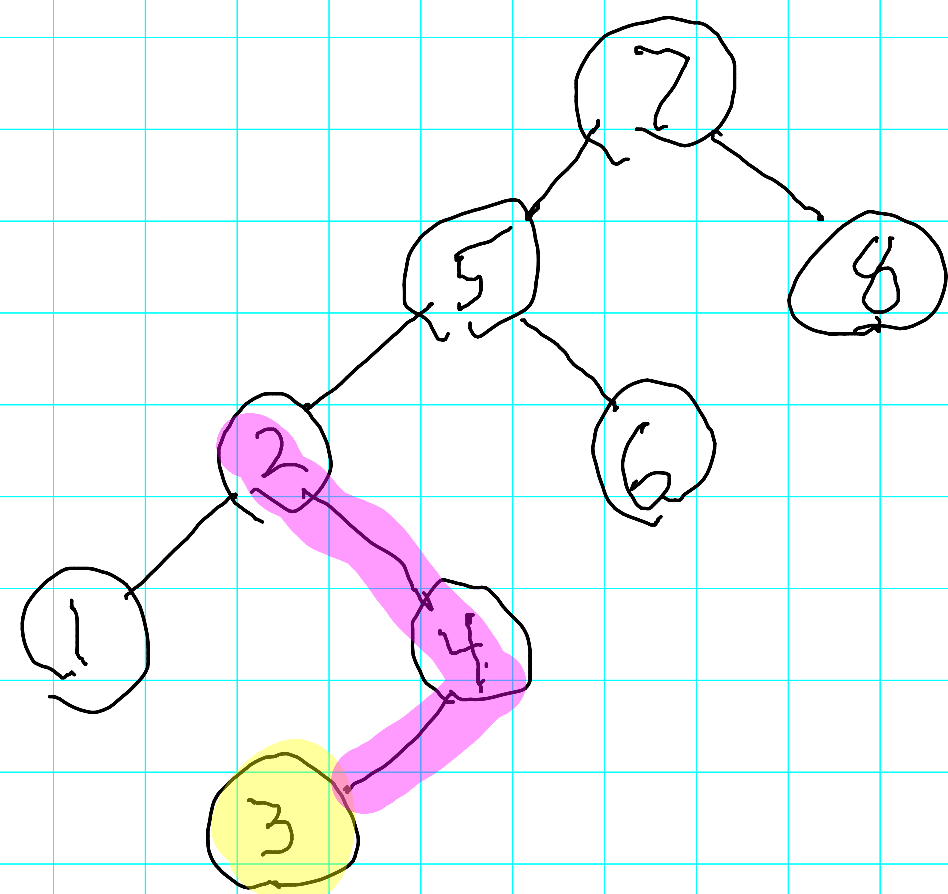



Problem: O(n) runtime

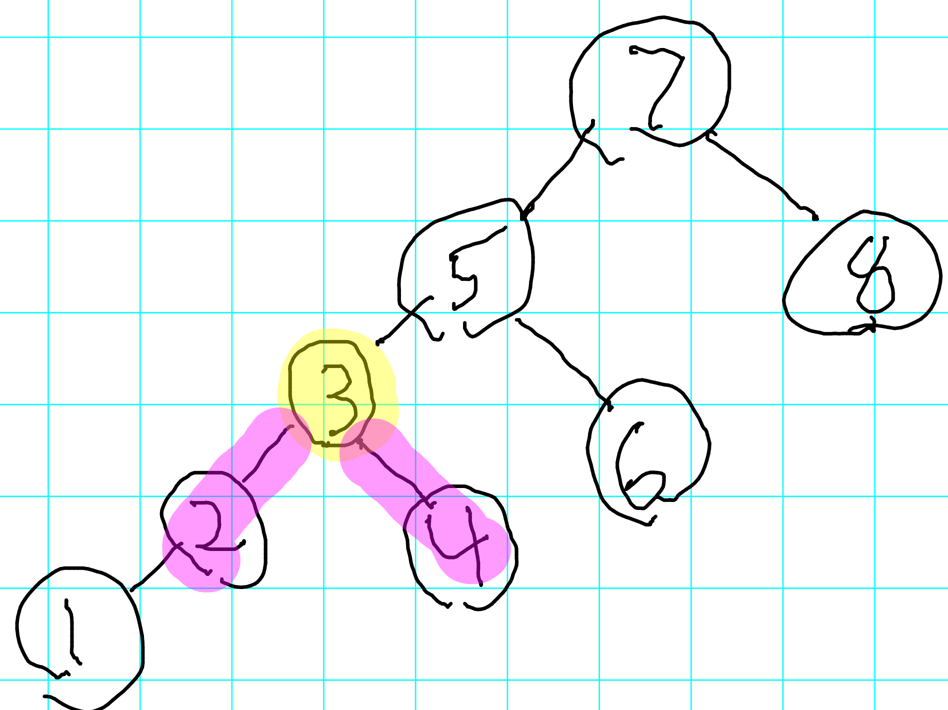
# Move-to-Root: Second Idea



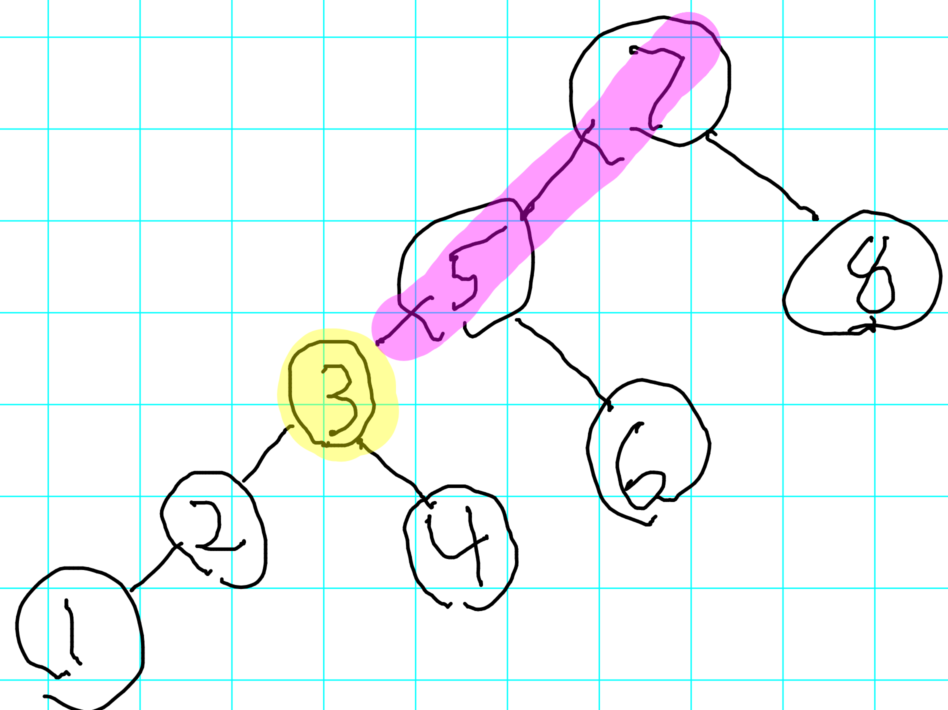
# Move-to-Root: Second Idea



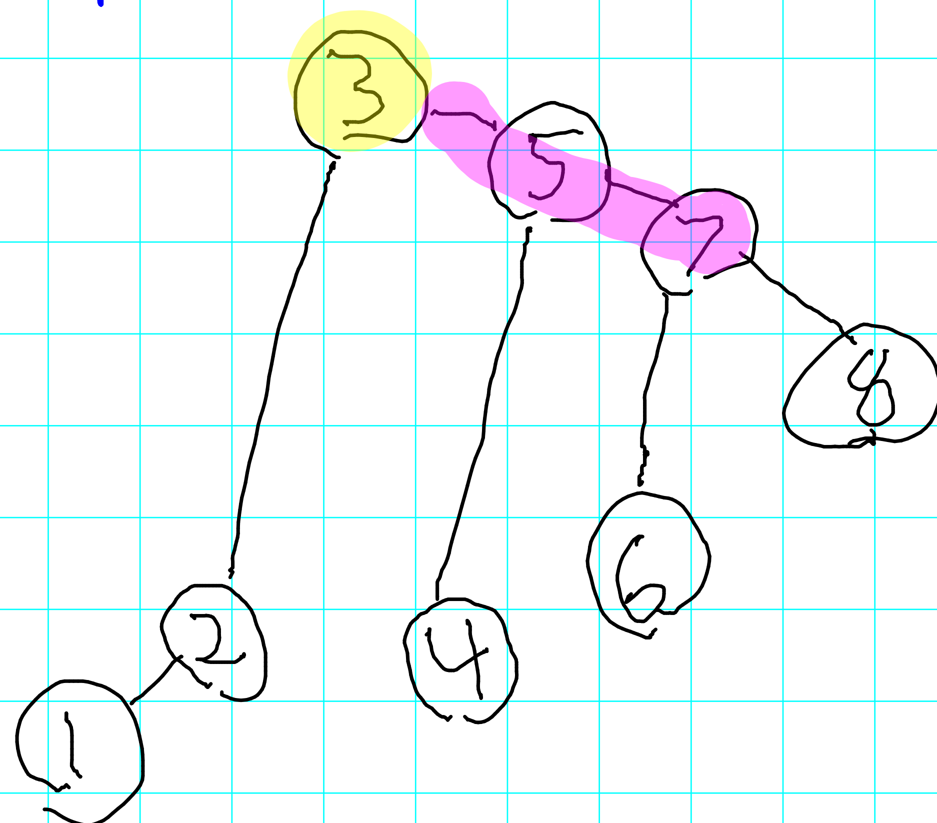
# Move-to-Root: Second Idea



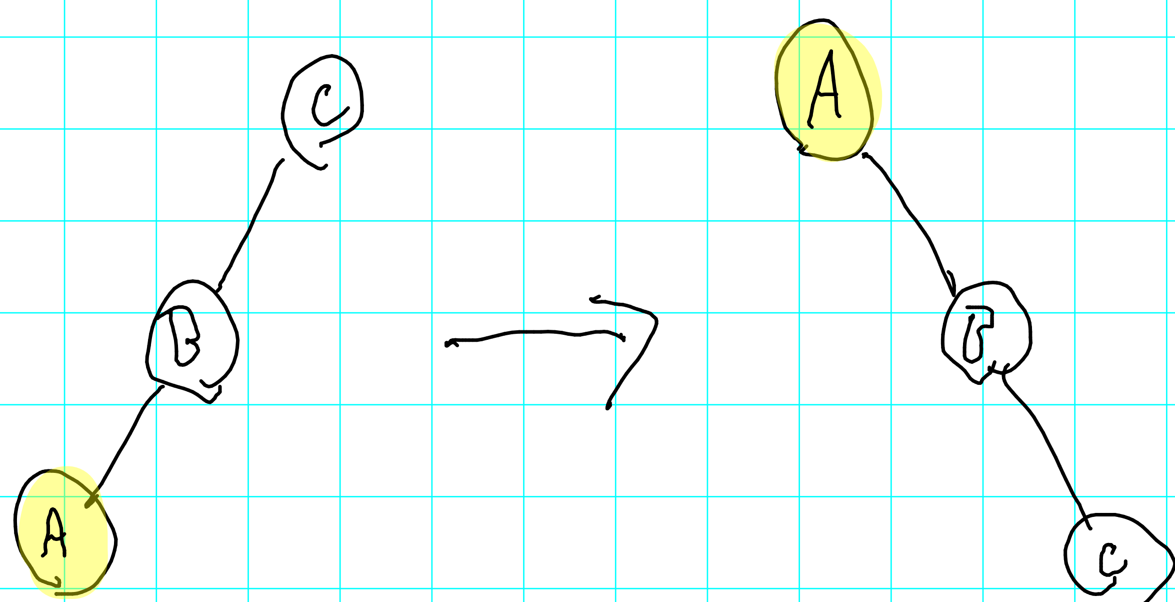
# Move-to-Root: Second Idea

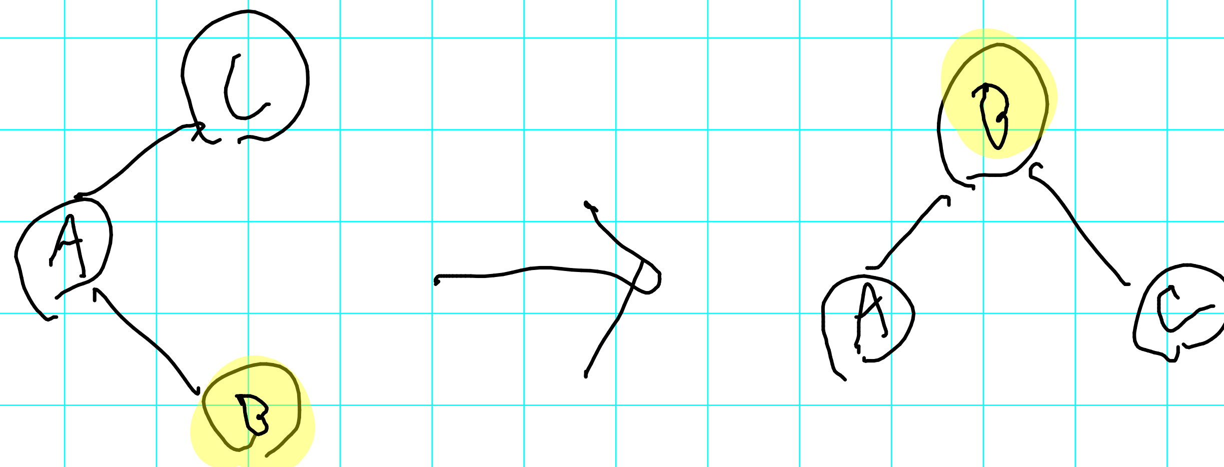


# Move-to-Root: Second Idea



# Splay Tree Rules



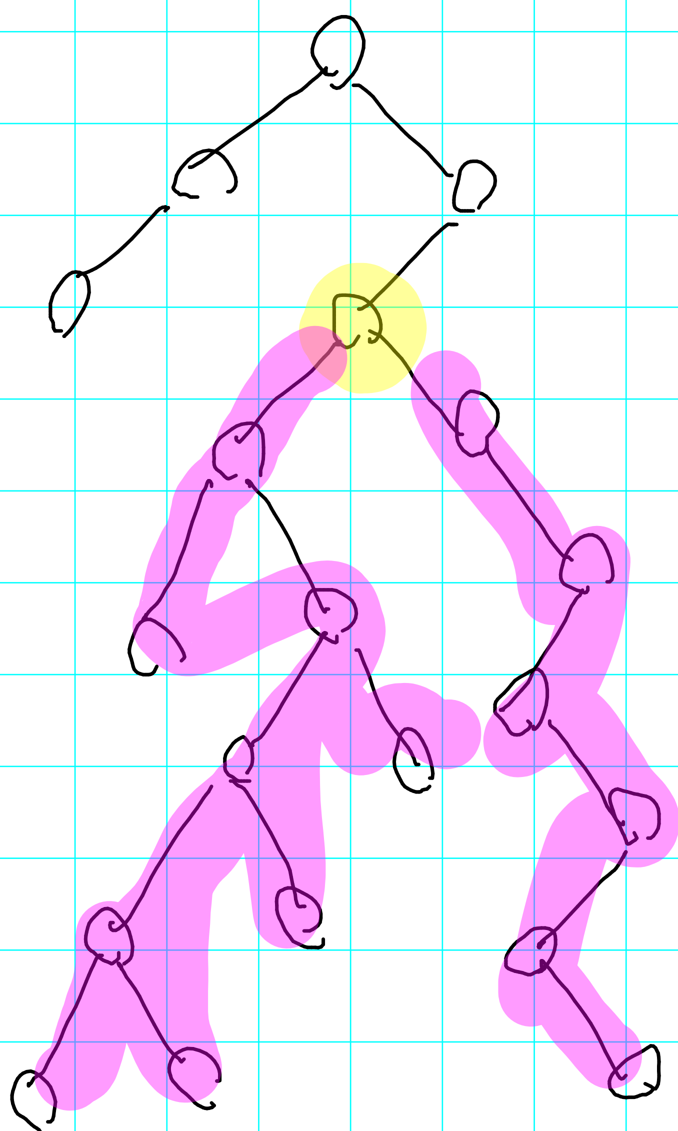



# Splay Trees

- EZ to understand
  how they work

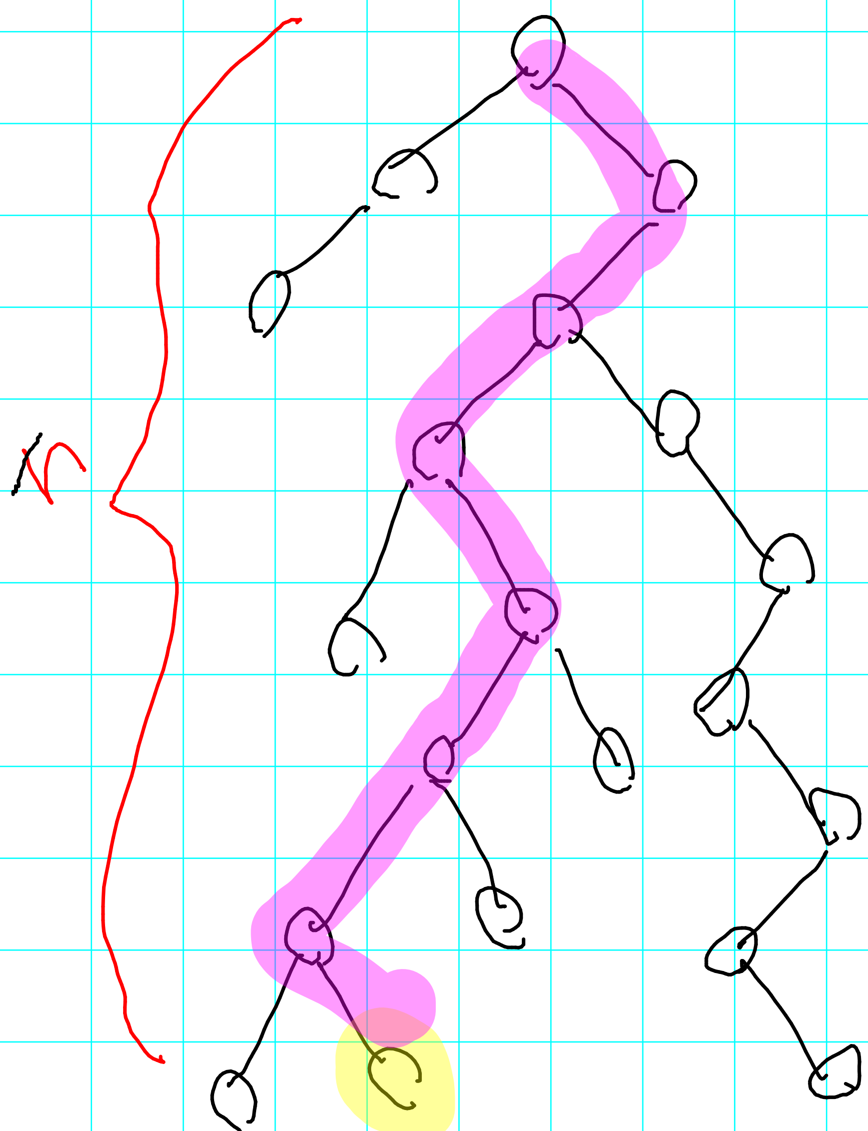
- Hard to understand
  why they work

# Amortized Analysis - Potential



- Potential of a node is log of subtree size

# Amortized Analysis - Potential




- Potential of a node is log of subtree size
- To splay a node at depth $h$ costs $h$
- Potential change is $O(\log n) - h$
- Amortized cost $= O(\log n)$

# The Big Conjecture

Splay Trees Are The BEST BST

(within constant factors)

# The Big Conjecture

Splay Trees Are The BEST BST

(within constant factors)

What is a BST?

# What is a BST? [Wilber 89]

- You know the structure
- Unit cost operations, one pointer
  - Go Left/Right/UP
  - Rotate Left/Right
- To search, bring the pointer to the item being searched.

# What is an optimal BST

Given a search sequence

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

$OPT(X) =$ Fastest any BST can run $X$

# What is an optimal BST?

Given a search sequence

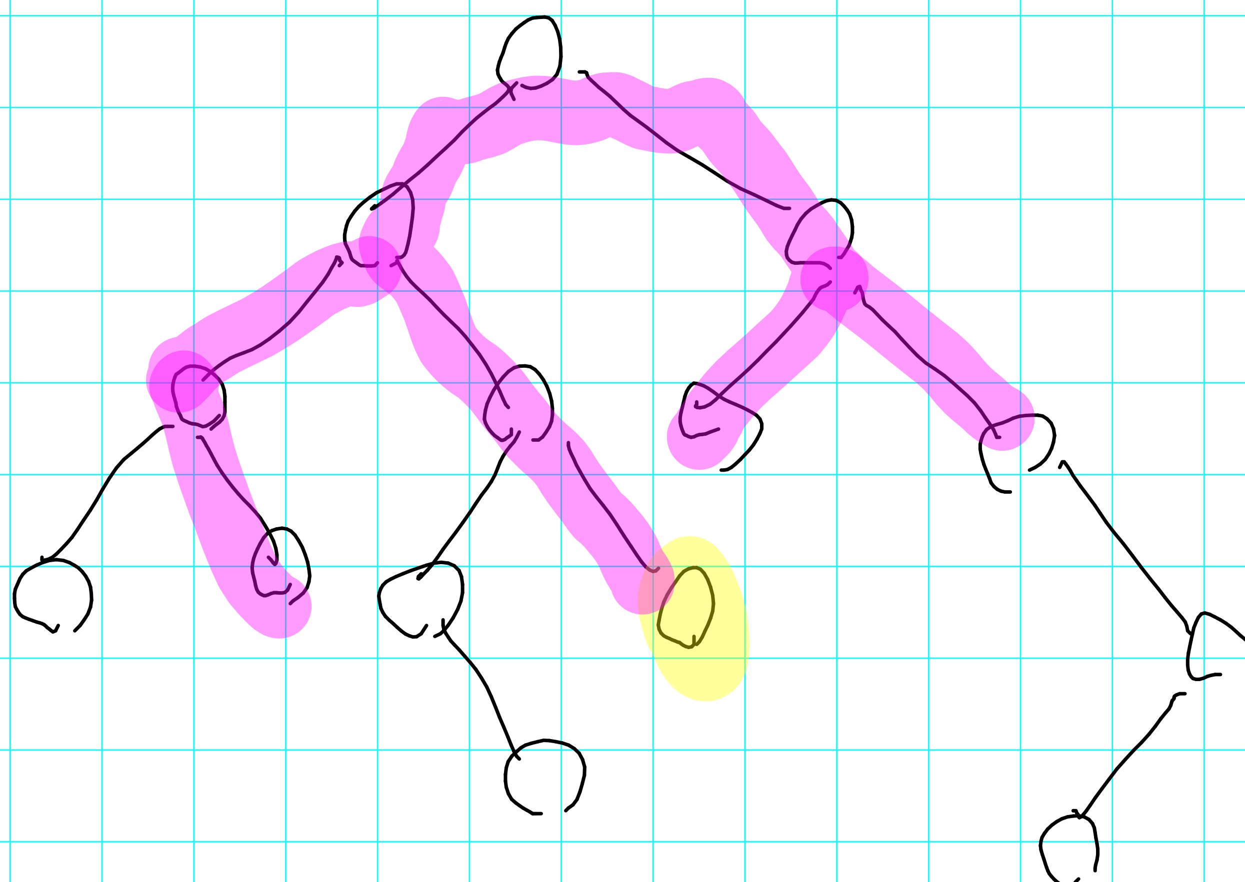
$$X = x_1, x_2 \ldots x_m$$

$OPT(X)$ = Fastest any BST can run $X$

Can be computed in time

$$O\left(5^{m 2\lceil \log_2 n \rceil}\right)$$

# BST Model - Another View



# Dynamic Optimality Conjecture

$$\text{SPLAY}(X) = O(\text{OPT}(X))$$

# Dynamic Optimality Conjecture

[S+T]

$$\mathrm{SPLAY}(X) = O(\mathrm{OPT}(X))$$

Online Offline

# BST Model

- Crucial, With a more powerfull model, the comparisom with the best offline algorithm is not so interesting, if space is unlimited.

# Open Problem

- Come up with any result on dynamic optimality in the ram model.

# Open Problem

- Come up with any result on dynamic optimality in the ram model.
- E.G. A structure

$$\text{YOURDS}(X) = o\left(\sqrt{\frac{\log n}{\log\log n}}\ \text{RAMOPT}(X)\right)$$

So we can't prove it. What next....

- Find some specific things that trees can do and prove/conj that splay trees can do it too.

# Splay trees $\leq$ Prehistory

- As good as "optimal" trees though we don't know the probabilities
- As good as static finger trees though we don't know the finger

[S+T]

# Splay trees & Prehistory

- As good as "optimal" trees though we don't know the probabilities
- As good as static finger trees though we don't know the finger

Amazing

# Side note

- Having the same runtime as "optimal" trees is not so hard even if you don't know the probabilities
- Just rebuild every n time with the observed probabilities

# Splay Trees — Working Set

1 2 1 3 4 7 3 1 2

# Splay Trees — Working Set

1 2 1 3 4 7 3 1 2

# Splay Trees — Working Set

1 2 1 3 4 7 3 1 2

$O(\log 4)$

- This implies everything on previous page

# Splay Trees — Working Set

[S+T]

1 2 1 3 4 7 3 1 2

O(log 4)

- This implies everything on previous page

All these just play with weights

- Not amazing

# Splay Trees Can Do It!
(we think)

- Dynamic Finger [Cole, Cole Mishra Schmidt Siegel 2000] 85 pages!

  Bitter hard proof

- Sequential Access [Tarjan 85]

  Combinatorica

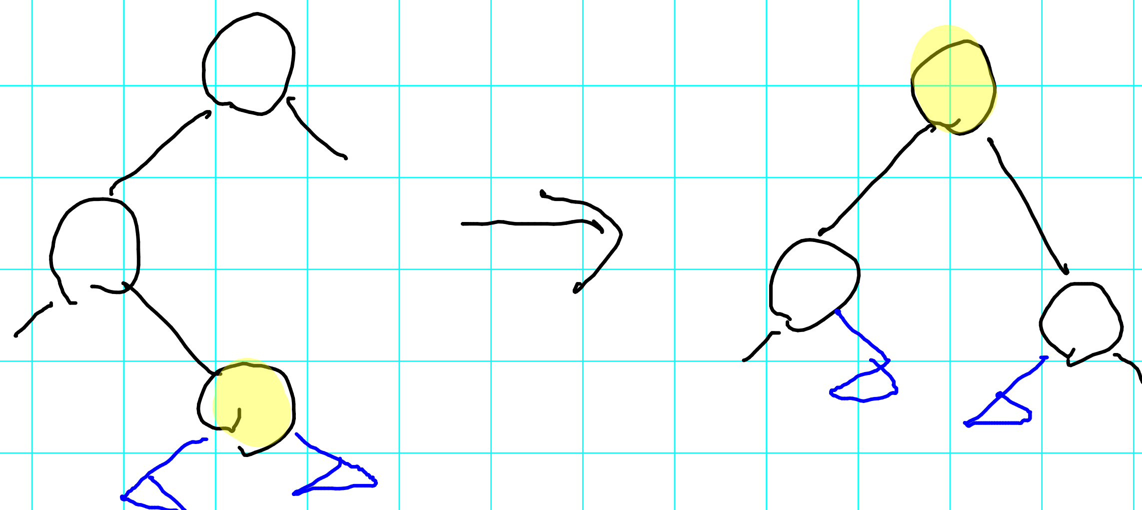
- Trees can be deques

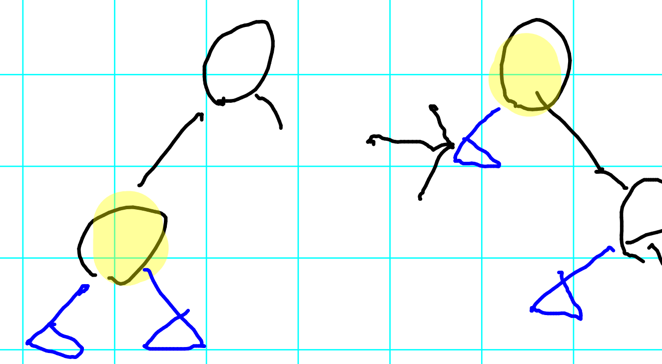
  Still no proof

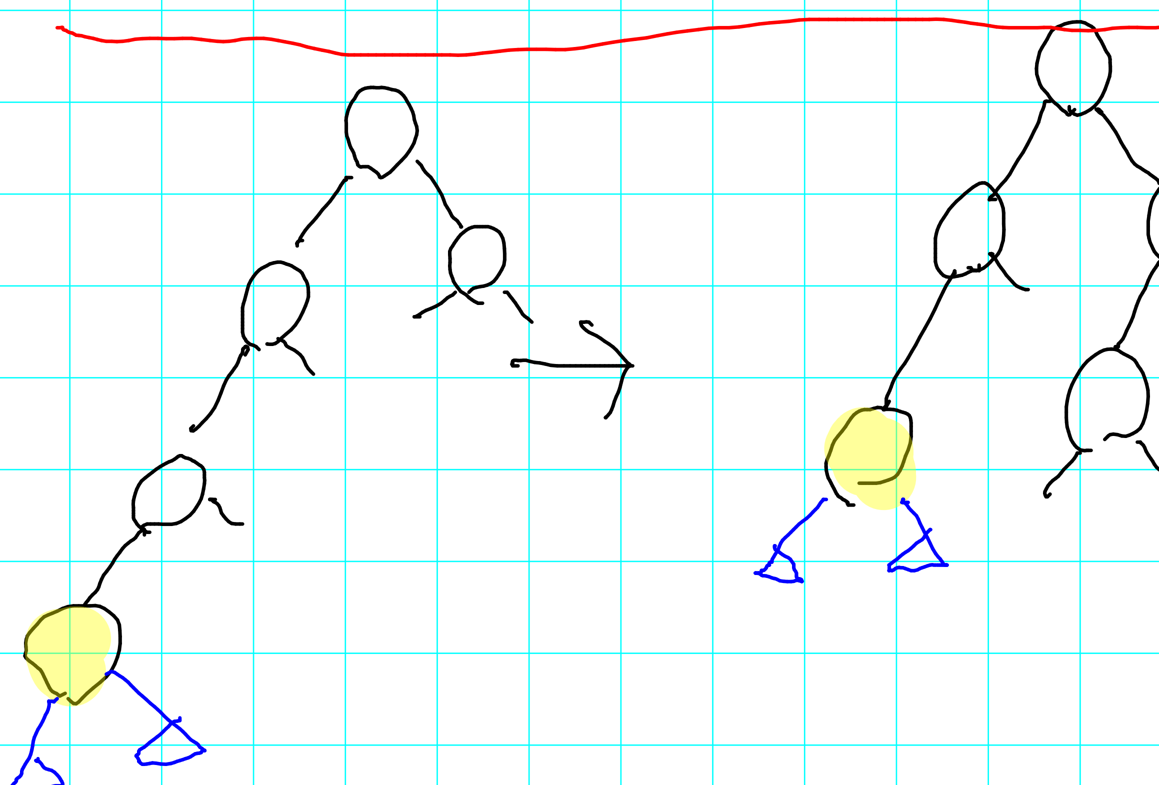
  close ... $O(\alpha^*(n))$ [pettie 08]

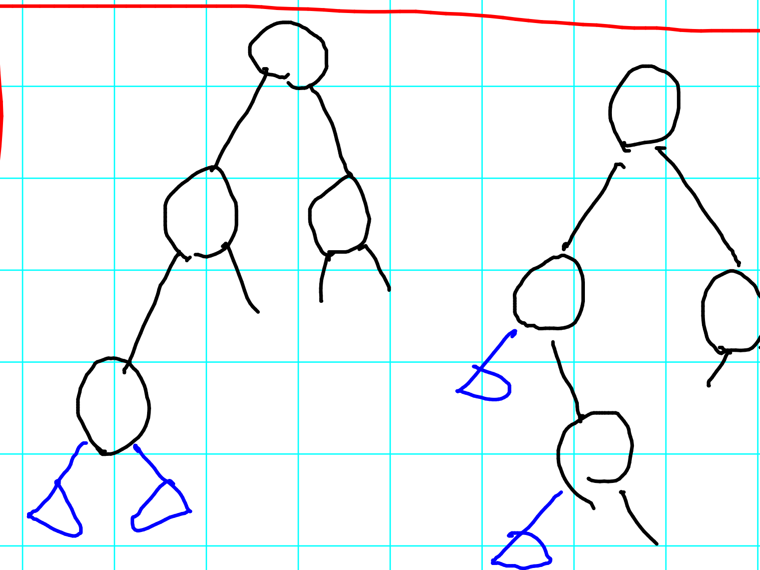
Are the splay rules special?

# Are the splay rules special?



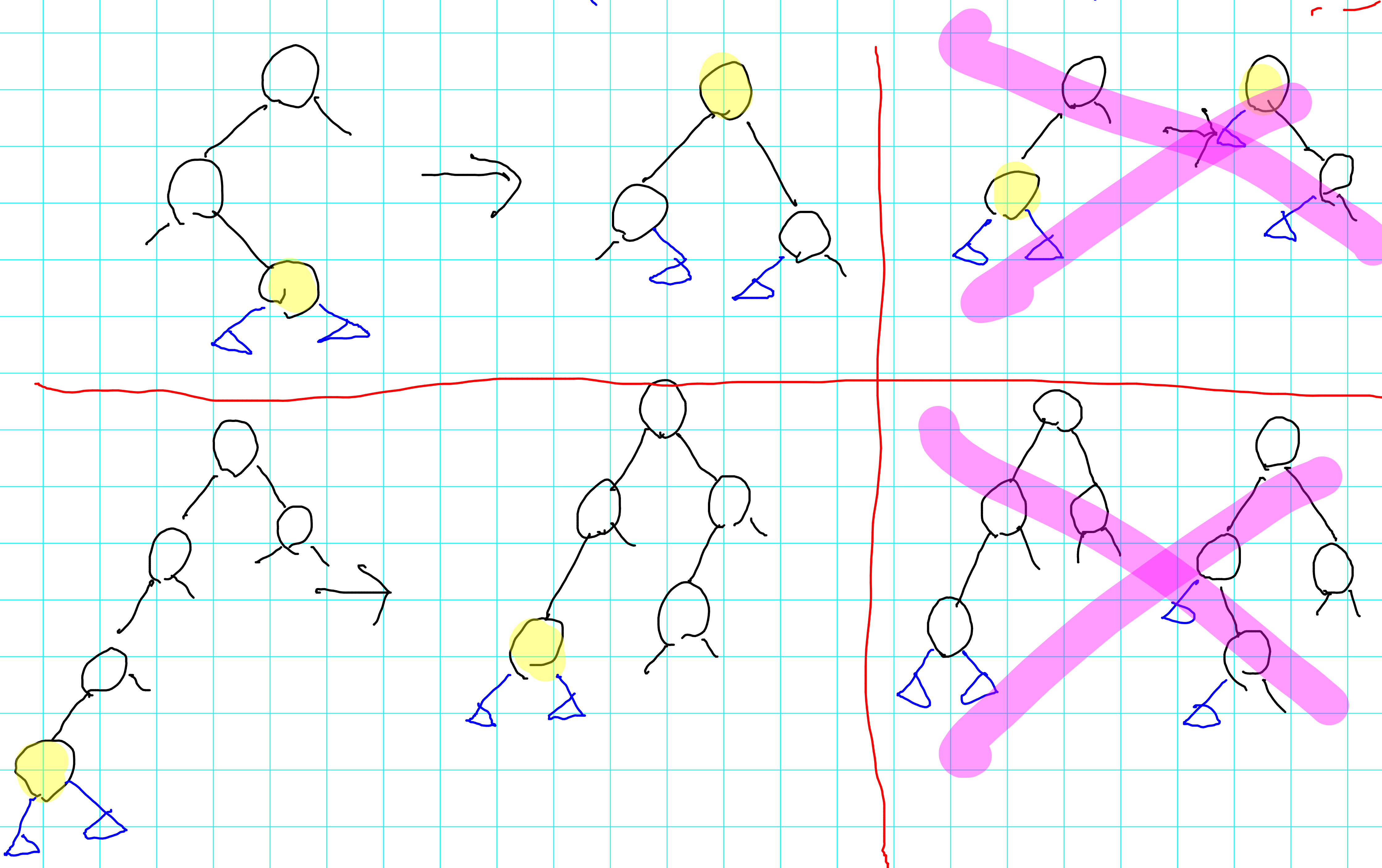




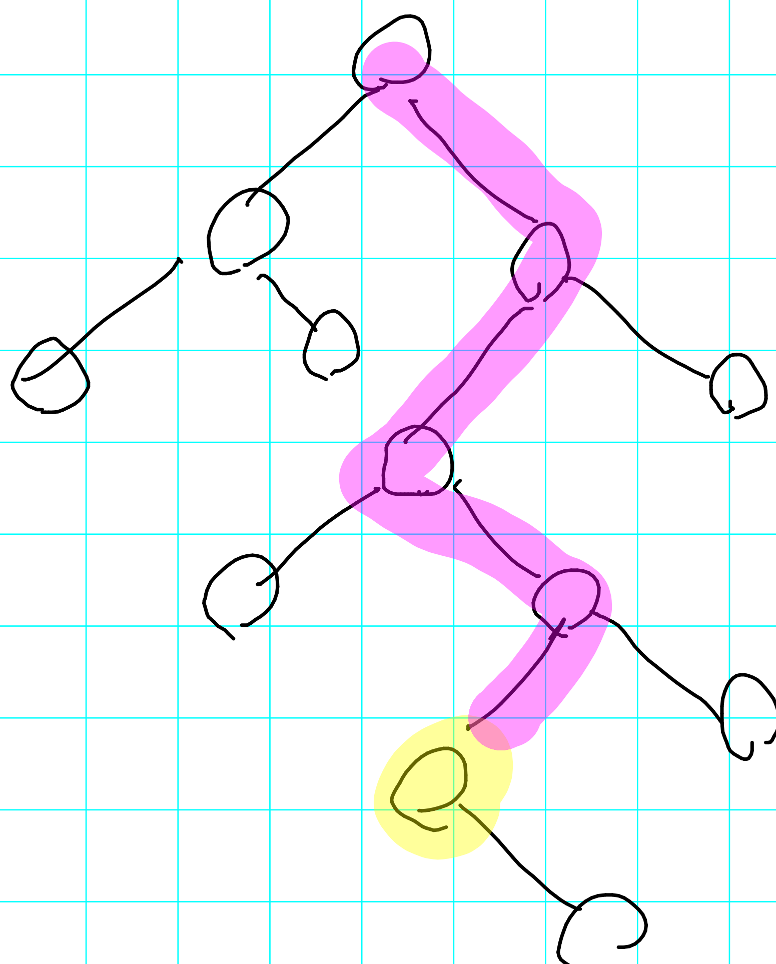


[Subramanian 95]

Are the splay rules special? :O



# So What Is Special?



Depth of node after splay

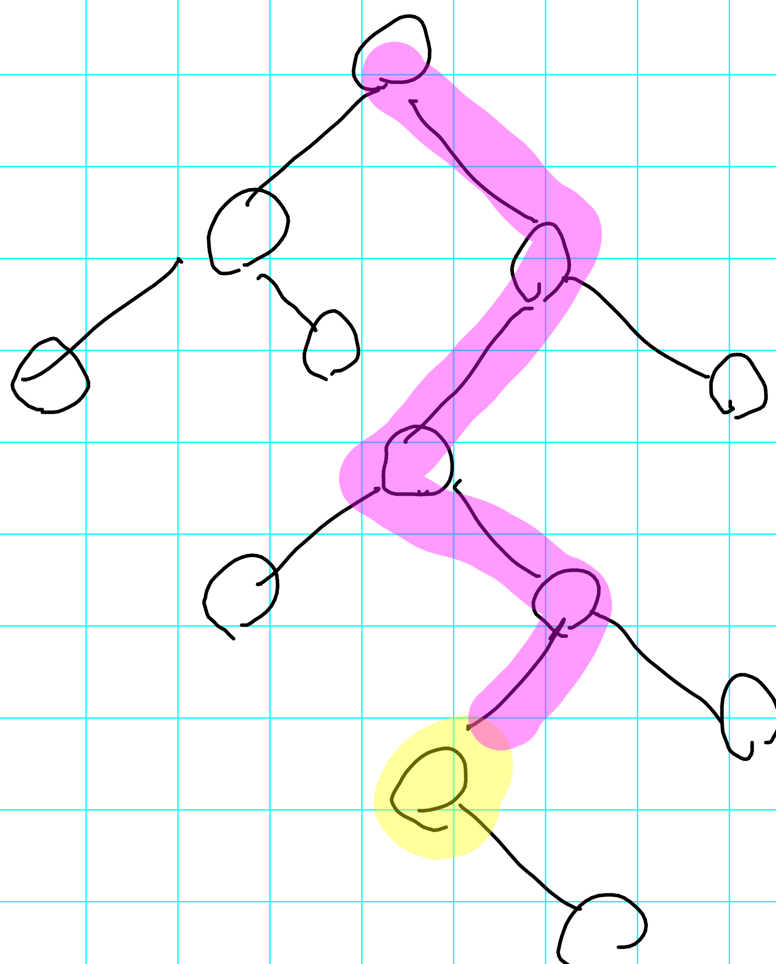
= Before splay

$- c\,|\text{Intersection with splay path}|$

$c < 1$

$+ O(1)$

# So What Is Special?



Depth of node after splay

= Before splay

− C |Intersection with splay path|

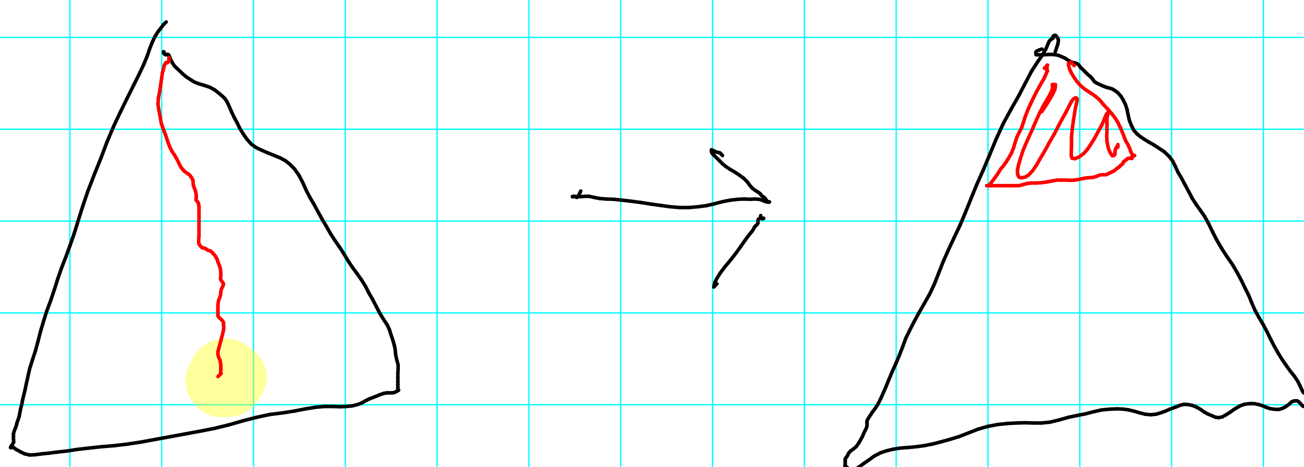
<1

+ O(1)

Would be nice to see a proof that used this directly

# Other Conjectured Variants

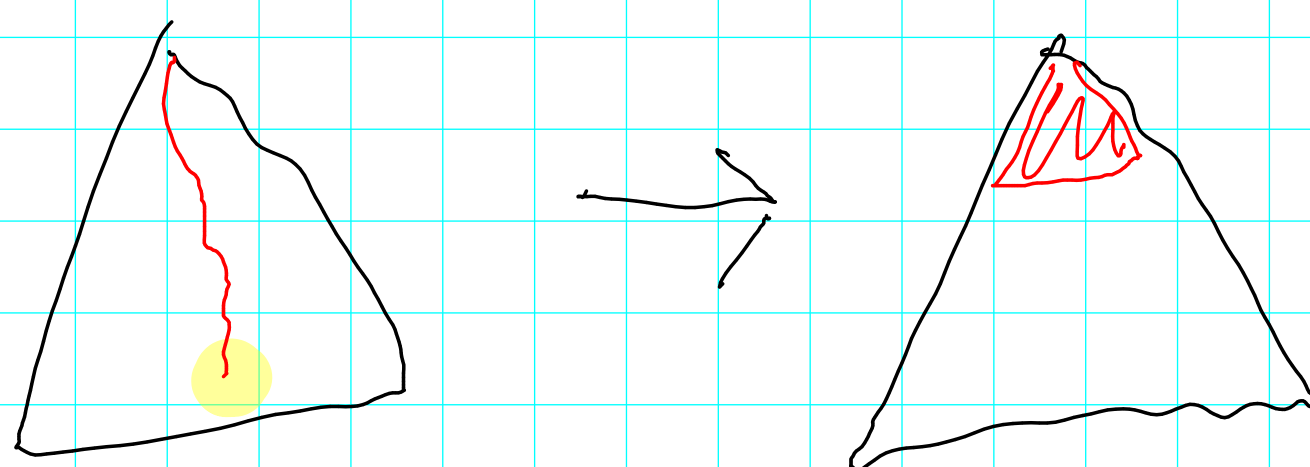
- Turn search path into complete balanced tree



[comm from Fredman]

# Other Conjectured Variants

- Turn search path into complete balanced tree

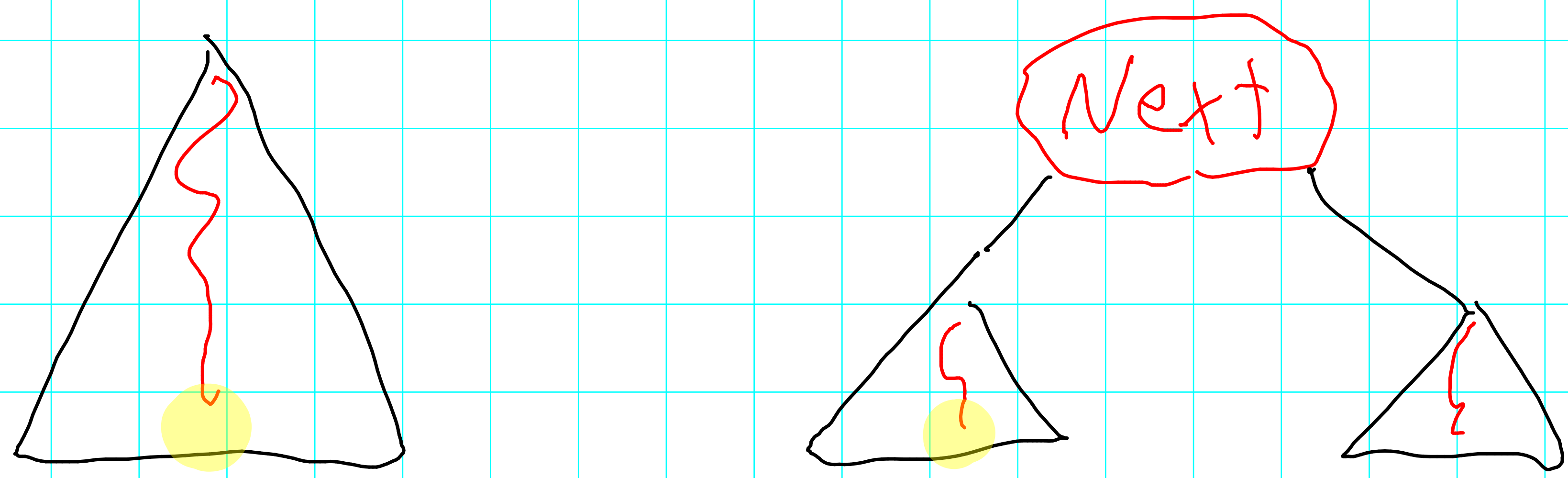


Conj: Not $O(\log n)$ amortized

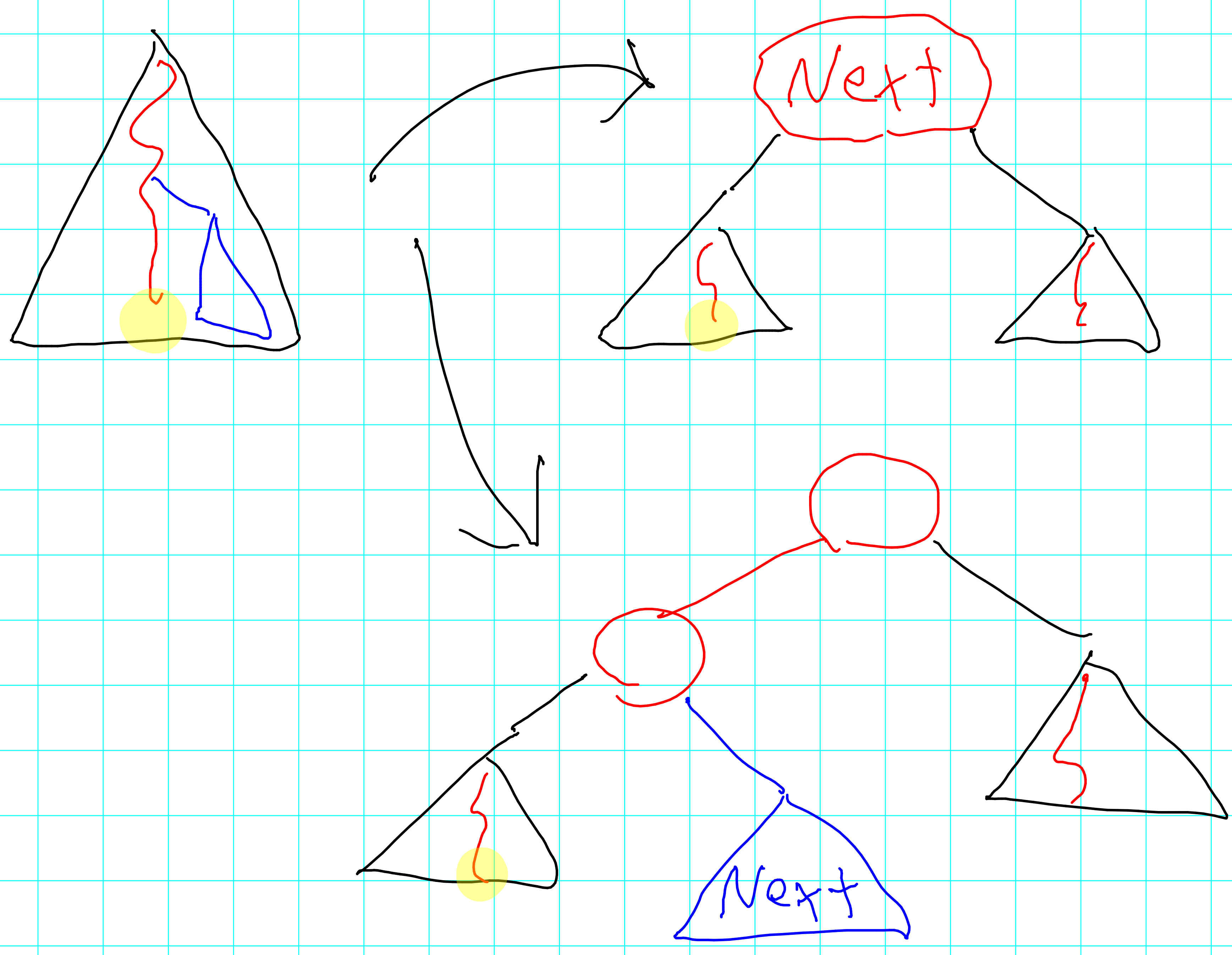
# Other Conjectured Variants

- Idea: Splay trees are conjectured to be $O(OPT(X))$ and online.
- Is there a poly-time $O(OPT(X))$ BST?
- If you knew the future who would you put closest to the root [comm Munro]

IAN



IAN



# IAN

- Conj: $IAN(x) = O(OPT(x))$

# IAN

- Conj: $IAN(X) = O(OPT(X))$

- Conj: IAN has $O(\log n)$ amortized time per search

SAD

# Esoterica

[Io3]

- If the key values were to be randomly permuted (maintaining equality) then WORKINGSET $(X) = O(\mathrm{OPT}(X))$

# Esoterica

- If the key values were to be randomly permuted (maintaining equality) then $\text{WORKINGSET}(X) = O(\text{OPT}(X))$
- If rotations are free, and you have LOTS of time to think there is an online dyn opt struct [Blum, Chawla, Kalai 02]

# So what next?

- The proofs are getting hard so lets try to invent some things that are simpler
-

# So what next?

- The proofs are getting hard so lets try to invent some things that are simpler
- Unified Trees: A combination

[IOI; Badoiu, Cole, Demaine, I 07]

# So what next?

- The proofs are getting hard so lets try to invent some things that are simpler
- Unified Trees: A combination of spatial and temporal locality

# So what next?

- What about a competitive ratio result?

$$MYDS(X) = c \cdot OPT(X)$$

$c = O(\log n)$ trivial

$c = O(1)$ dynopt

# So what next?

- What about a competitive ratio result?

$$\mathrm{MYDS}(X) = c \cdot \mathrm{OPT}(X)$$

$c = O(\log n)$ trivial

$c = O(1)$ dynopt

[Demaine, Harmon, I, Patrascu 07]

- $\mathrm{TANGO}(X) = O(\log\log n) \cdot \mathrm{OPT}(X)$

# Competitive Trees

- Competitive ratio stuff needs a decent lower bound

$$OPT(X) \geq \text{somefunct}(X)$$

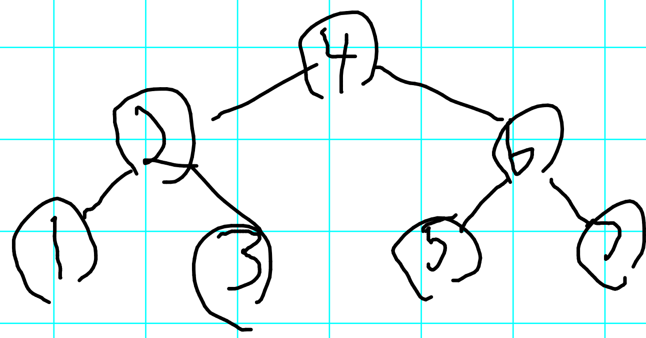
- We are luckly, more than luckly, there are two!

[Wilber 89]

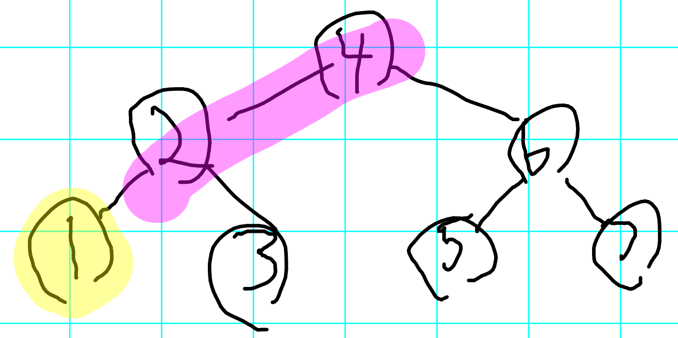
# Defaur

- Thinking about trees, rotations, etc, hurts my head

- There must be a Better way

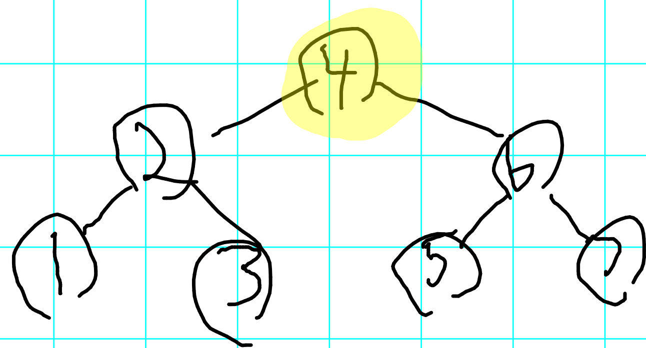


$X = 1, 4, 5, 7, 6, 2, 3$

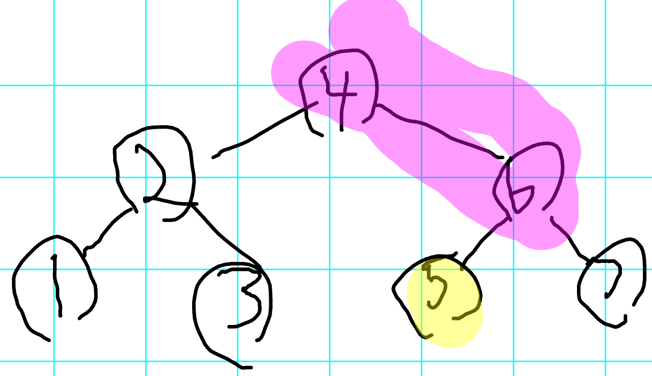


$X = 1, 4, 5, 7, 6, 2, 3$

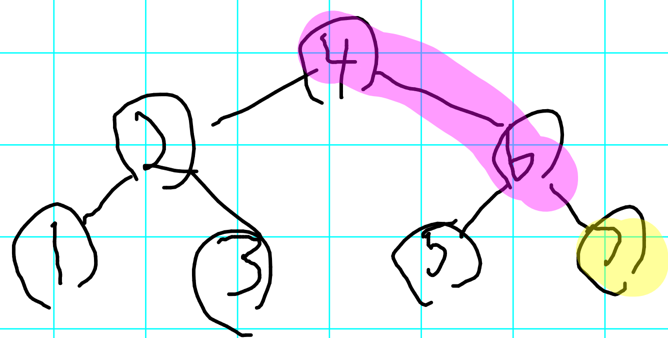
X X X



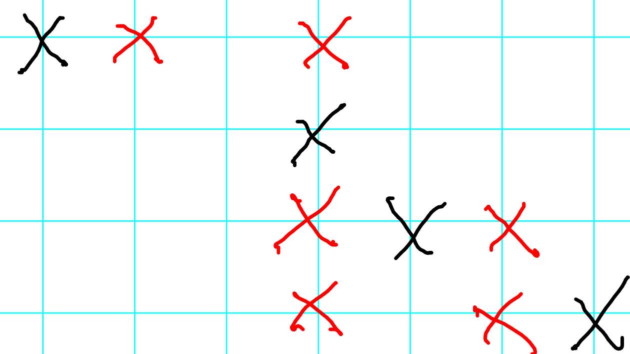

X = 1, 4, 5, 7, 6, 2, 3

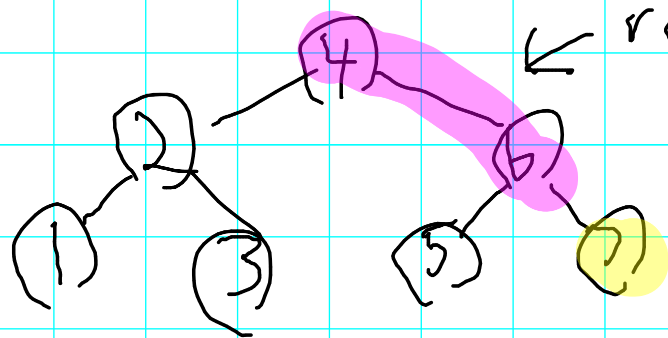


X = 1, 4, 5, 7, 6, 2, 3



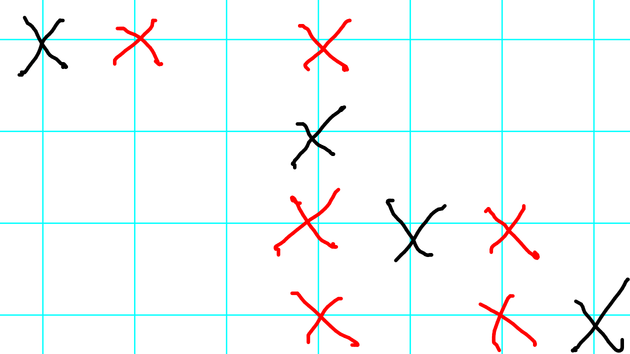

$X = 1, 4, 5, 7, 6, 2, 3$

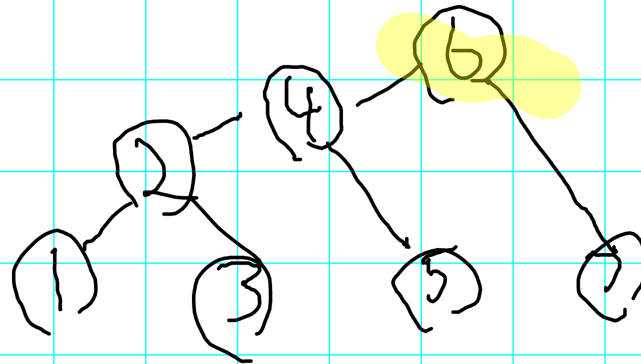




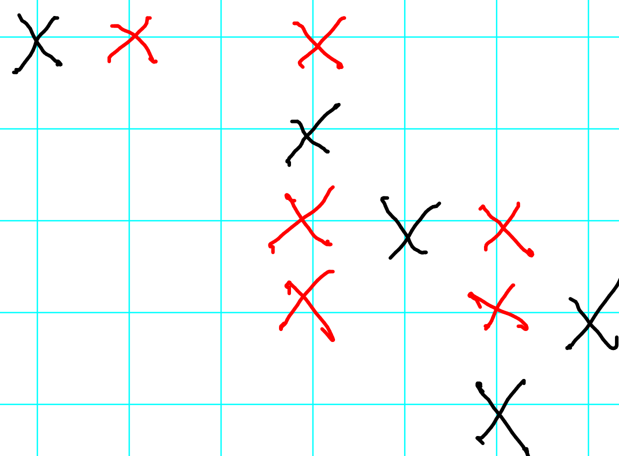
rotate

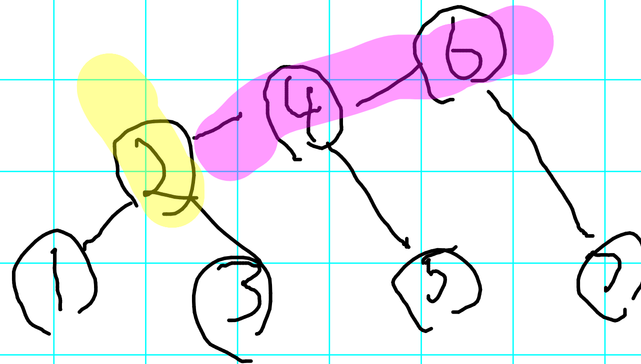
$X = 1, 4, 5, 7, 6, 2, 3$



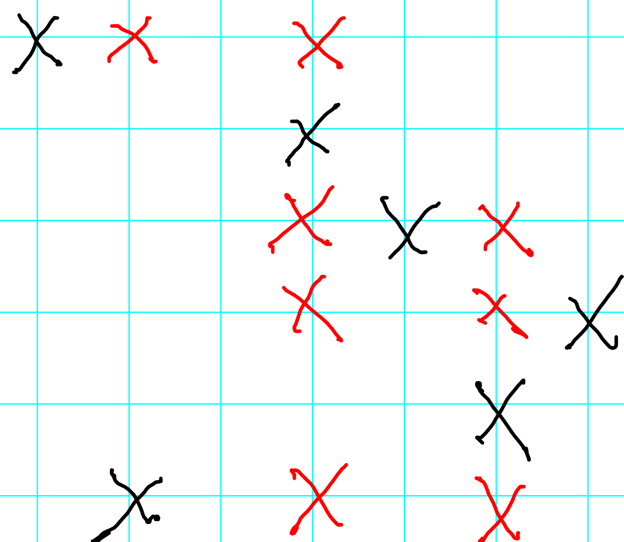


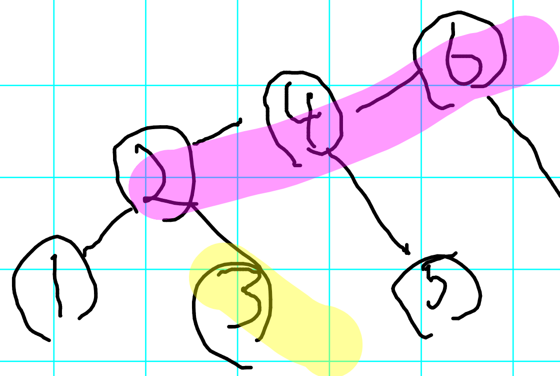

X = 1, 4, 5, 7, 6, 2, 3



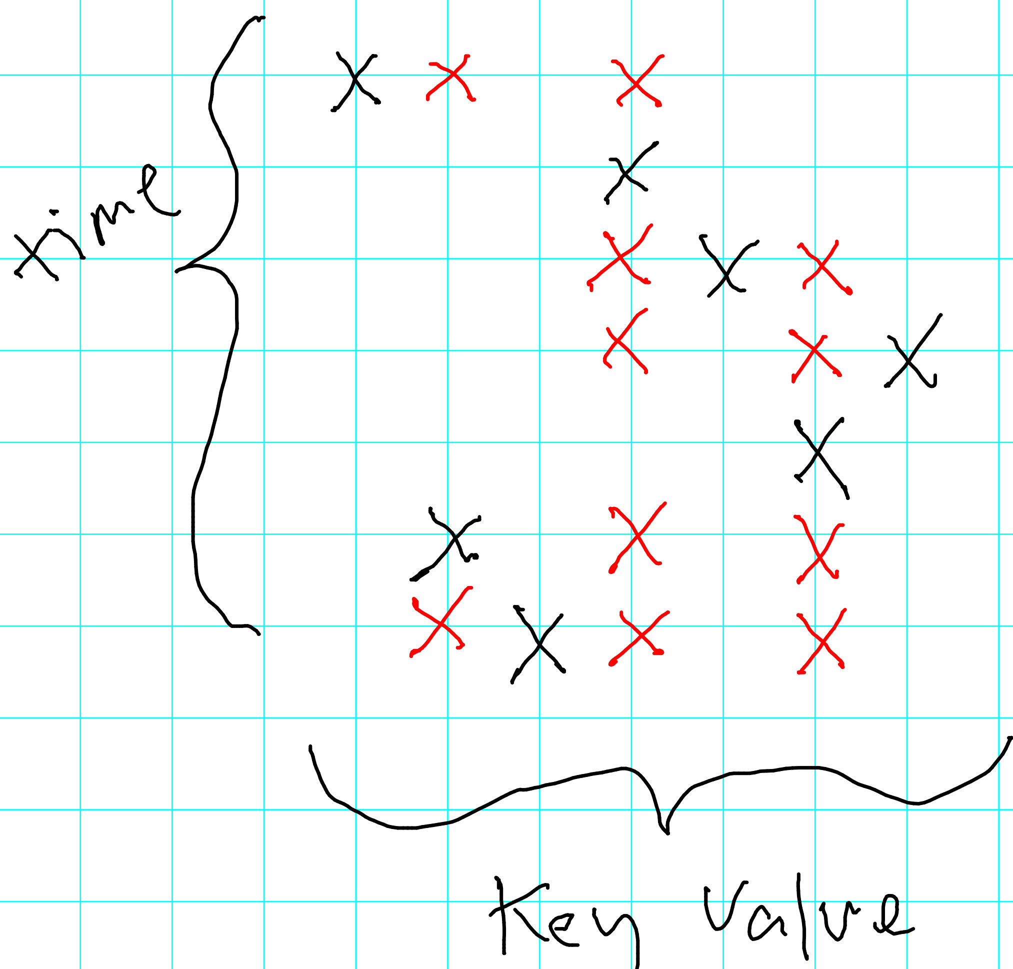



$X = 1, 4, 5, 7, 6, 2, 3$






X = 1, 4, 5, 7, 6, 2, 3




Black X

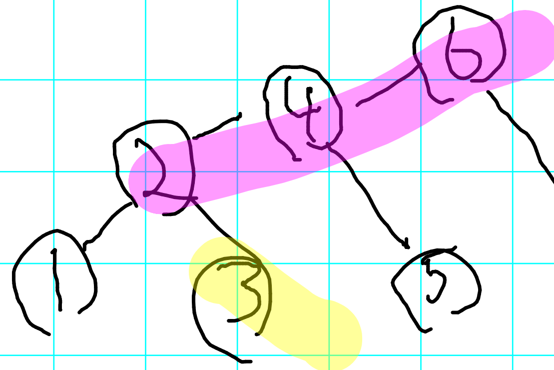
= Item Searched

Red X

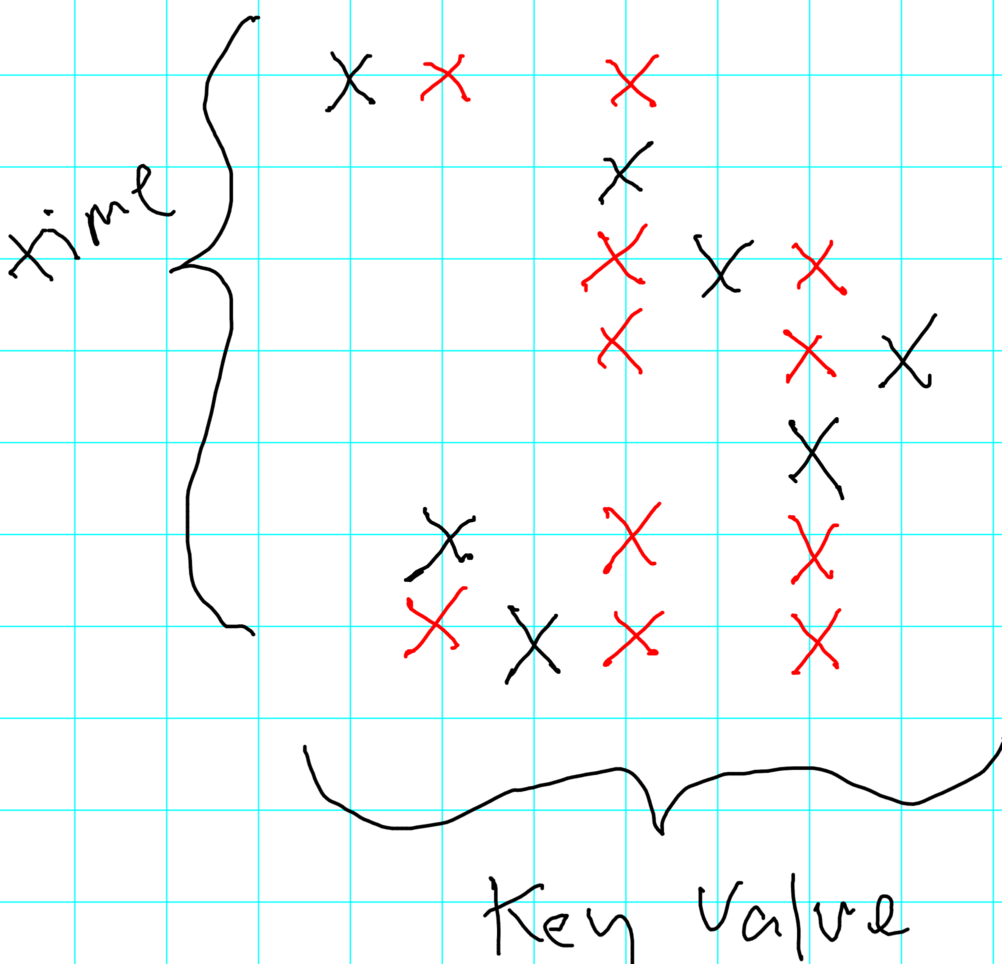
= Item touched

Total # of X's

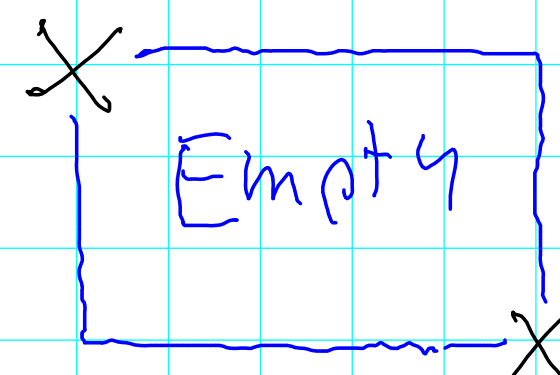
= Total cost of all searches

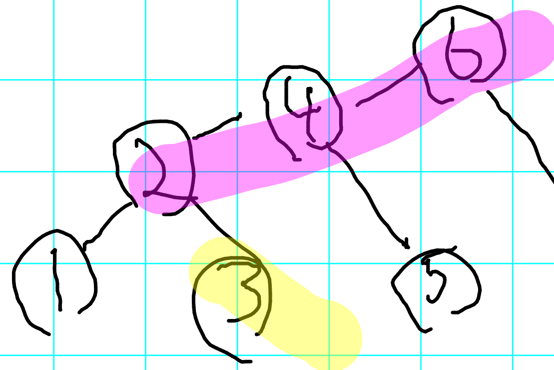


$X = 1, 4, 5, 7, 6, 2, 3$

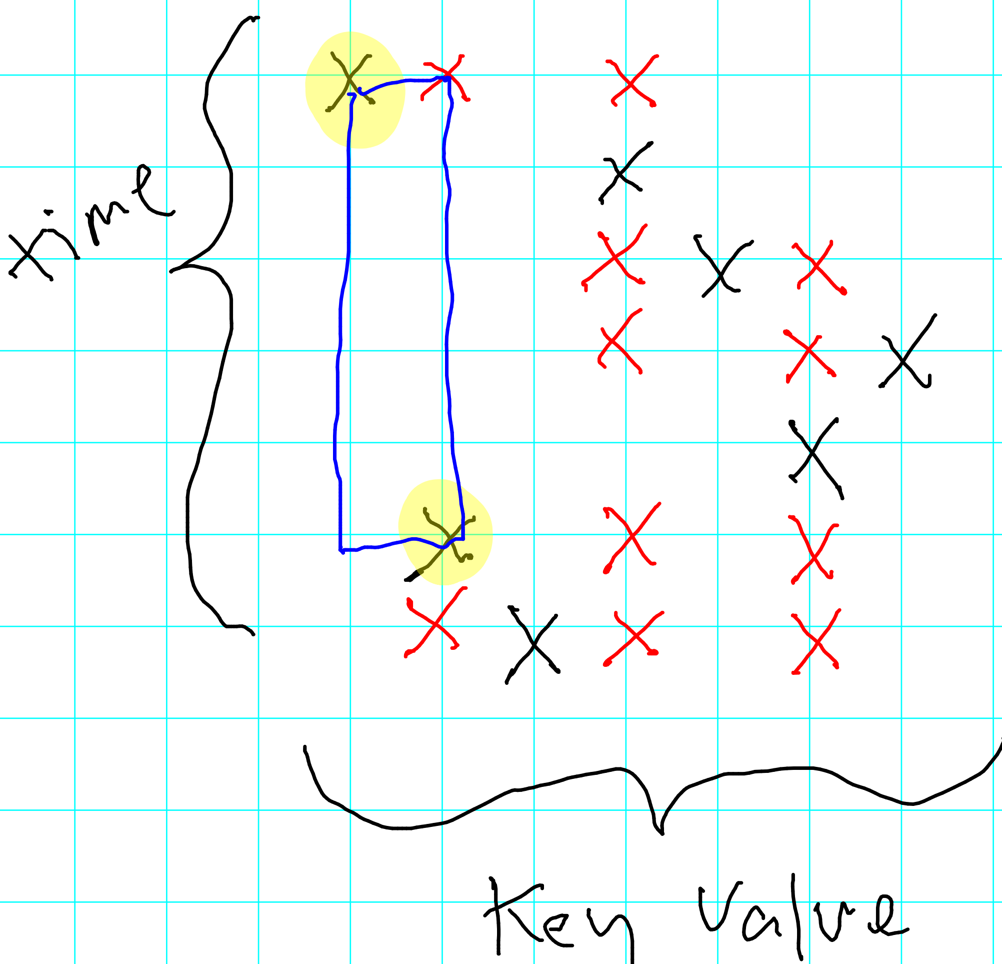



Can you find this

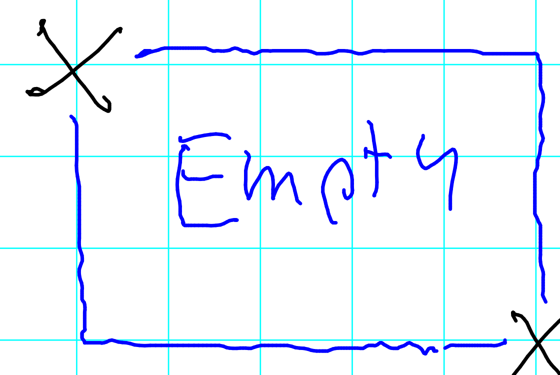





X = 1, 4, 5, 7, 6, 2, 3




Can you find this




No!

# Box Model {OHIP, unpublised}

- No empty boxes $\leftrightarrows$ "ortogonally satisfied"

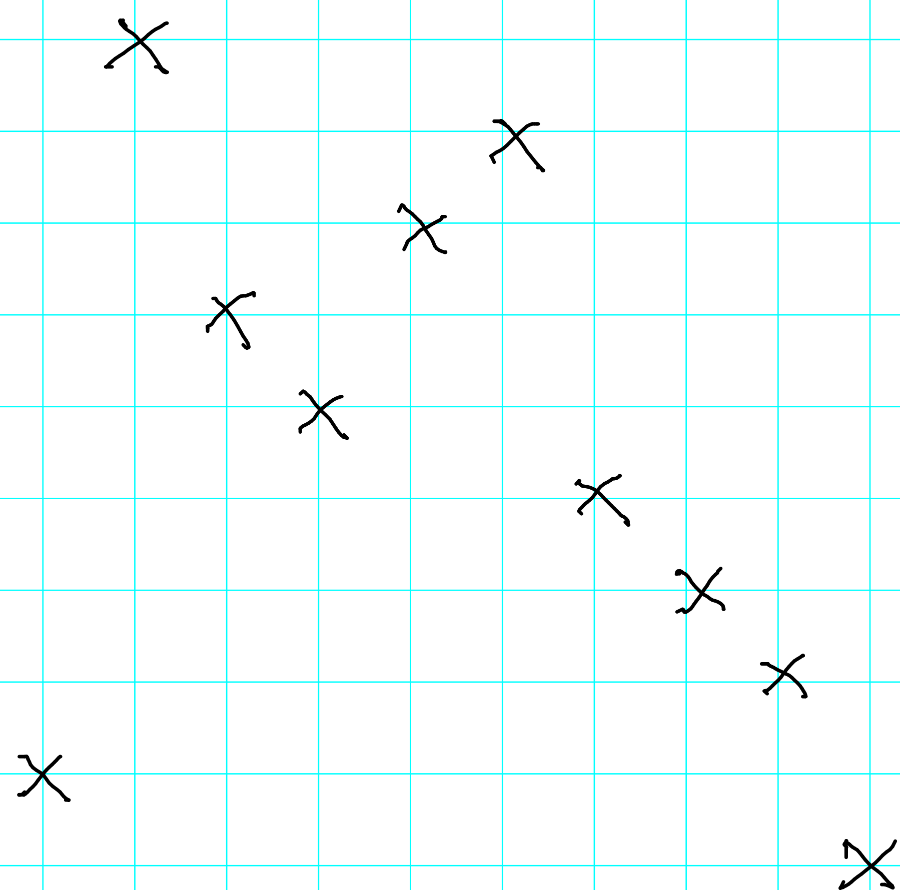
# Box Model

- No empty boxes $\Leftrightarrow$ "ortogonally satisfied"

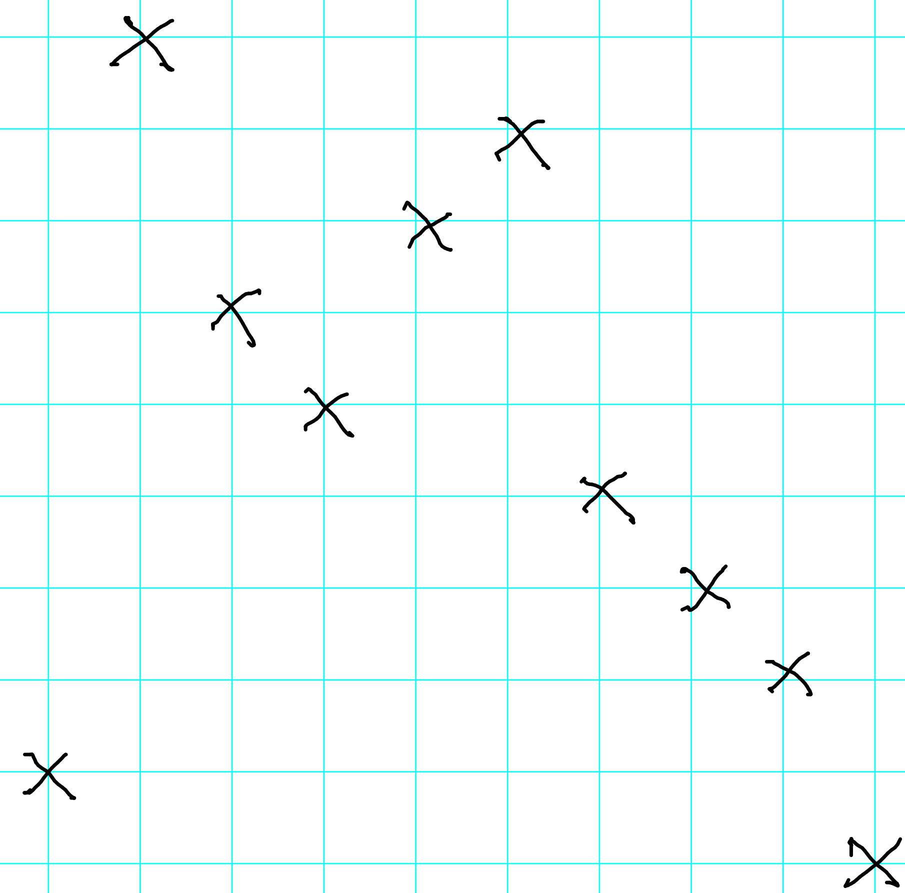
- All BST $\longrightarrow$ orthogonally satisfied

- orthagonally satisfied $\longrightarrow$ BST

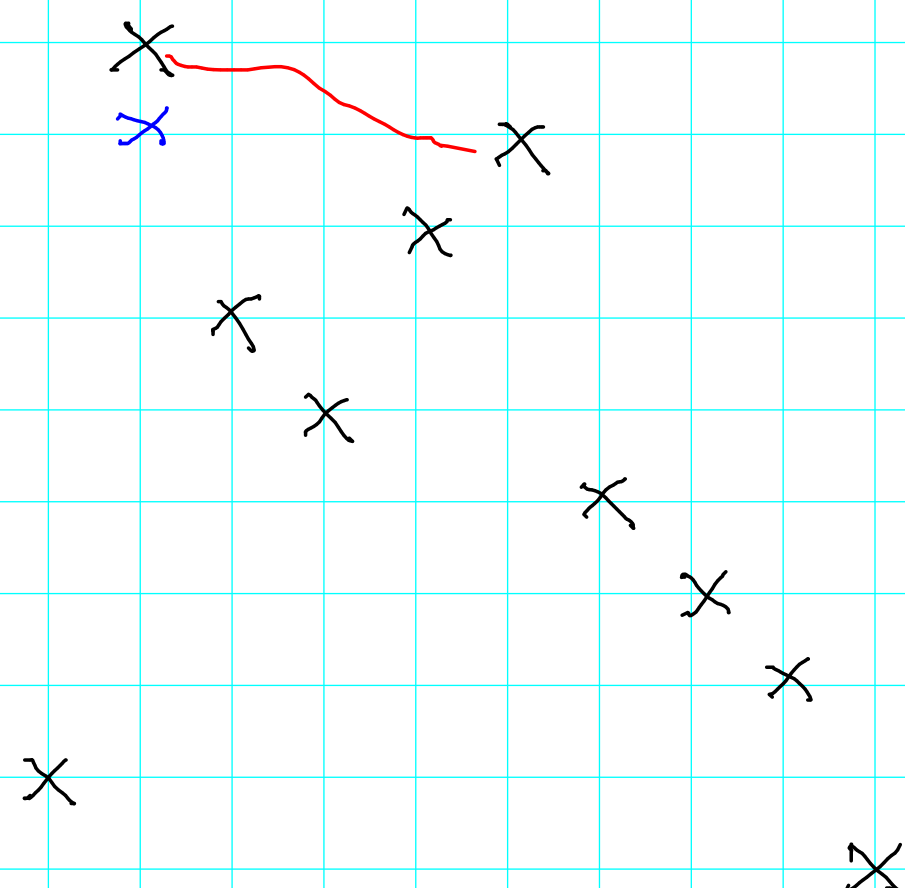
Minimal OS Superset = DynOPT



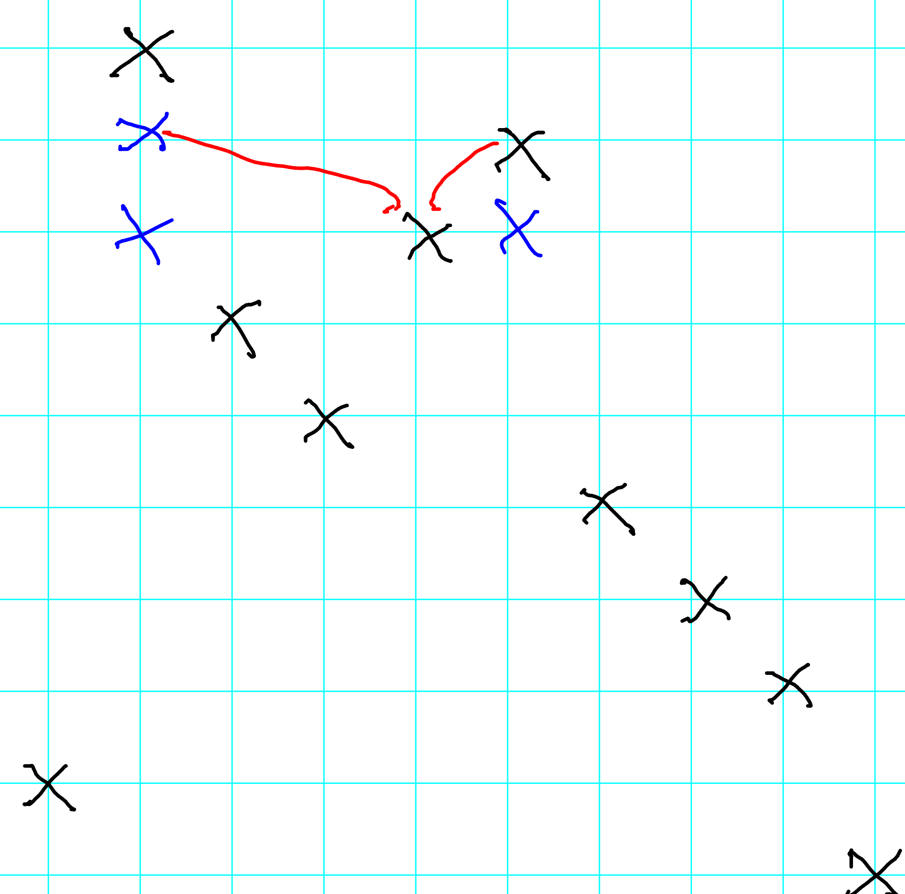
# One Idea: Greedy



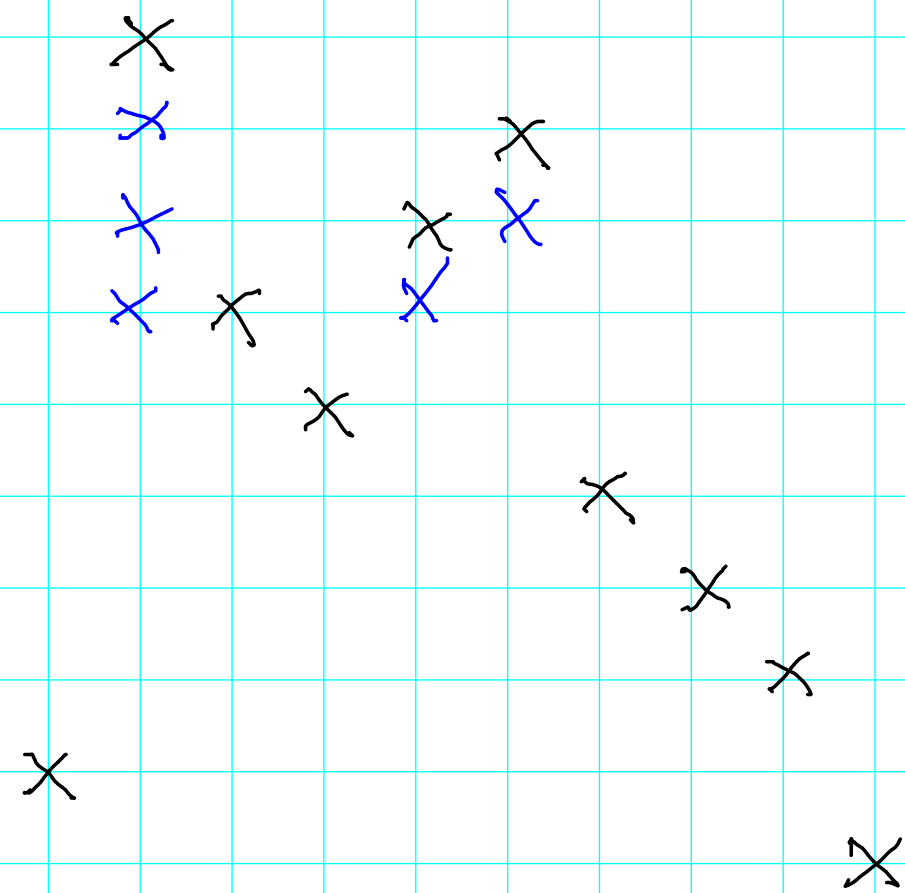
One Idea: Greedy



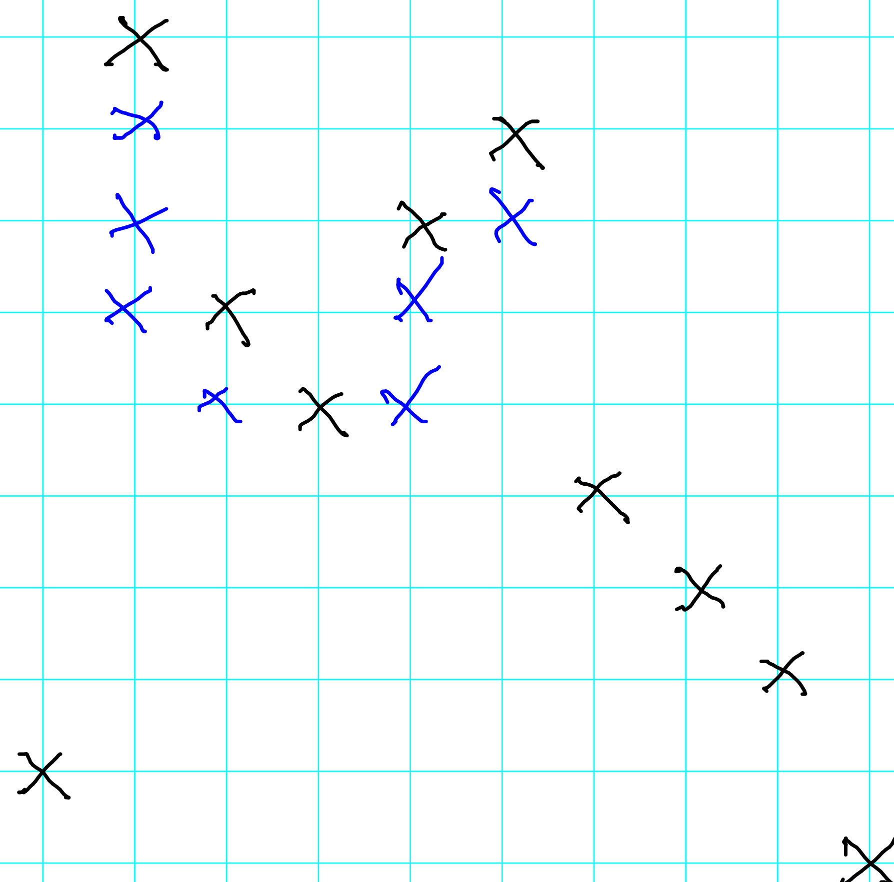
One Idea: Greedy



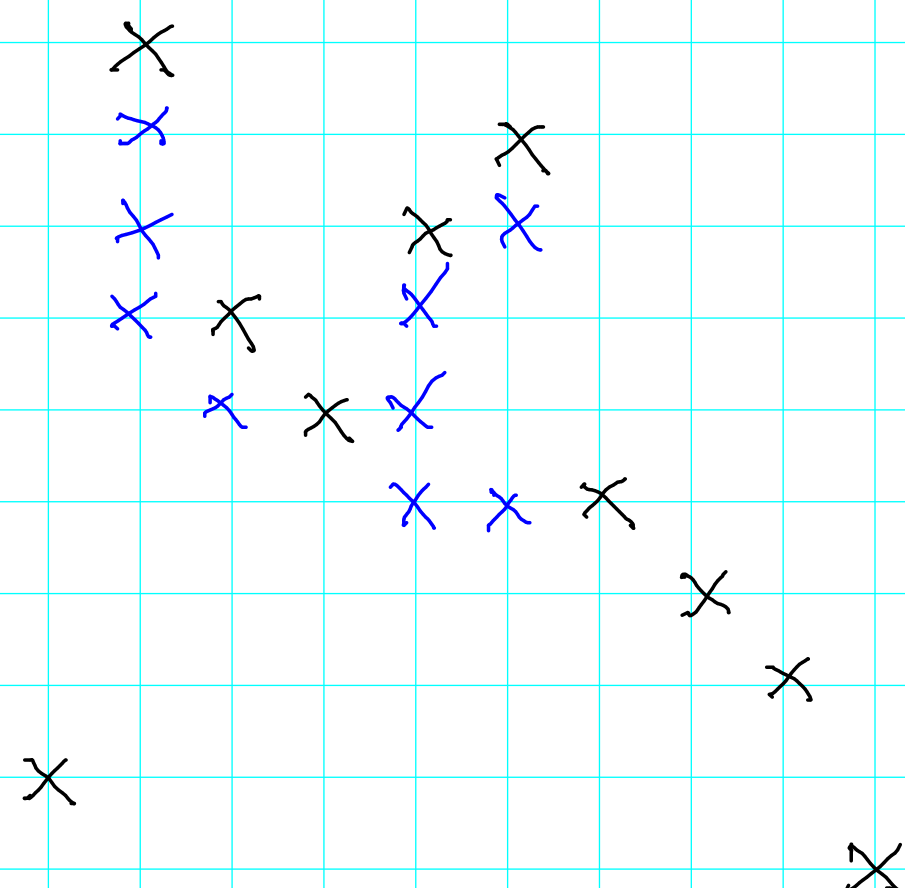
One Idea: Greedy



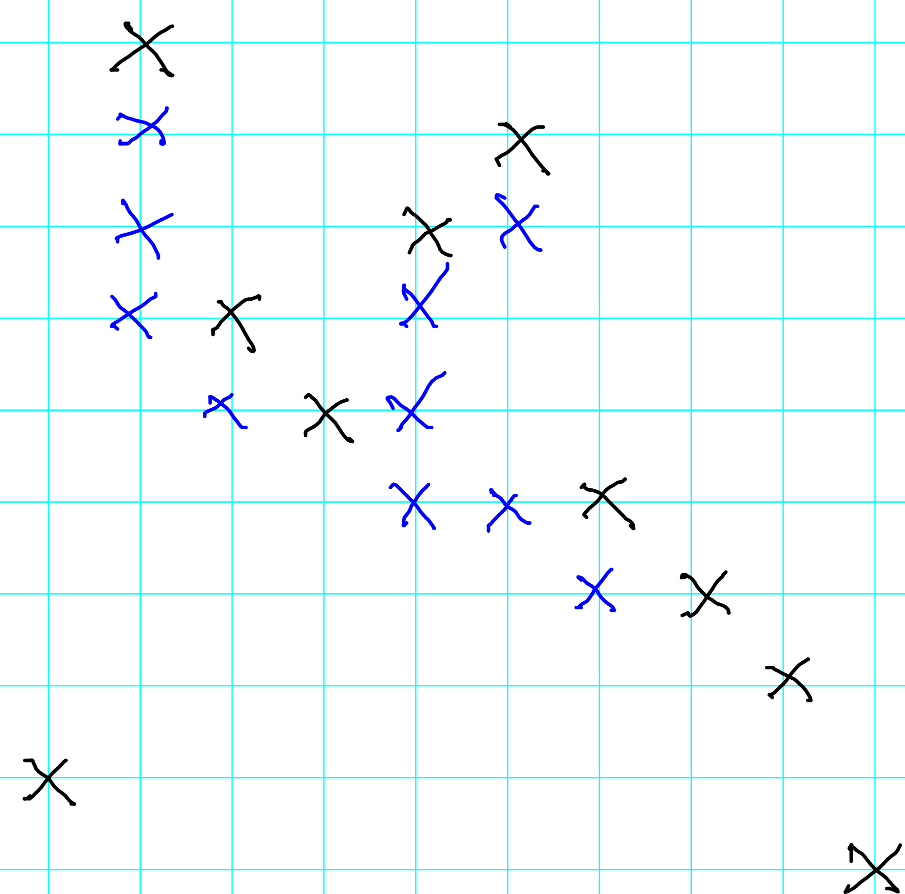
One Idea: Greedy



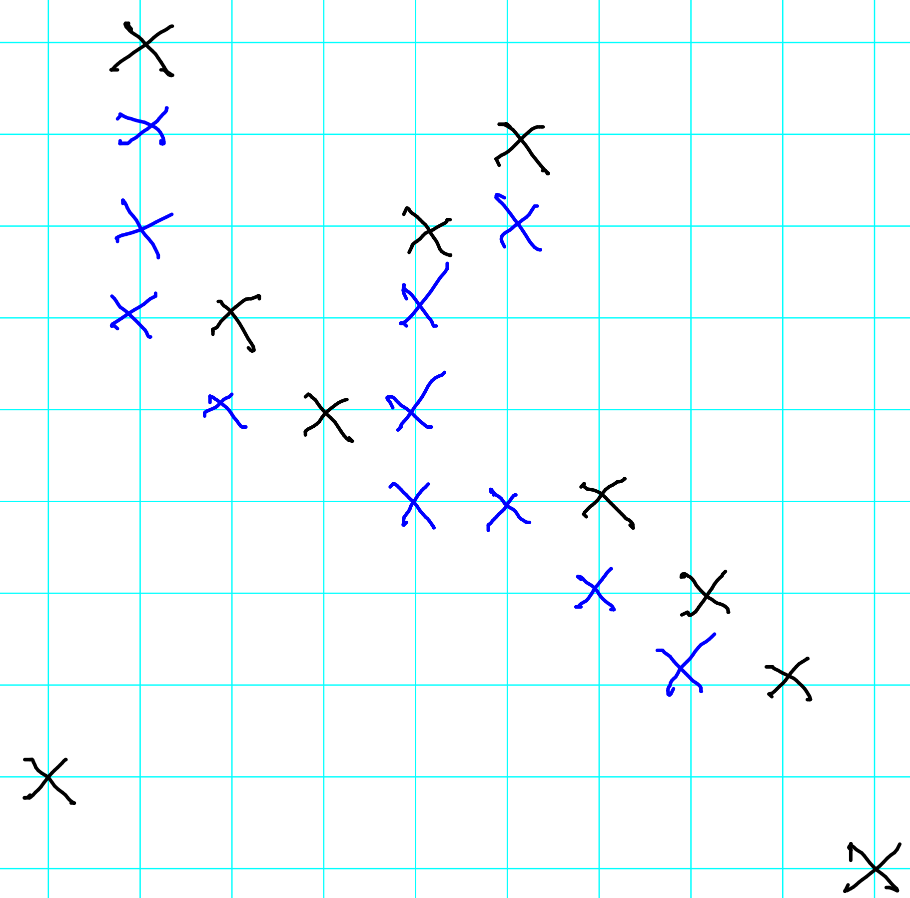
One Idea: Greedy



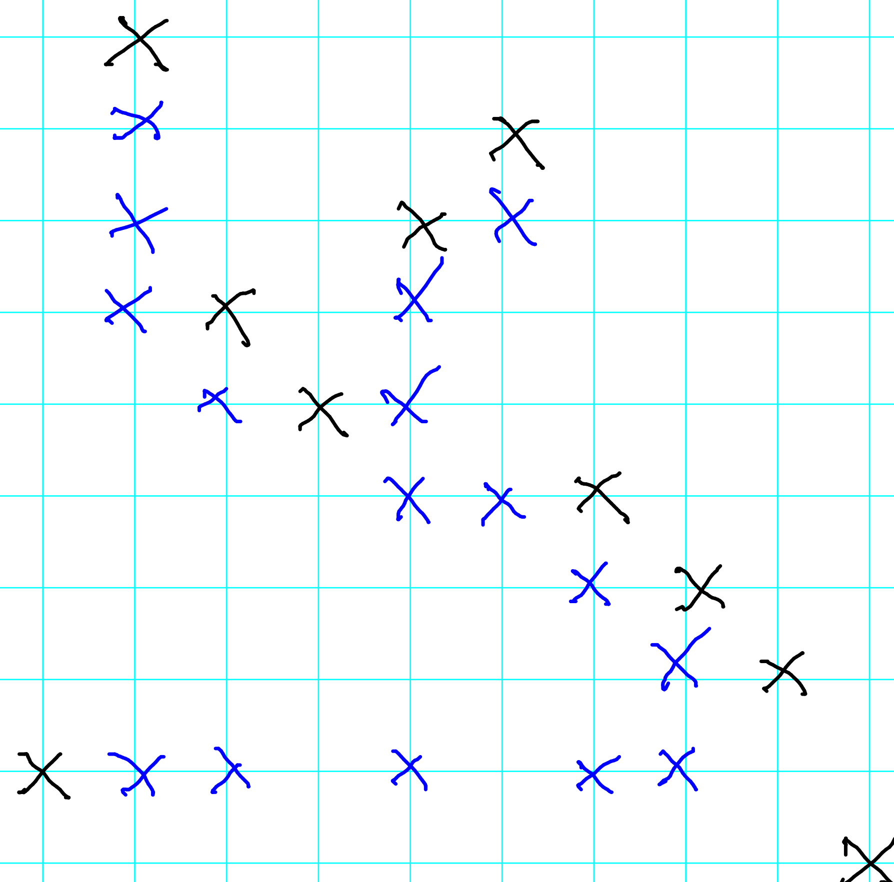
One Idea: Greedy



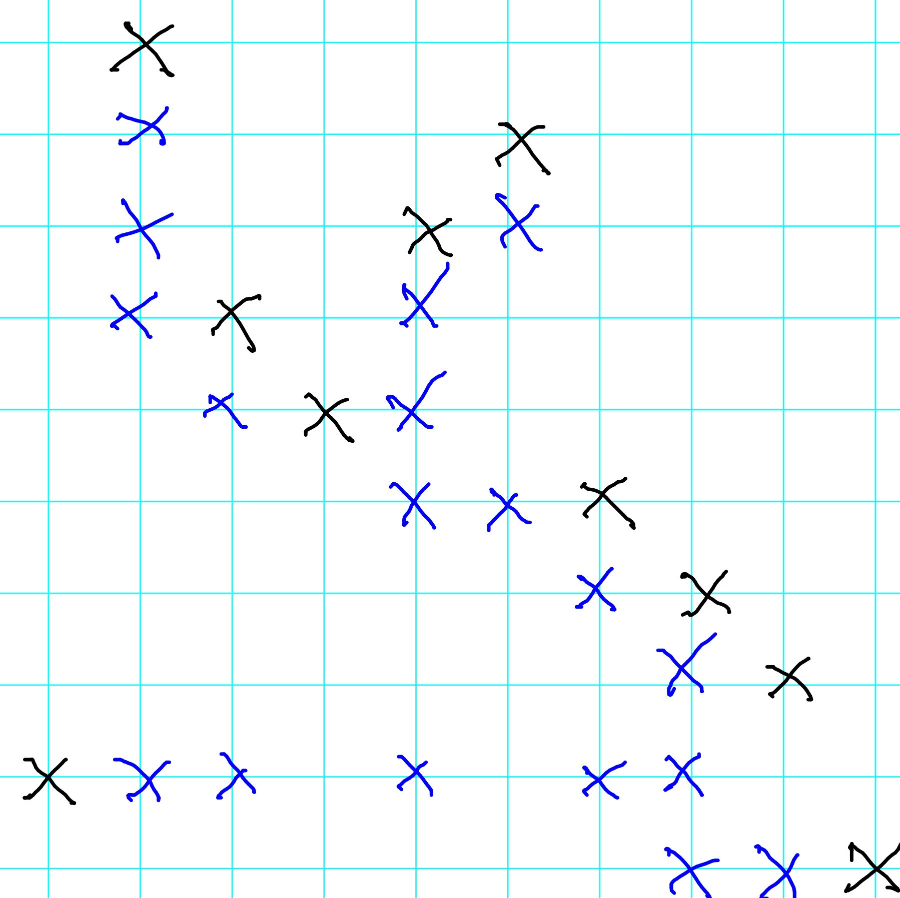
One Idea: Greedy



One Idea: Greedy



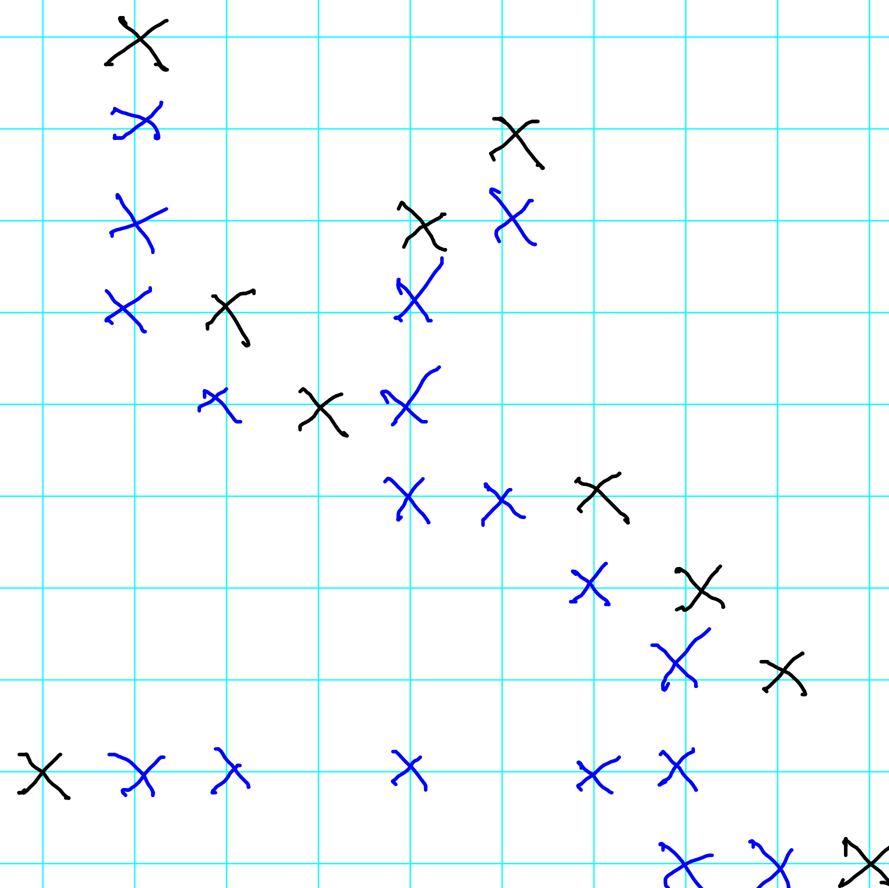
# One Idea: Greedy



<u>Observation</u>

This algorithm is IAN

# One Idea: Greedy



**Observation**

This algorithm is IAN

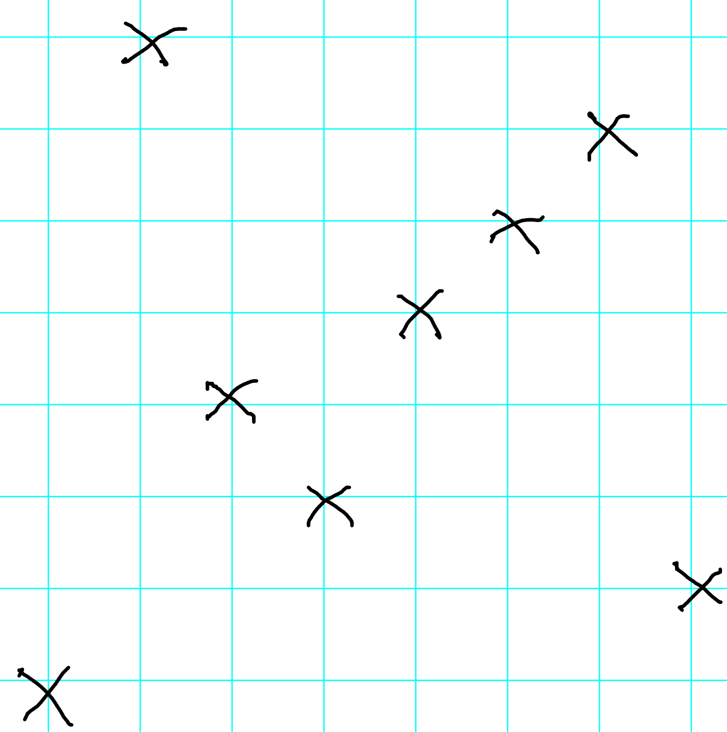
**Observation 2**

We can make an online BST with runtime $\Theta(IAN(x))$

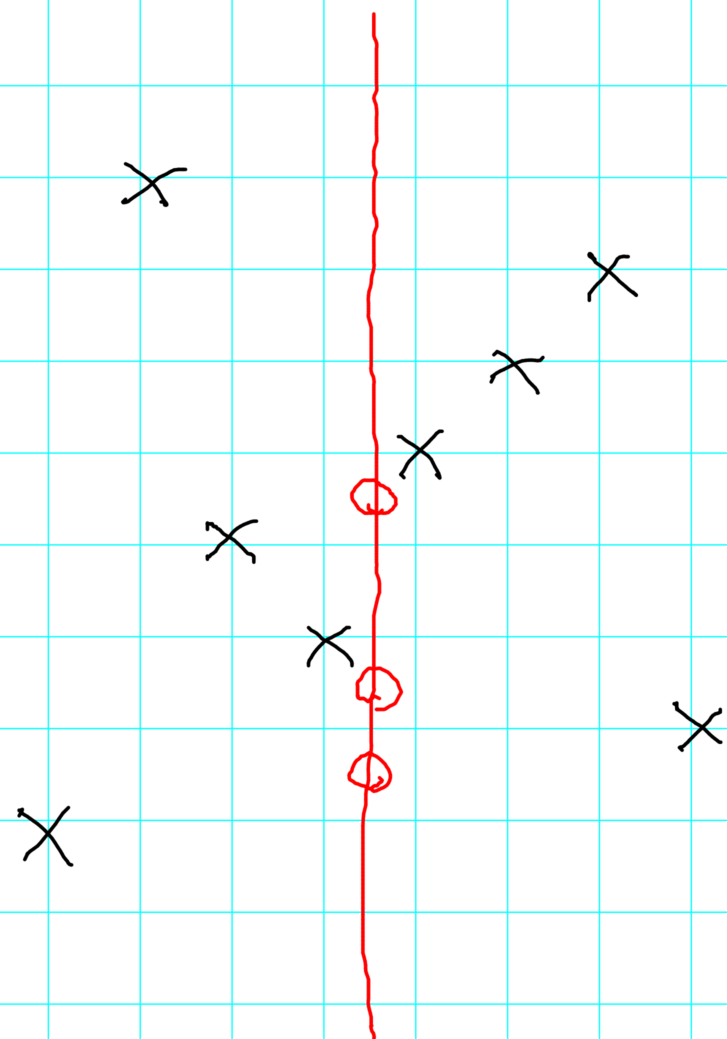
# Where were we?

- Lower bounds
  - WI
  - WII

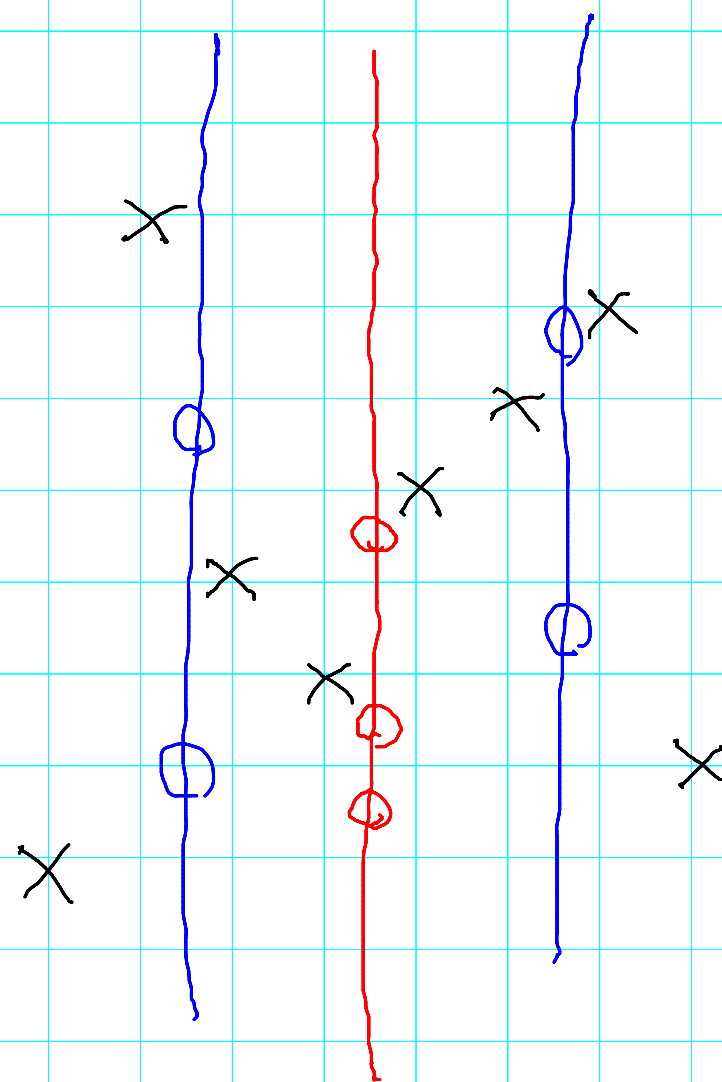
W I



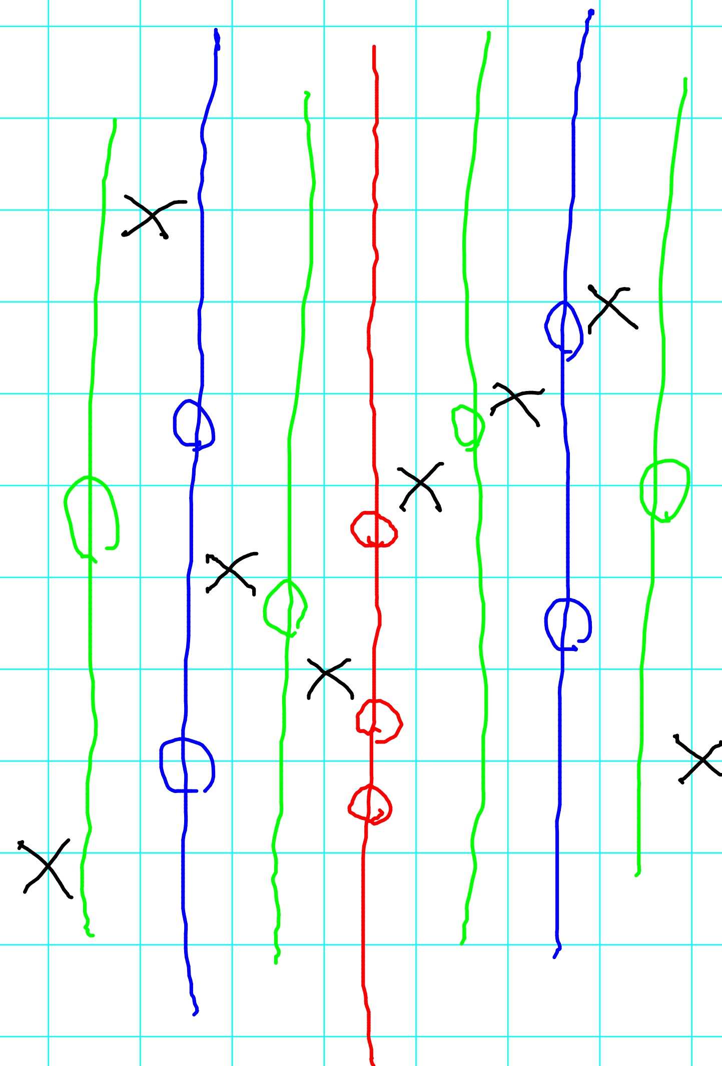
W I



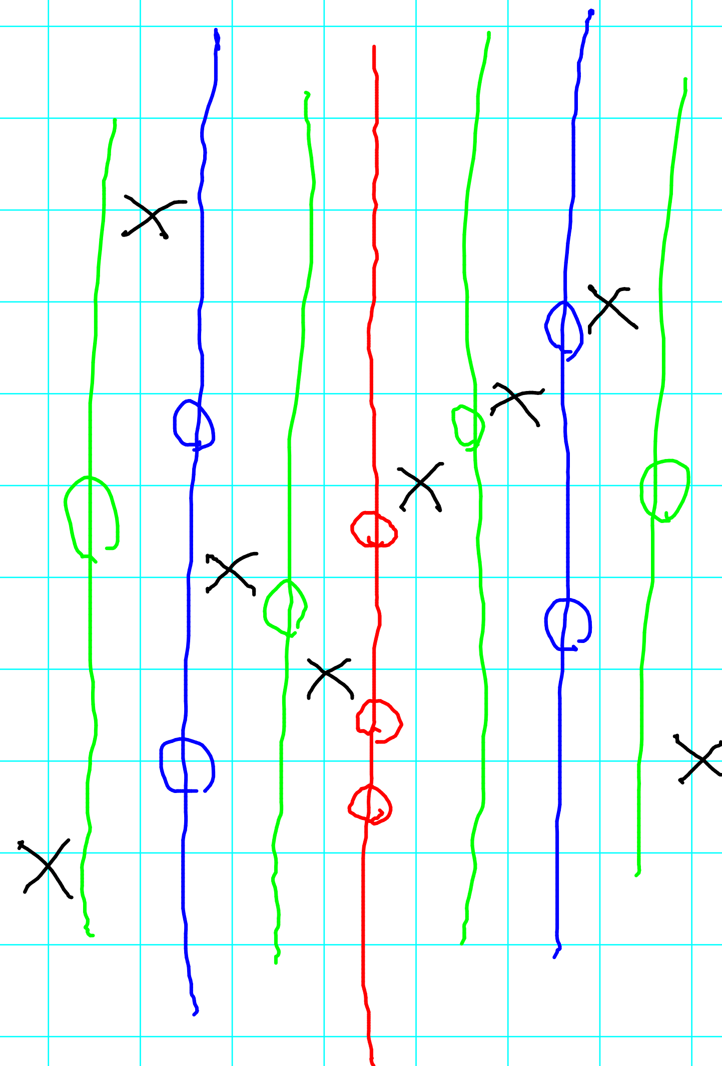
W I



# W I



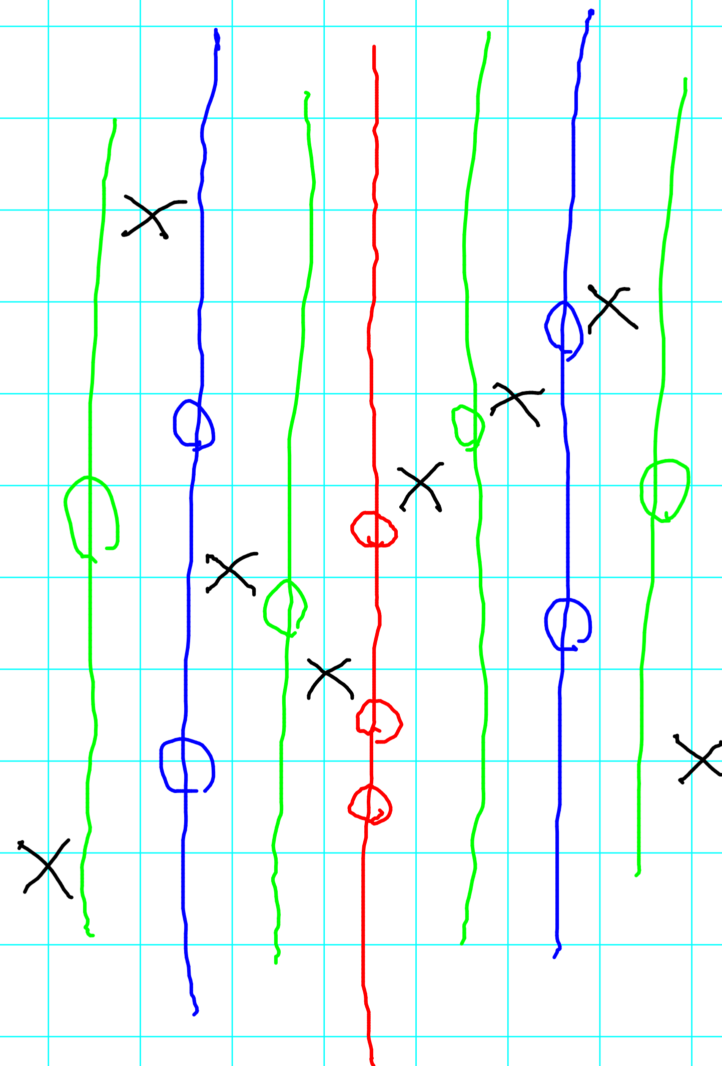
# WI



$WI(x)$ = # of circles

$OPT(x) = \Omega(WI(x))$

# WI

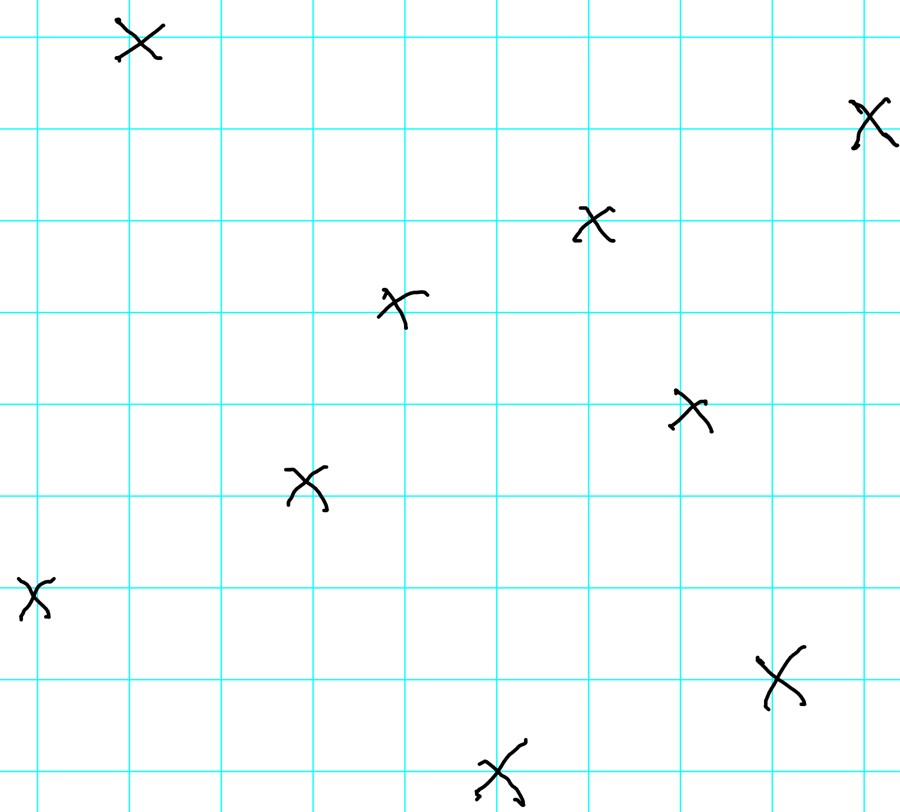


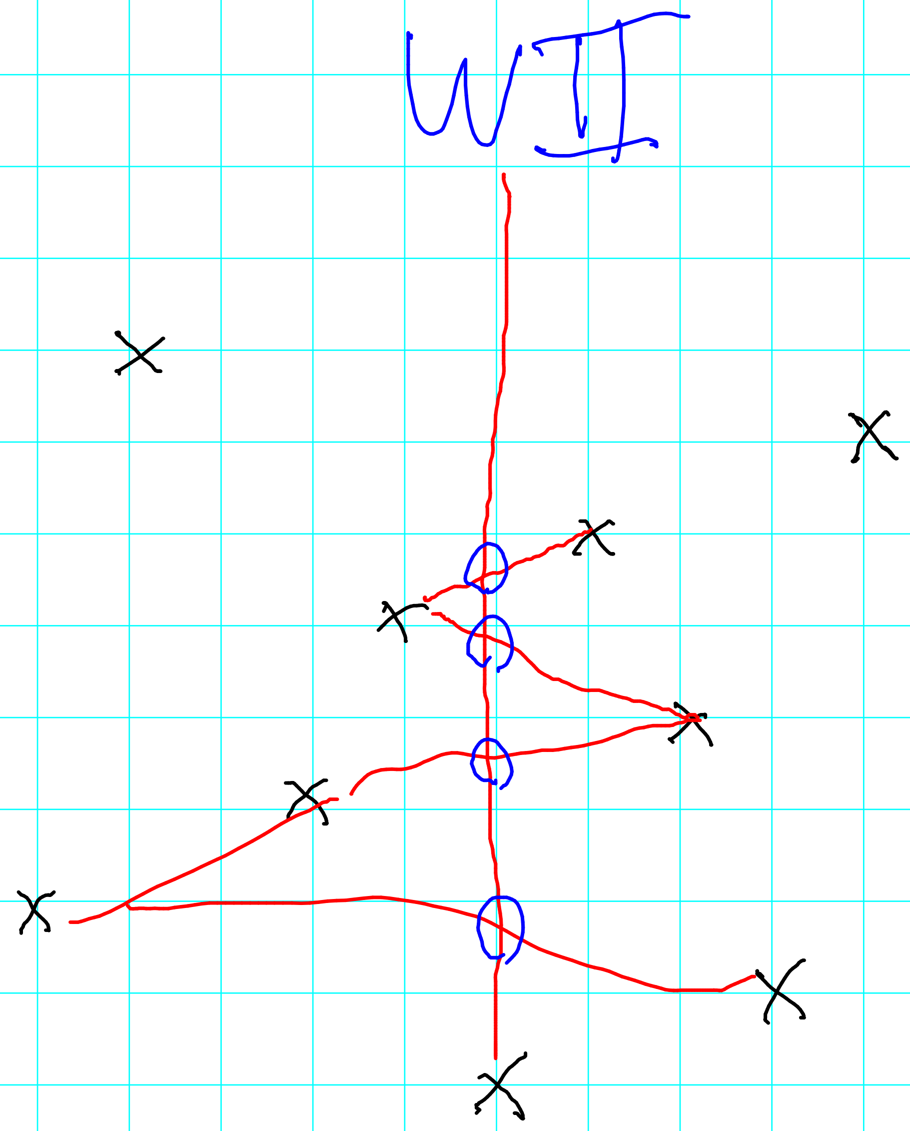
$WI(x) = \#$ of circles

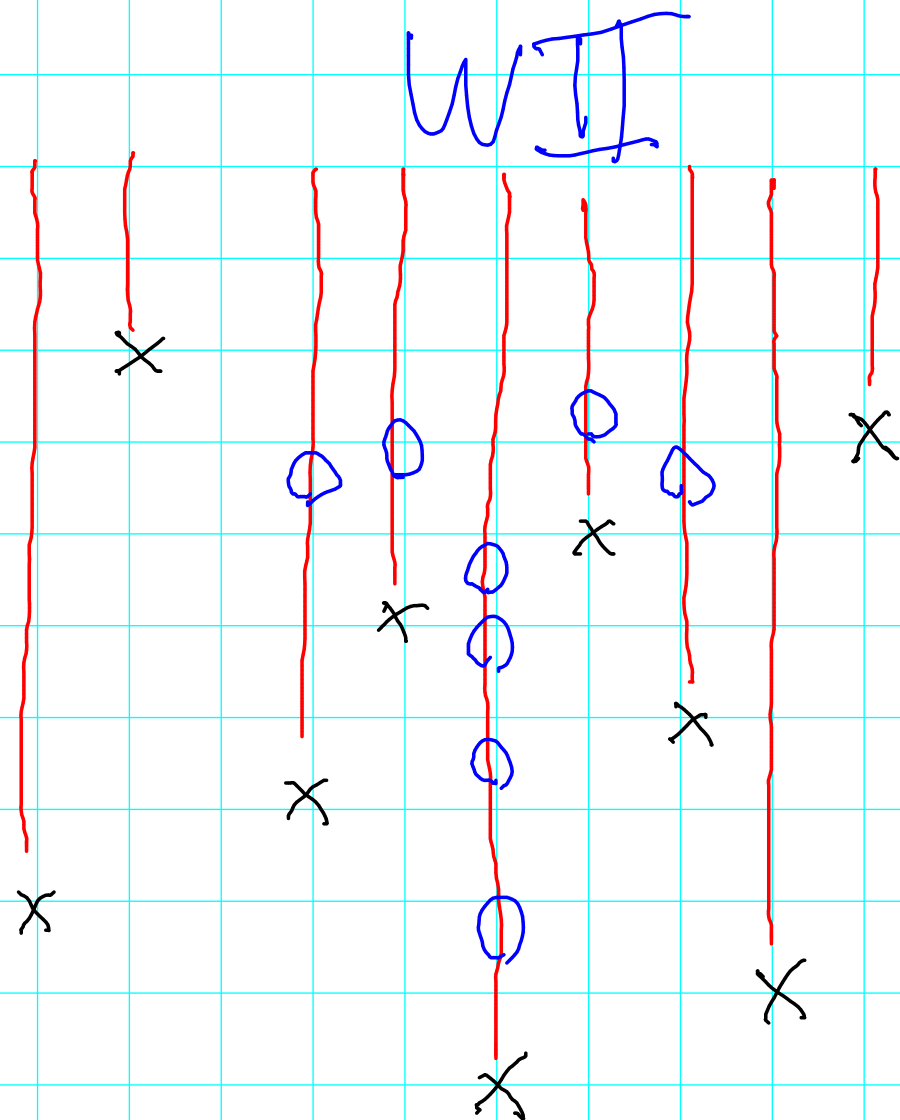
$OPT(x) = \Omega(WI(x))$

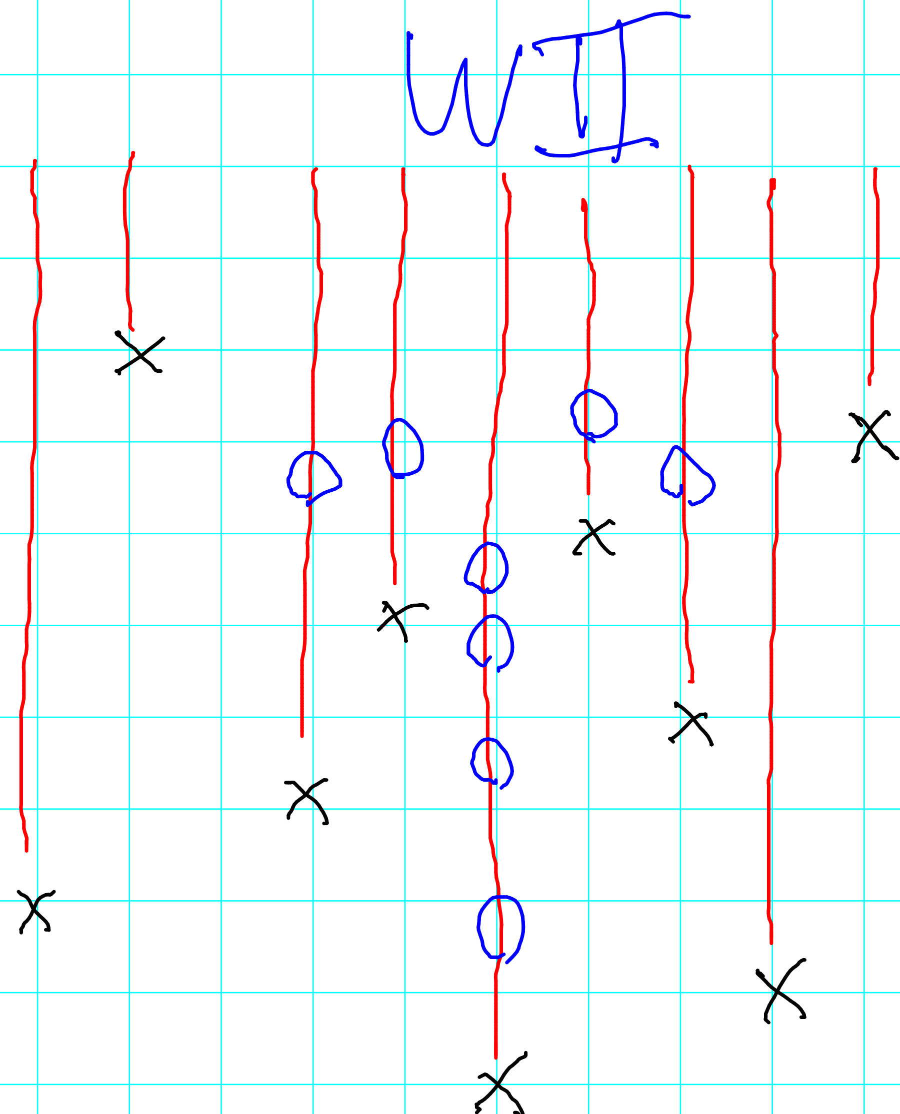
$TANGO(x) = O(\log \lg n \; WI(x))$

WI



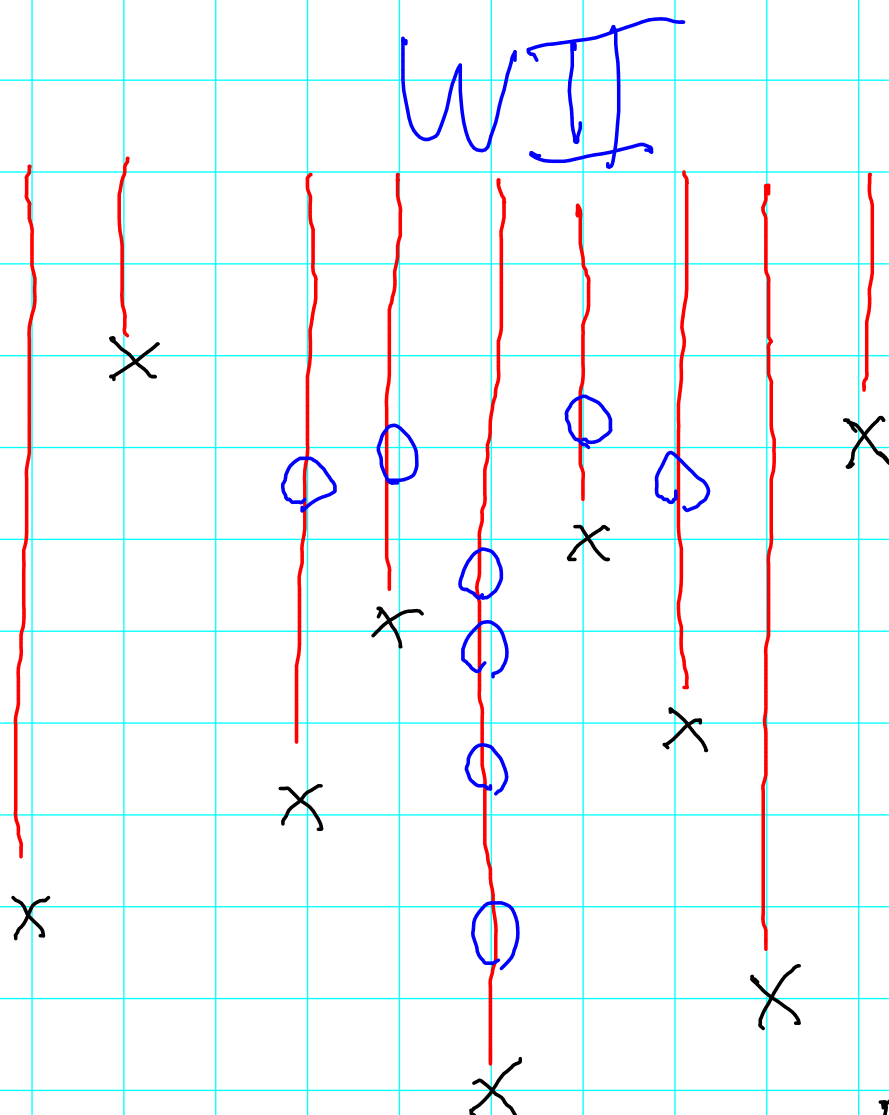

WJ


WI




$WII(X)$ = # of circles

$OPT(X) = \Omega(WII(X))$



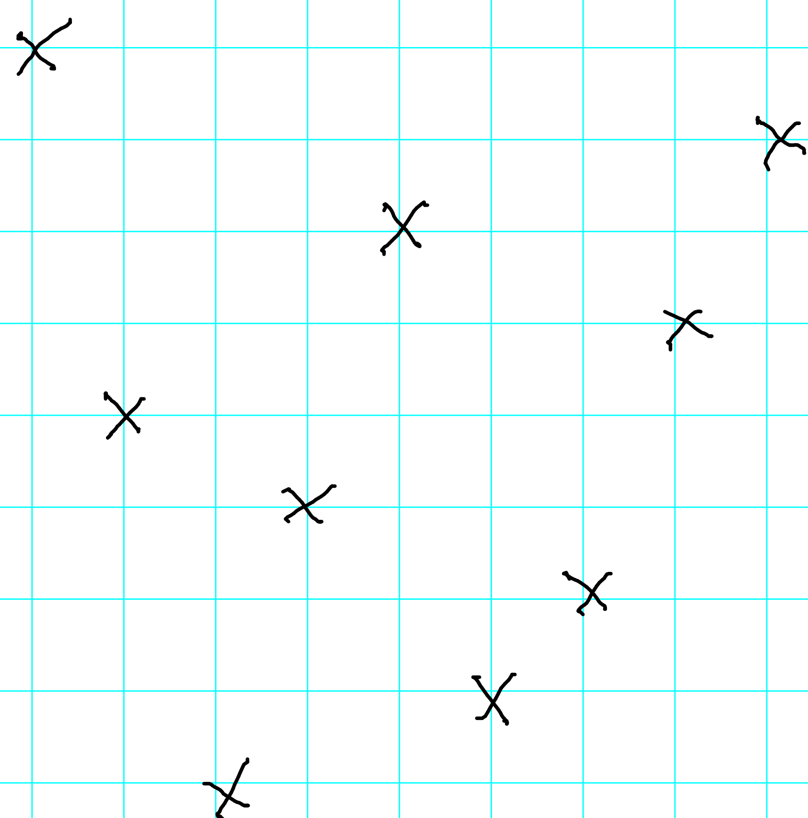

$WII(X)$ = # of circles

$OPT(X) = \Omega(WII(X))$

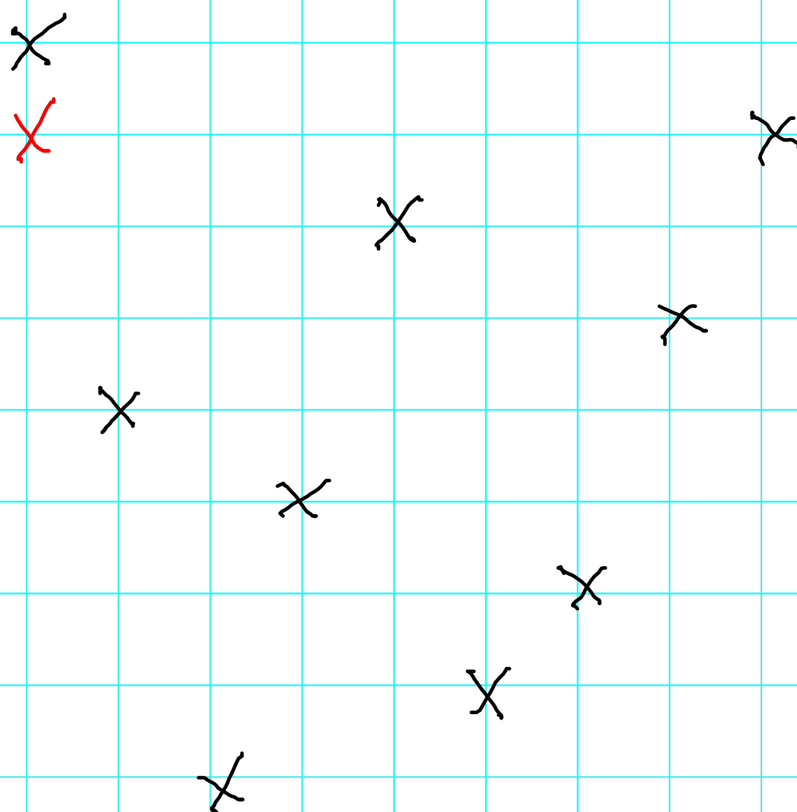
Don't know which is better, WI or WII.

Conj: WII is better and is tight

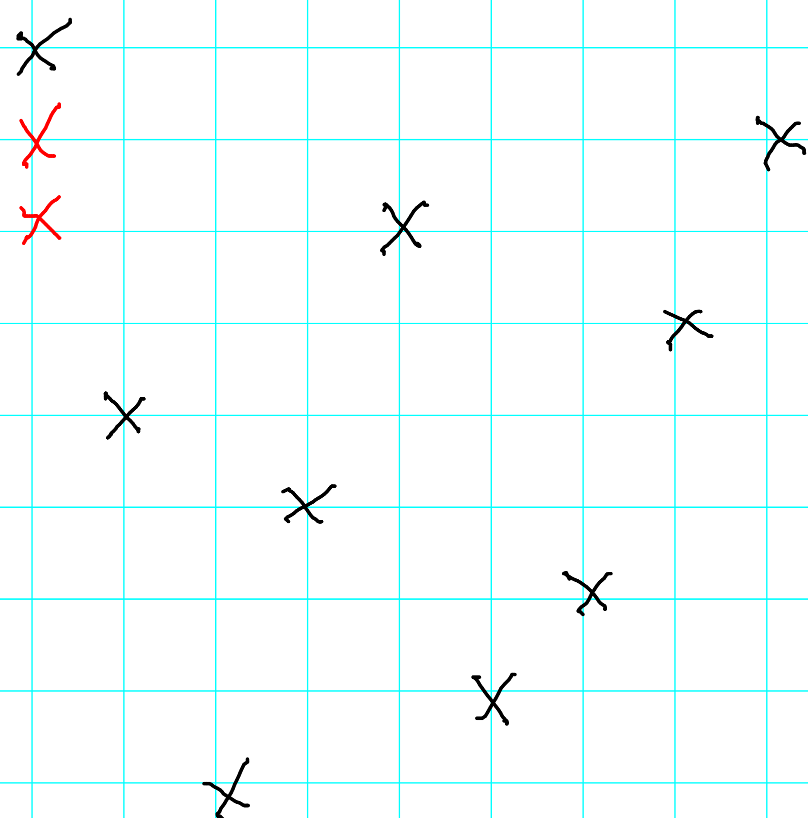
NISIAN [Horman's thesis]



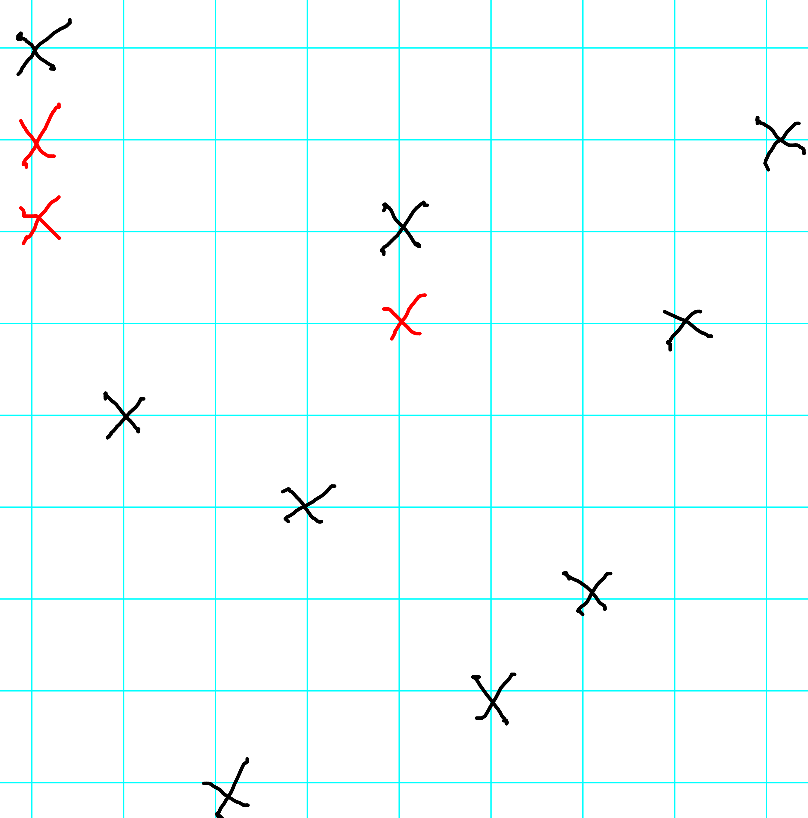
NISIAN



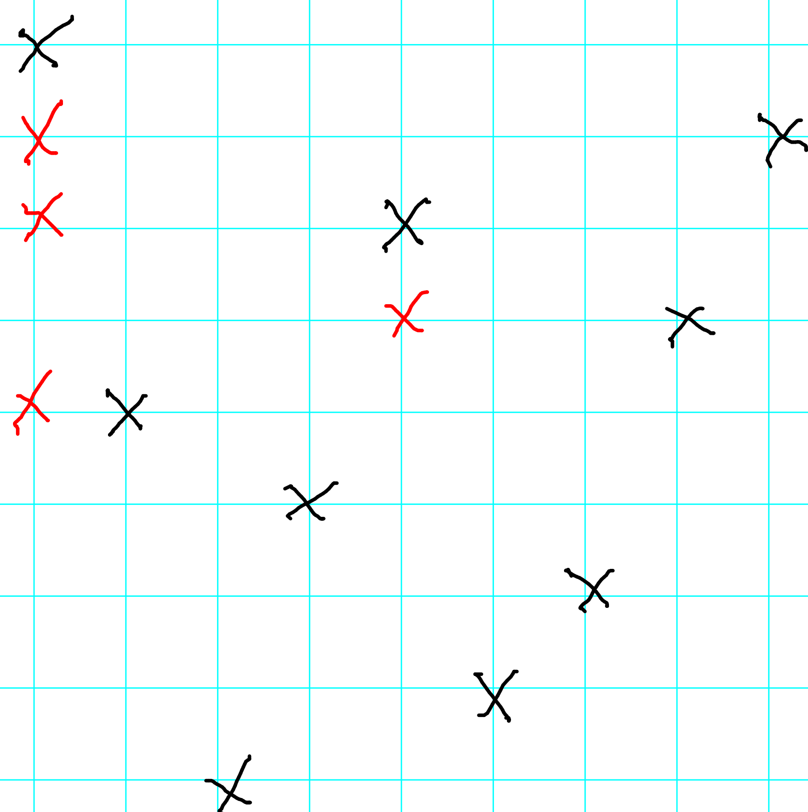
NISIAN



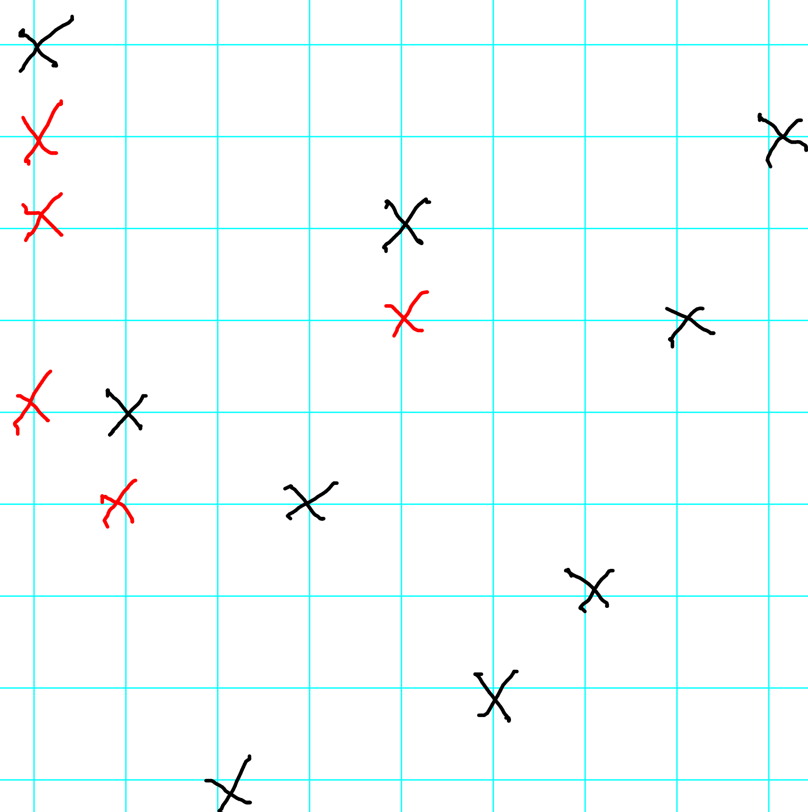
NISIAN



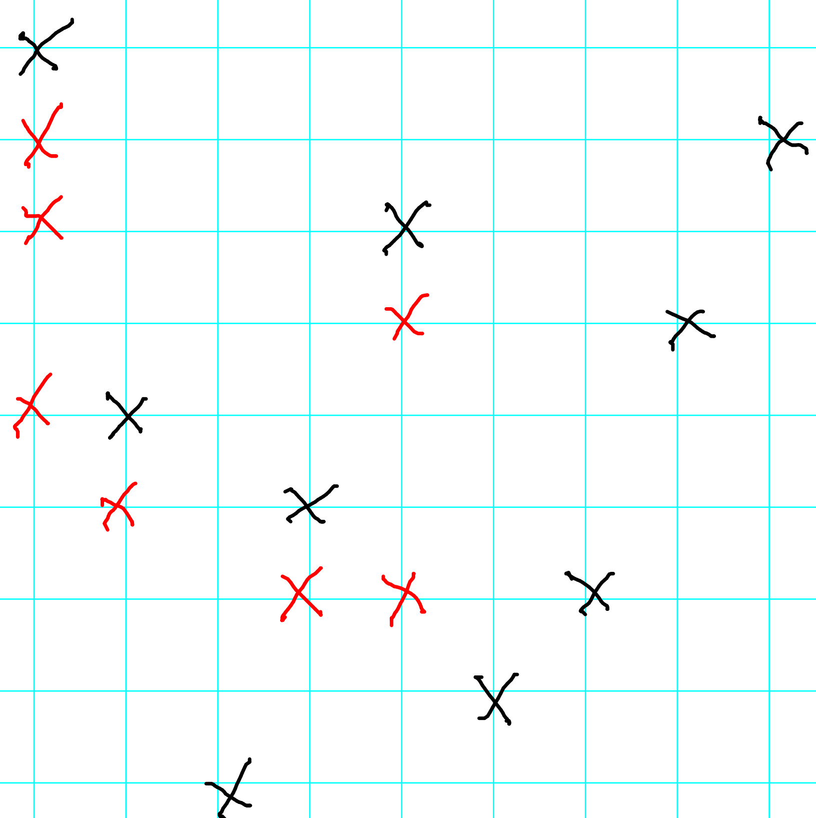
NISIAN



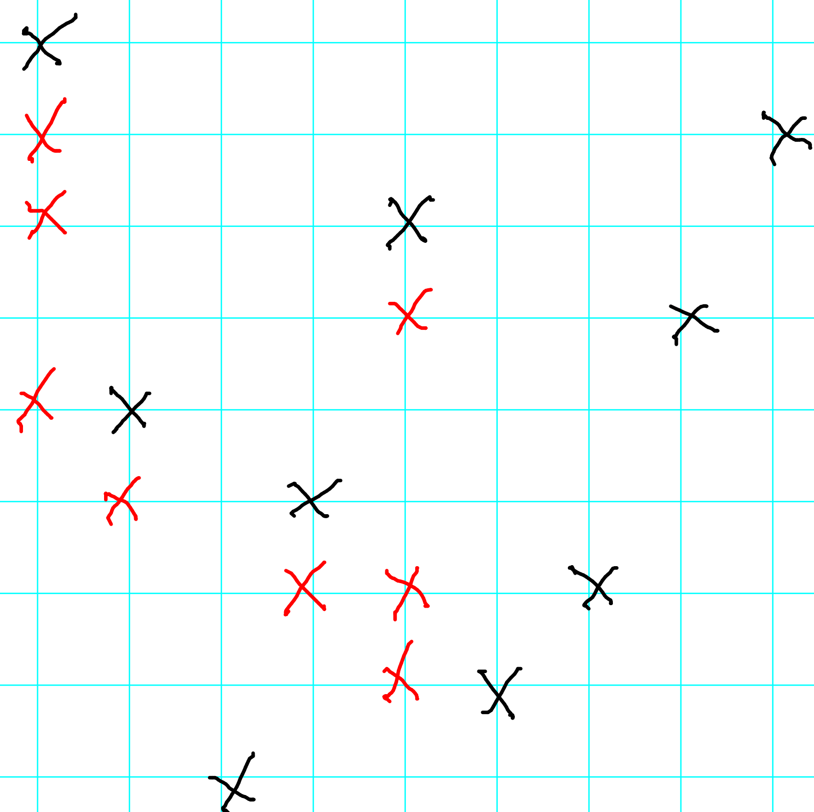
NISIAN



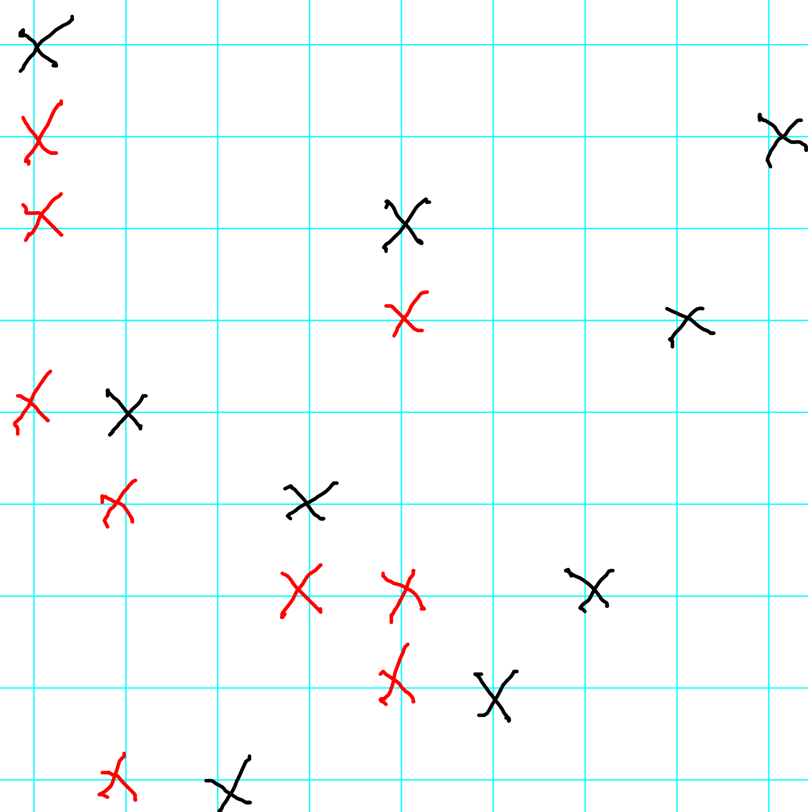
NISIAN



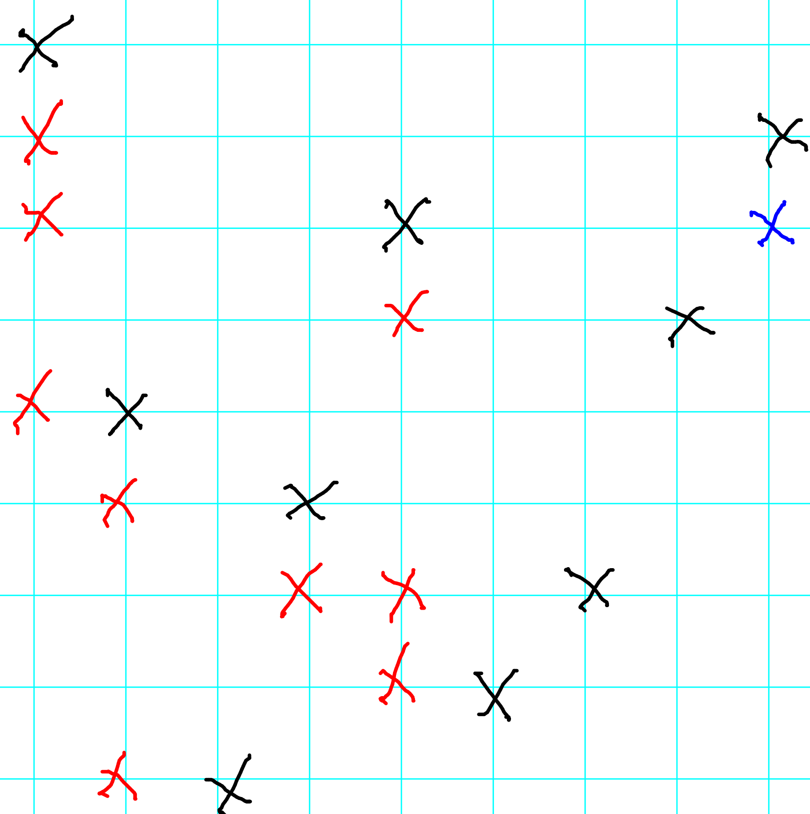
NISIAN



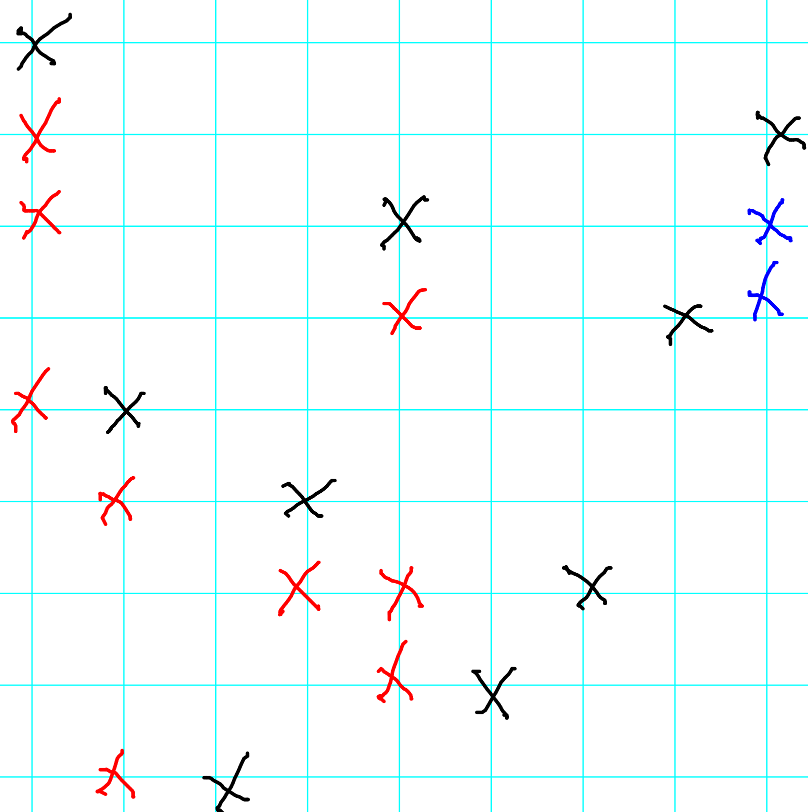
NISIAN



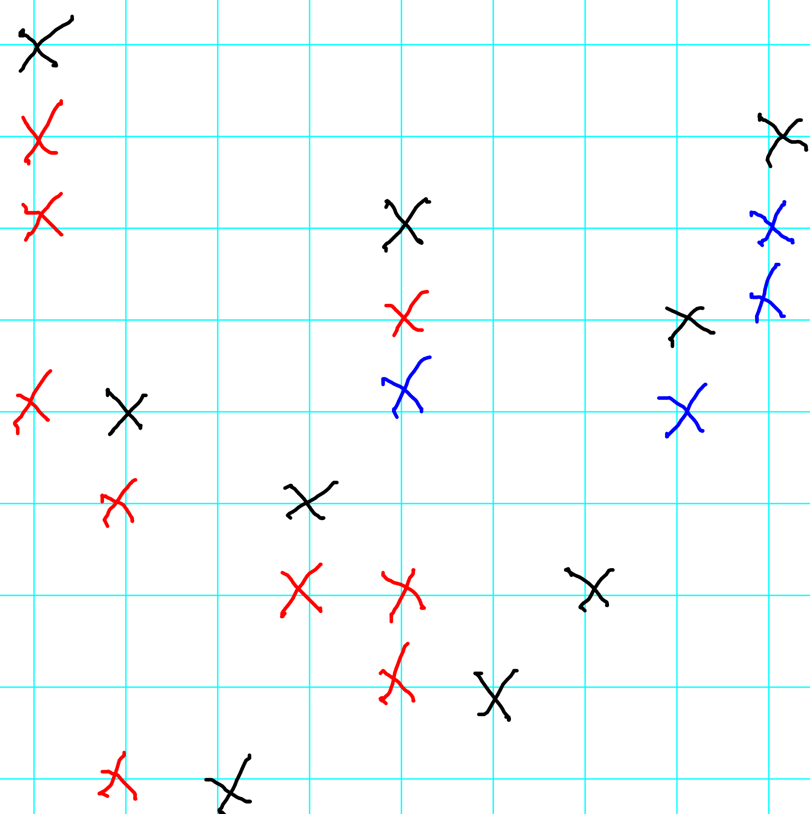
NISIAN



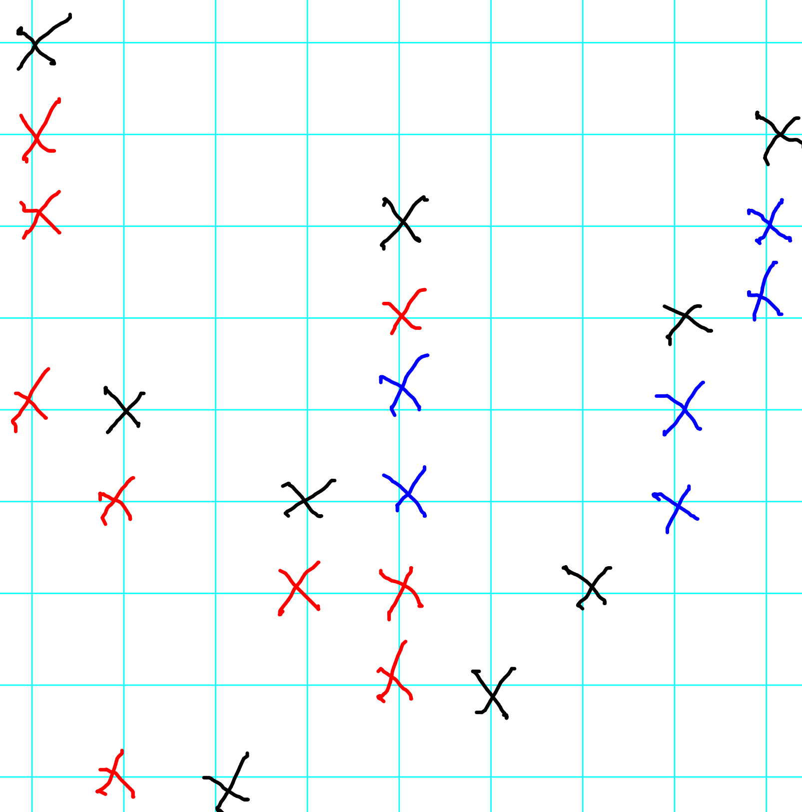
NISIAN



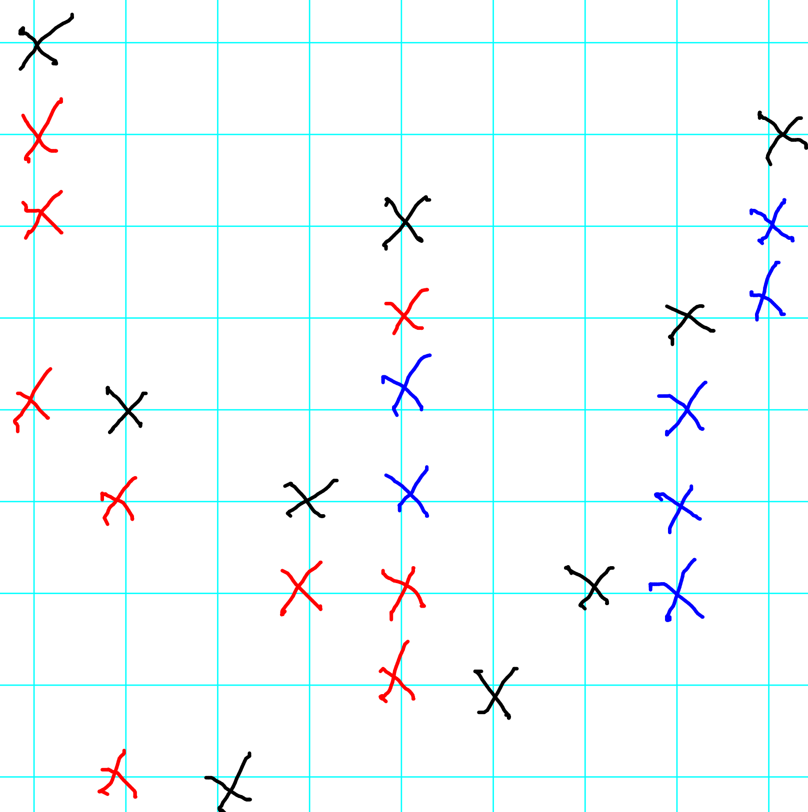
NISIAN



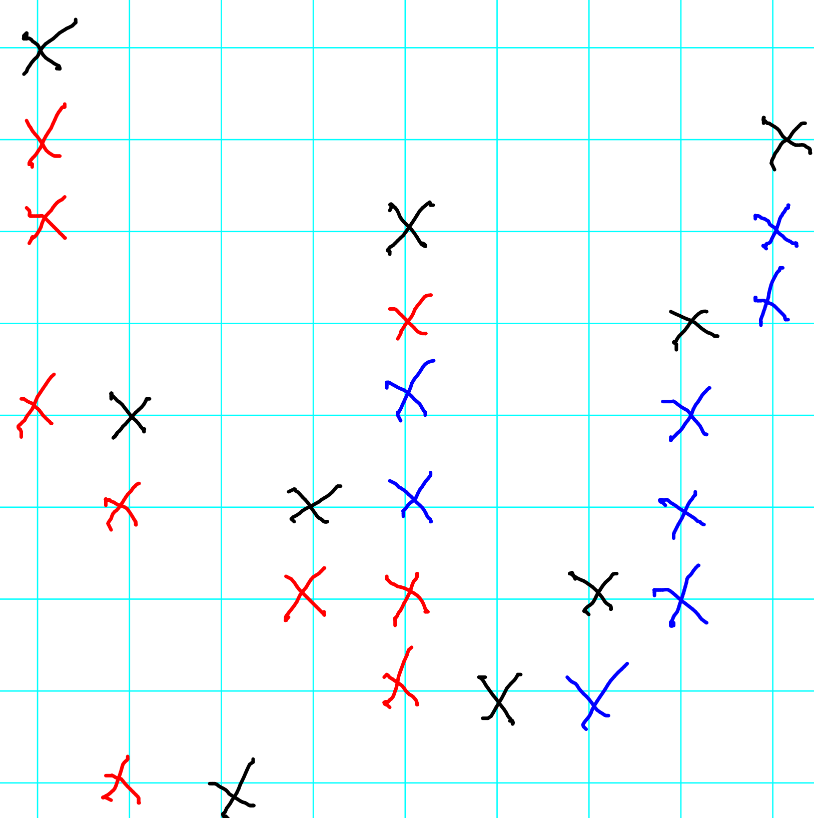
NISIAN



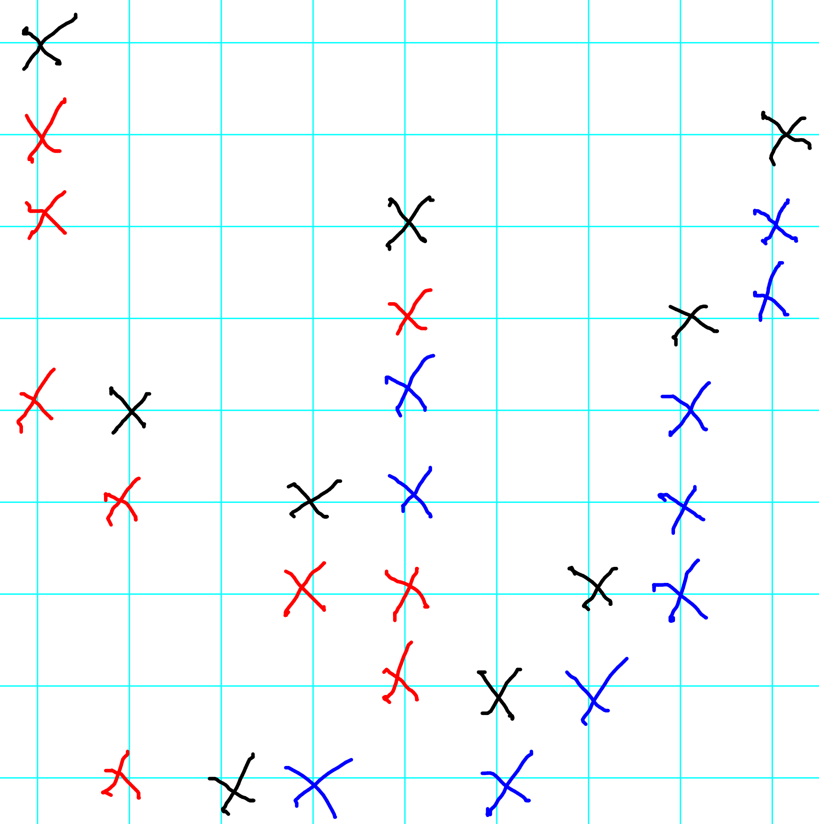
NISIAN



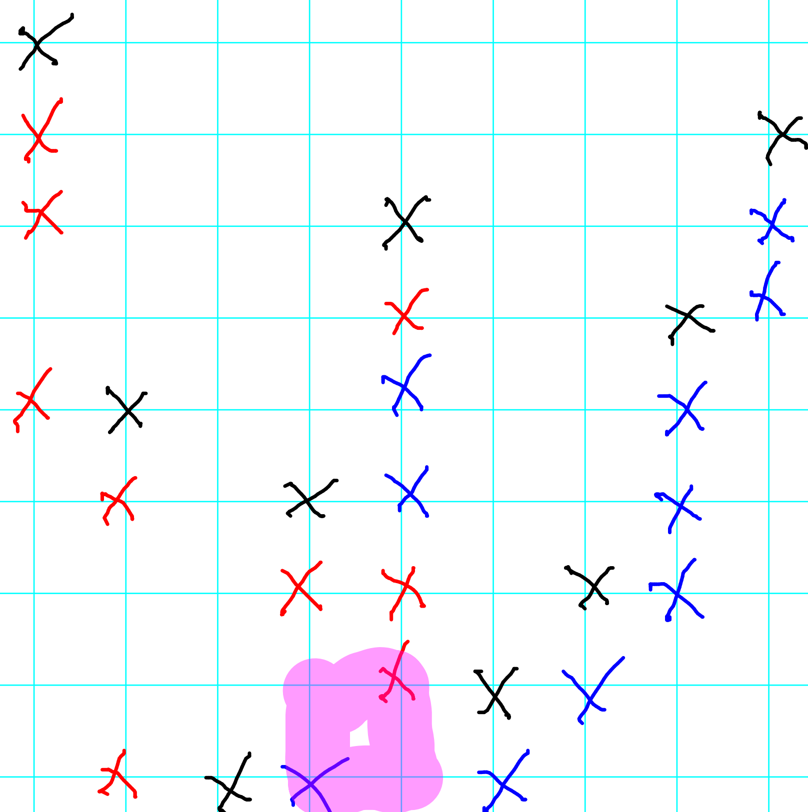
NISIAN



NISIAN

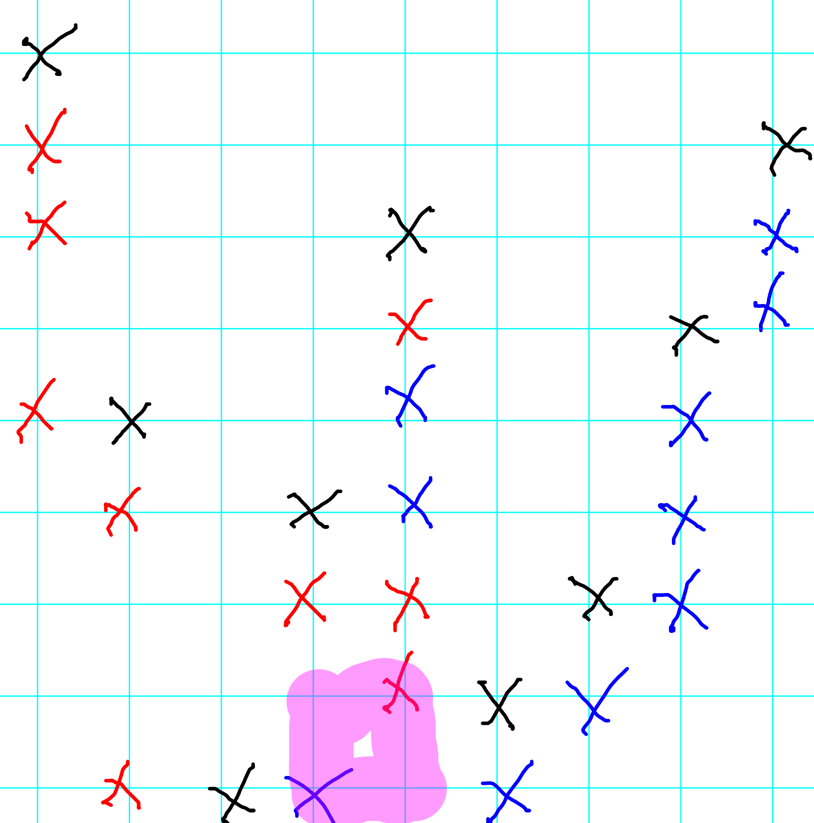


NISIAN



Not OS

# NISIAN



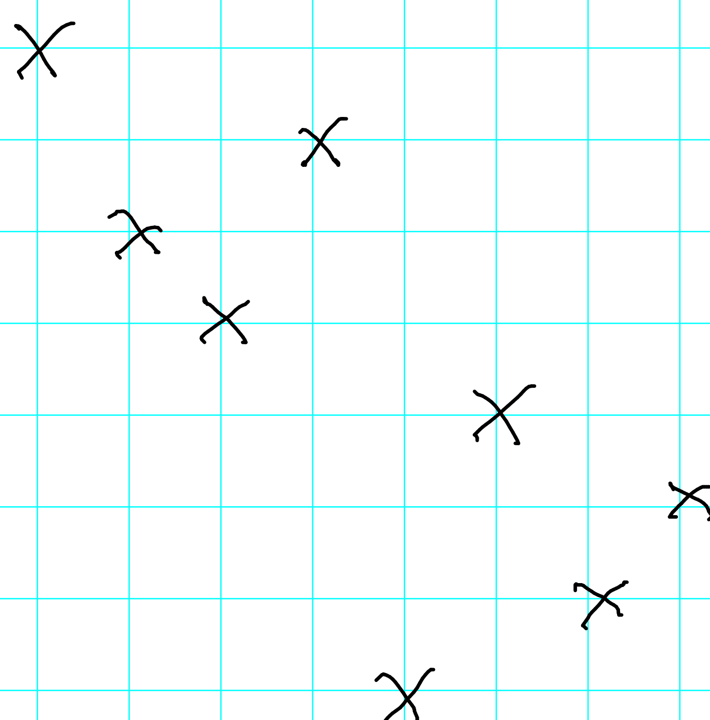
Not OS

But it is a LB!!

And no worse than W1, W2

$$\Omega(W1(X)) = NISIAN(X) = O(OPT(X))$$
$$\Omega(W2(X)) =$$

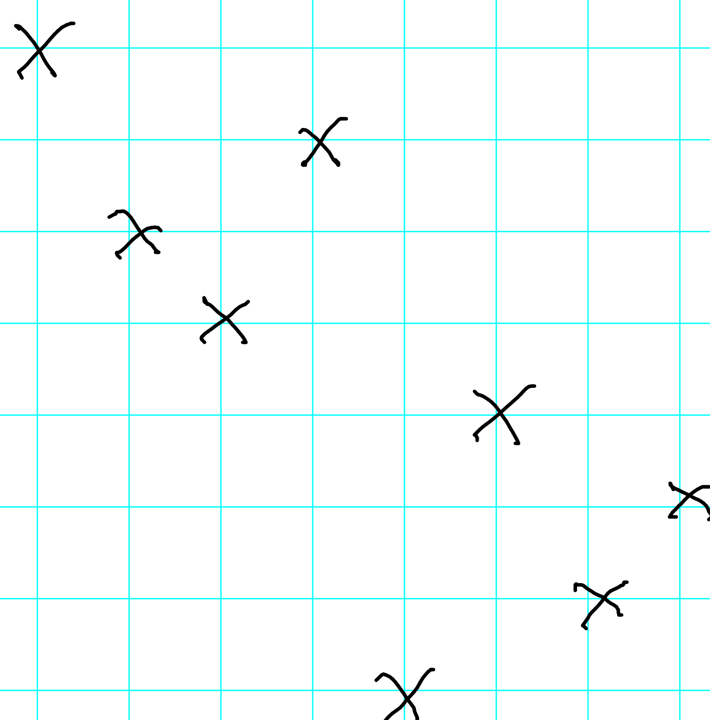
# Conclusion - Trees



A constant-factor minimal OS superset

Doesn't seem so hard

# Conclusion - Trees



A constant-factor minimal OS superset Doesn't seem so hard

(Exact is NPC for a variant)

# Heaps

- Prehistory
- Pairing Heaps
- Skew Heaps
- connections, alternatives

# Prehistory: Fibonacci Heaps

[Fredman + Tarjan 87]

| | Fib-Heap |
|---|---|
| Extract-Min | $O(\log n)$ |
| Insert | $O(1)$ |
| Decrease-Key | $O(1)$ |
| Meld | $O(1)$ |

# Prehistory: Fibonacci Heaps

| | Fib-Heap |
|---|---|
| Extract-Min | $O(\log n)$ |
| Insert | $O(1)$ |
| Decrease-Key | $O(1)$ |
| Meld | $O(1)$ |

Best possible, so lets invent something else that can't be better and we have no clue how fast they run.

# Prehistory: Fibonacci Heaps

| | Fib-Heap |
|---|---|
| Extract-Min | $O(\log n)$ |
| Insert | $O(1)$ |
| Decrease-Key | $O(1)$ |
| Meld | $O(1)$ |

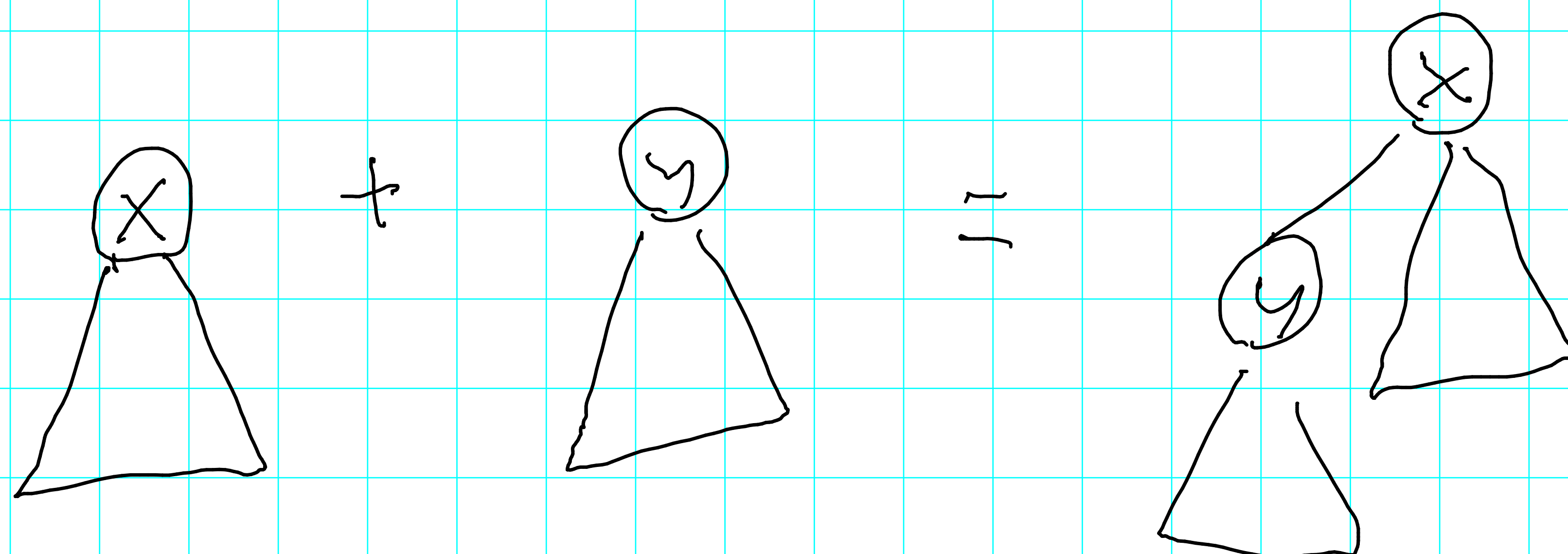
Best possible, so lets invent something else that can't be better and we have no clue how fast they run.

Fib Heaps are
- complicated
- have $\log\log n$ balance bits per node

# Pairing Heaps

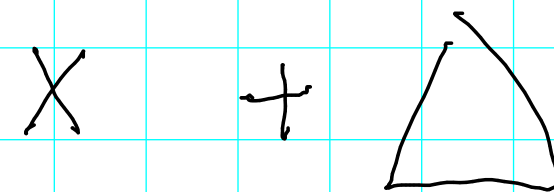
[Fredman
Sedgewick
Sleator
Tarjan
86]

Primitive: Pair

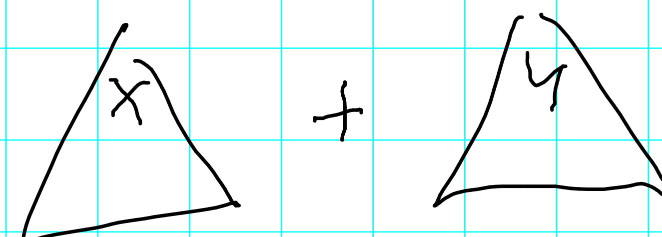


# Pairing Heap

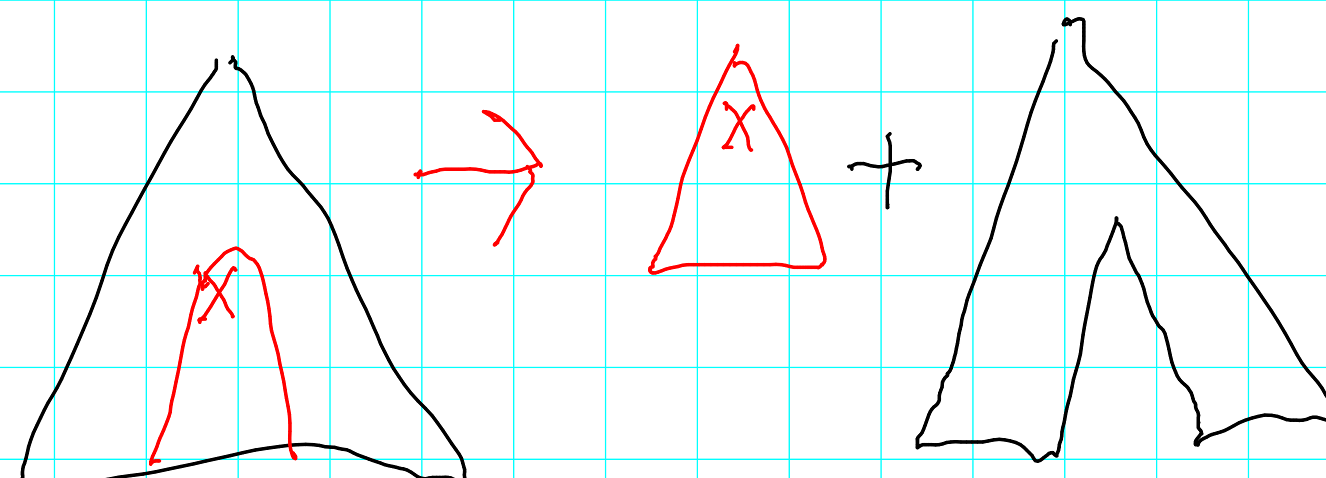
Insert(x)



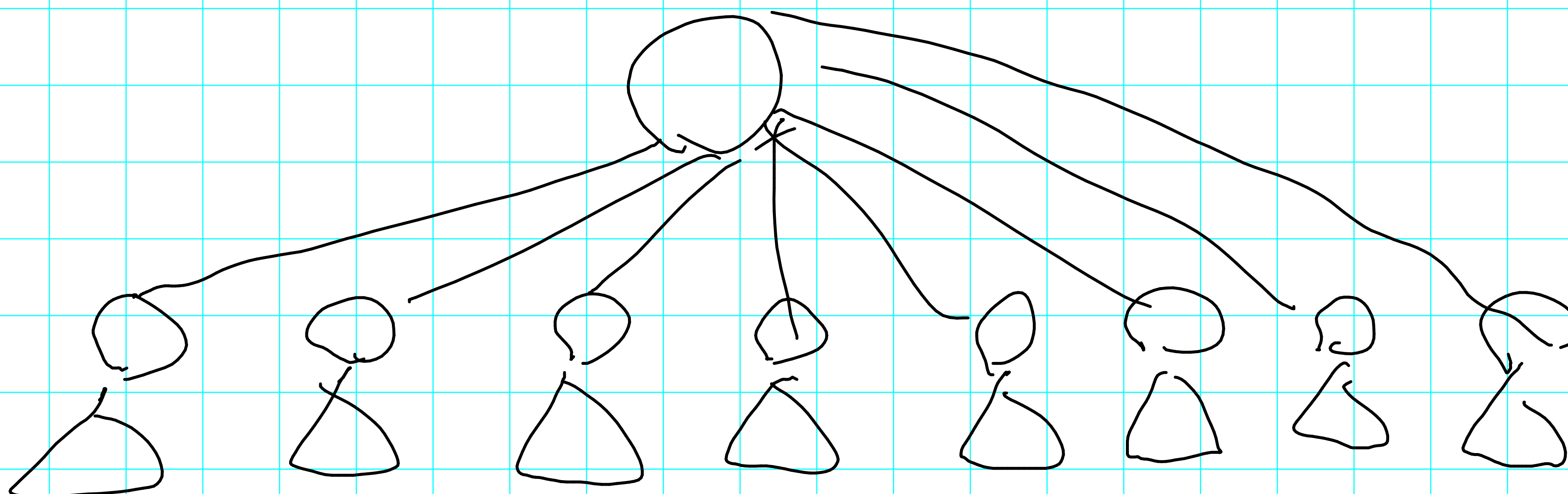

Meld(x,y)



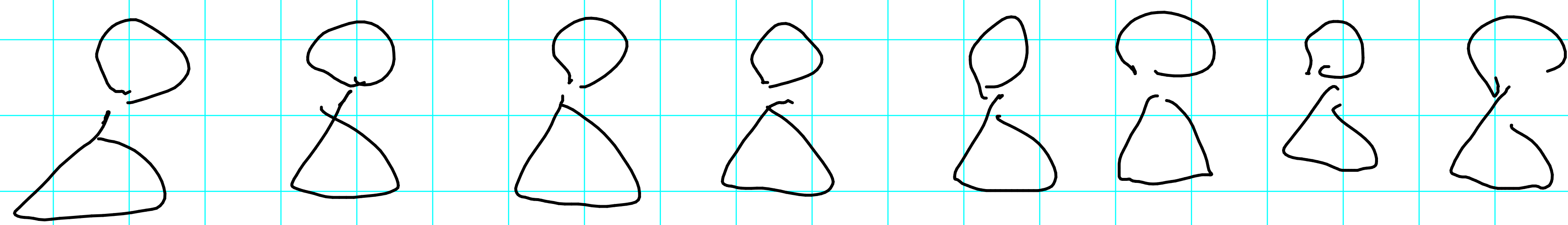

Decrease-Key(x)



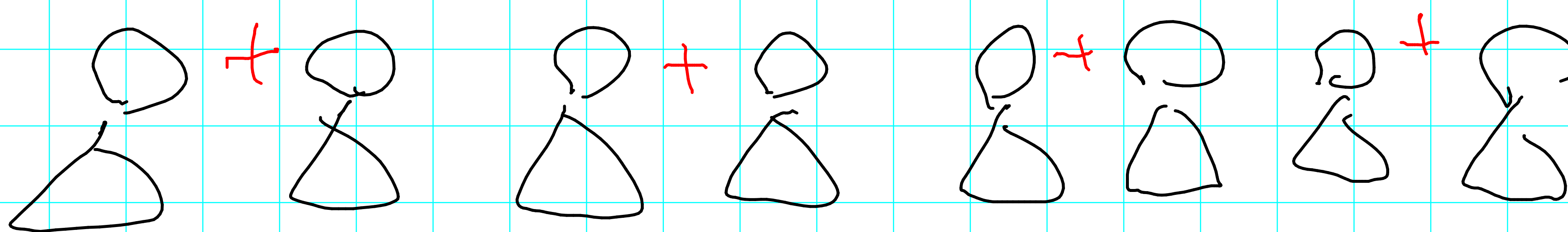
# Pairing Heap - Extract-Min



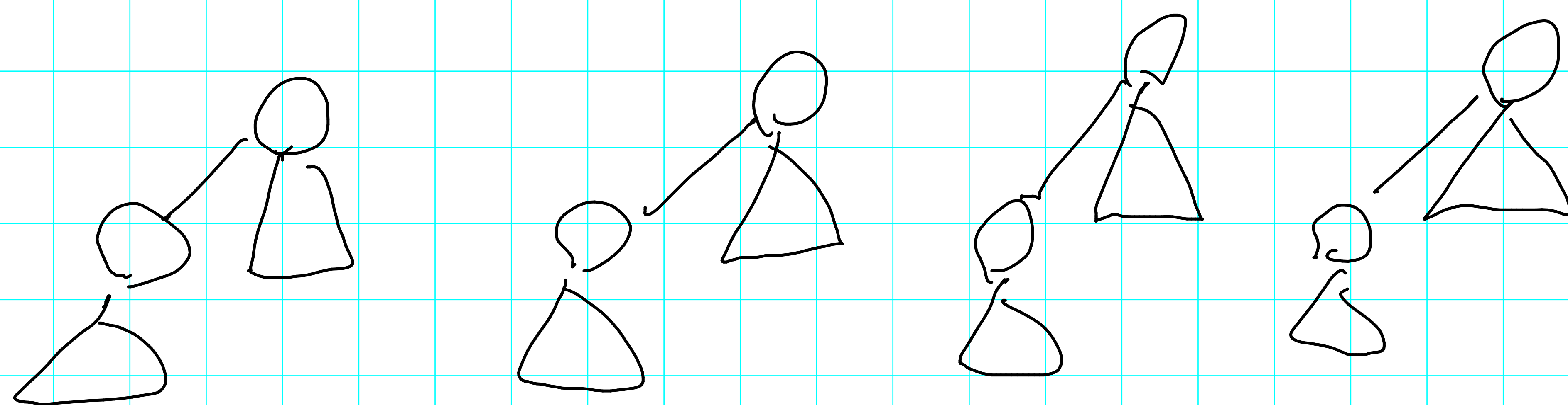
# Pairing Heap - Extract-Min



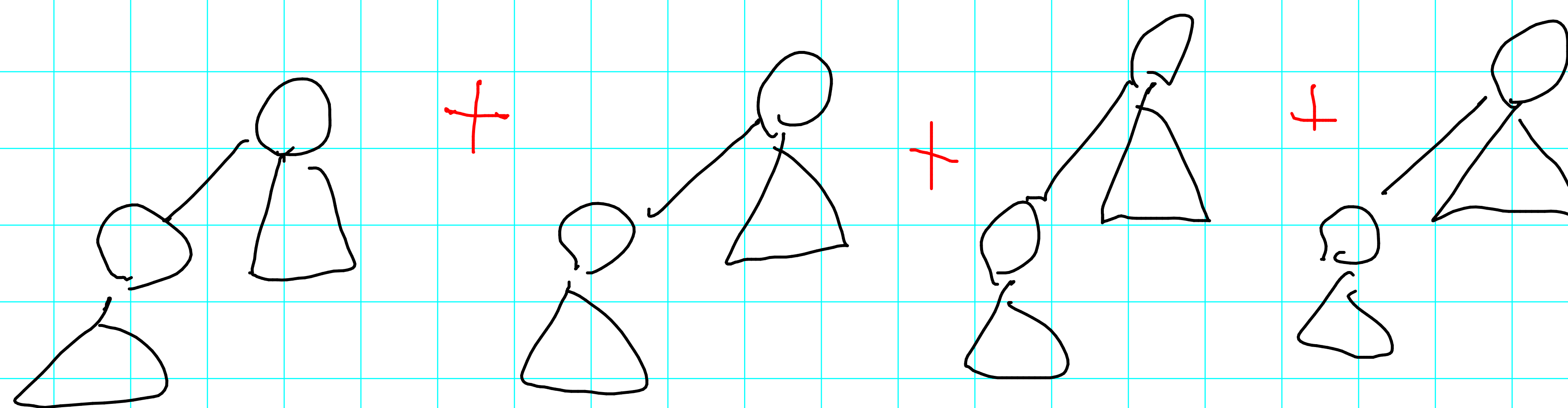
# Pairing Heap - Extract-Min



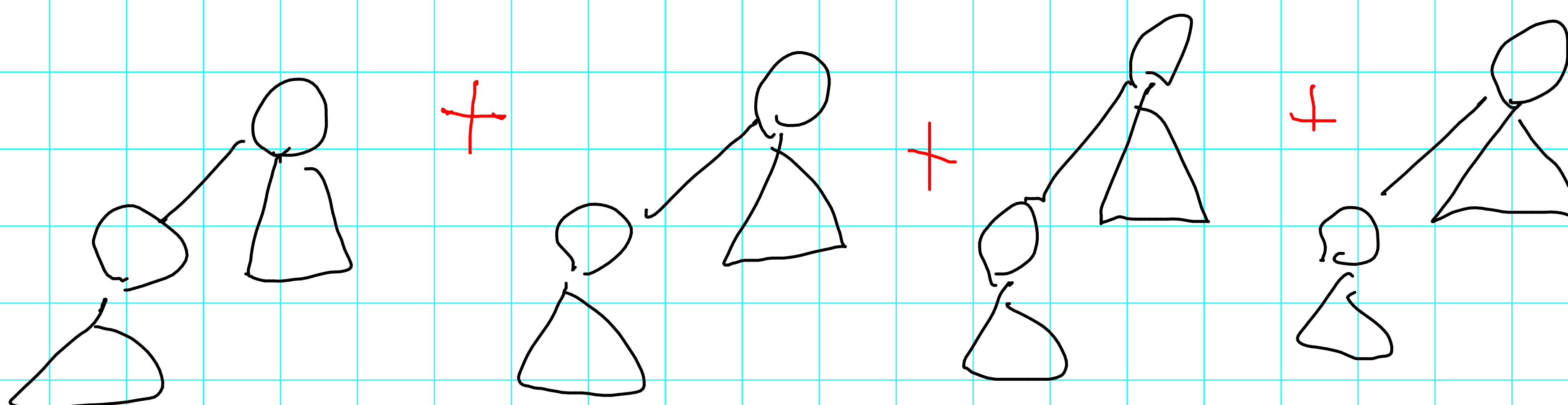
# Pairing Heap - Extract-Min



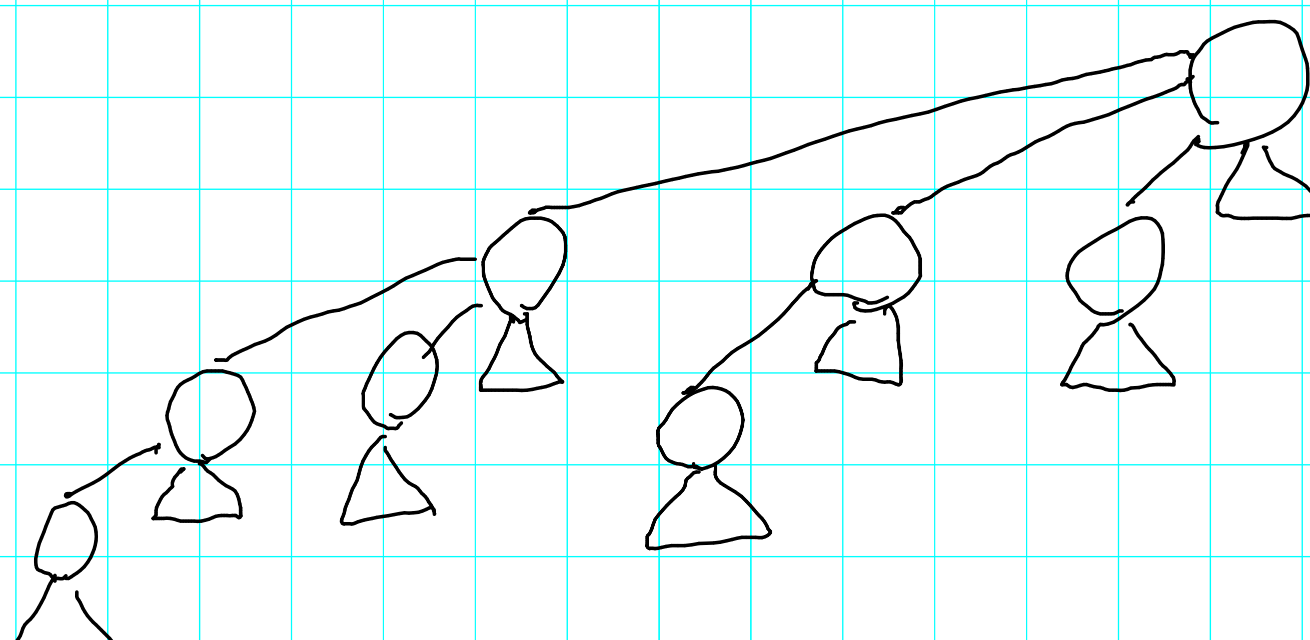
# Pairing Heap - Extract-Min



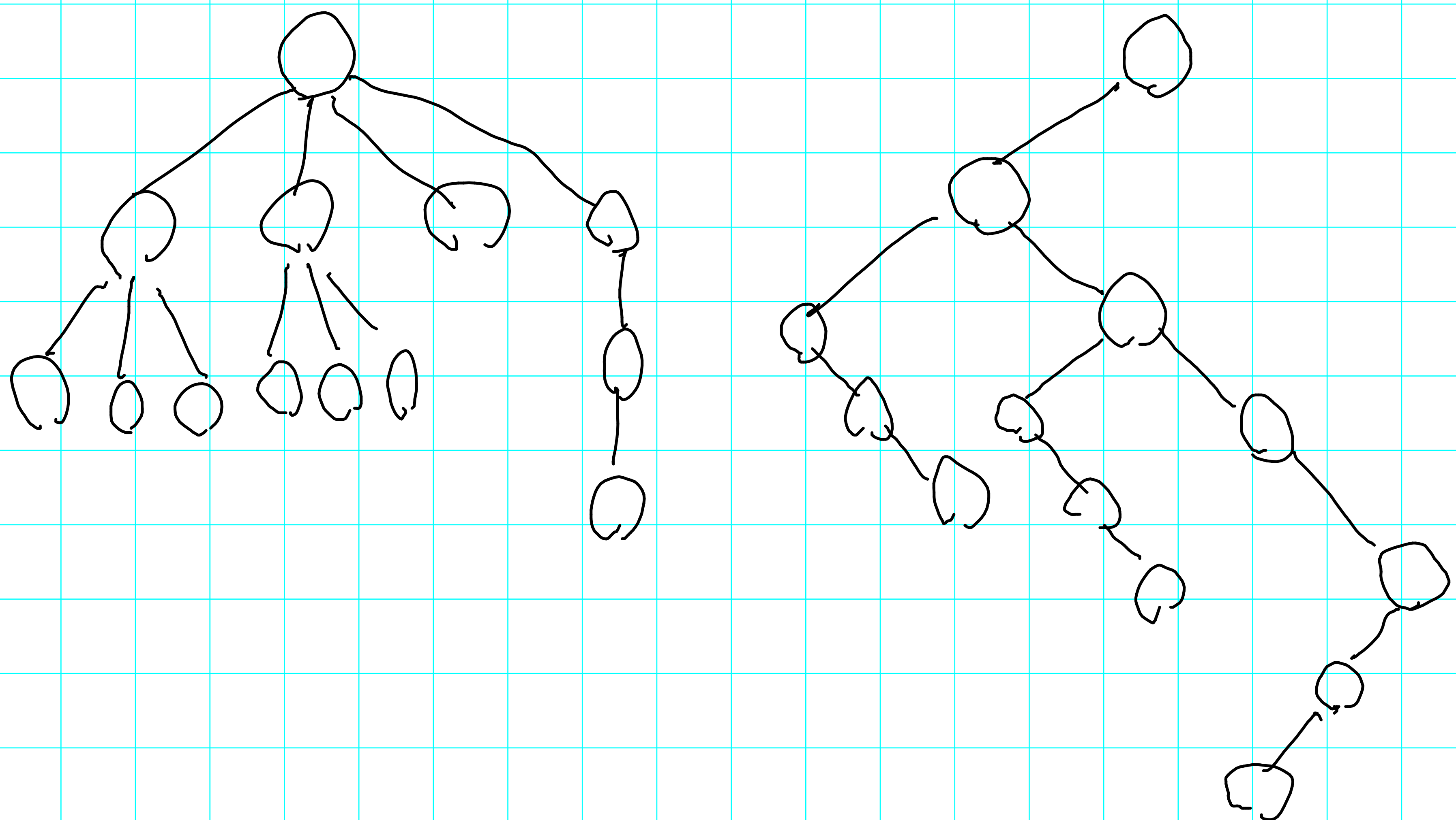
# Pairing Heap - Extract-Min



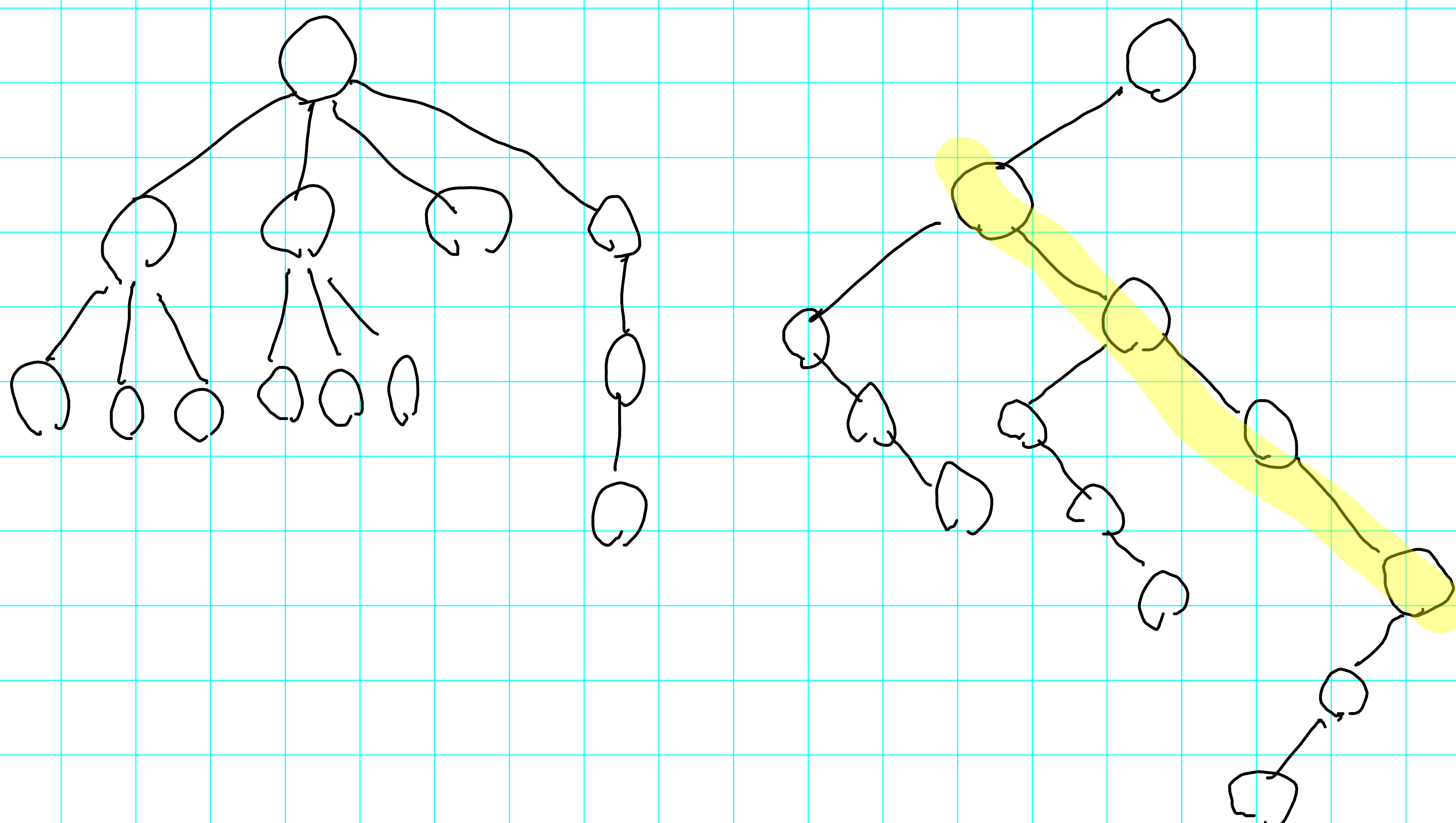
e.g.



Tilt Your Head and it Looks Like a Splay



Tilt Your Head and it Looks Like a Splay



Runtime ???

| | Fib | Pair |
|---|---|---|
| Extract-Min | $O(\log n)$ | $O(\log n)$ |
| Insert | $O(1)$ | $O(\log n)$ |
| Decrease-Key | $O(1)$ | $O(\log n)$ |
| Meld | $O$ | $O(\log n)$ |

| | Fib | Pair | Pair [IOO] |
|---|---|---|---|
| Extract-Min | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| Insert | $O(1)$ | $O(\log n)$ | $O(1)$ |
| Decrease-Key | $O(1)$ | $O(\log n)$ | $O(\log n)$ |
| Meld | O | $O(\log n)$ | O |

| | Fib | Pair | Pair [IO0] | Pair [Pettie 05] |
|---|---|---|---|---|
| Extract-Min | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| Insert | $O(1)$ | $O(\log n)$ | $O(1)$ | $O(2^{2\sqrt{\log \log n}})$ |
| Decrease-Key | $O(1)$ | $O(\log n)$ | $O(\log n)$ | $O(2^{2\sqrt{\log \log n}})$ |
| Meld | $O$ | $O(\log n)$ | $O$ | $O(2^{2\sqrt{\log \log n}})$ |

+ "Emperical Evidence" of $O(1)$ D-K

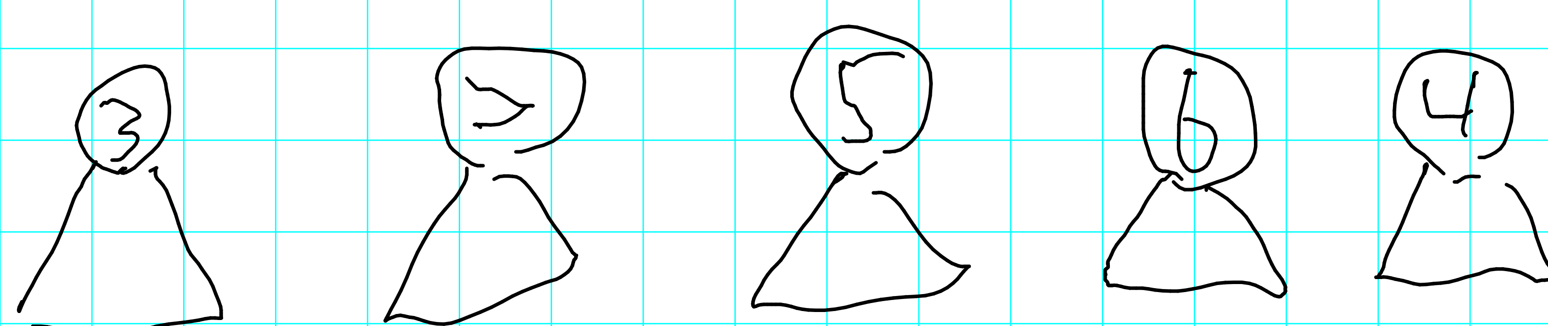
[Stasko, Vitter 87]

# Lower Bounds [Fredman 99]

Model of computation:

Generalized from pairing and Fib heaps

Forest of heaps. Combine based on comparisons on the roots and looking at any additional bits attached to the nodes



# Lower Bounds

Result:

If Insert is $O(\log n)$

and Extract-min is $O(\log n)$

and Decrease-key is $c \log n$, $c < 1$

then the number of augmented

bits is $> \log \log n$

# Lower Bounds

Result:

If Insert is $O(\log n)$

and Extract-min is $O(\log n)$

and Decrease-key is $c \log n$, $c<1$

then the number of augmented

bits is $> \log \log n$

Pairing heaps have 0 augmented bits

so Decrease-key is $> \log n$

# Open Problems

- What is the real runtime of pairing heaps?
- Is there some self-adjusting structure with the same runtime as Fib heaps. Need to step outside of the model

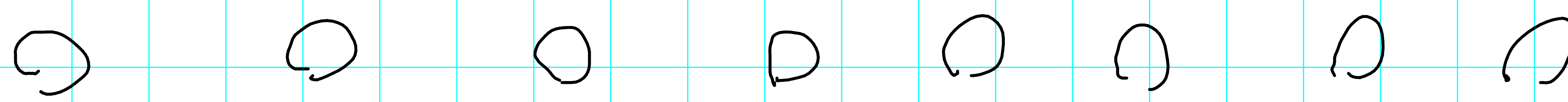
# Beyond The Worst Case

- Pairing heaps have a working-set property [IOa]

- Conj: Excluding Decrease-Key Dynamic Optimality = Working Set in the Heap model

- Queaps ~ not in the model [I, Langerman 05]

# Variants



How to combine the children of the old root?

Almost anything works except one incremental pass

[My qualifying exam question from Fredman]

# Variants

$$(O + O) + (O + O) + (O + O) + (O + O)$$

# Variants

$$(\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)$$

$$((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)) + ((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc))$$

# Variants

$$(\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)$$

$$((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)) + ((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc))$$

$$\bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc \quad \bigcirc$$

# Variants

$$(\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)$$

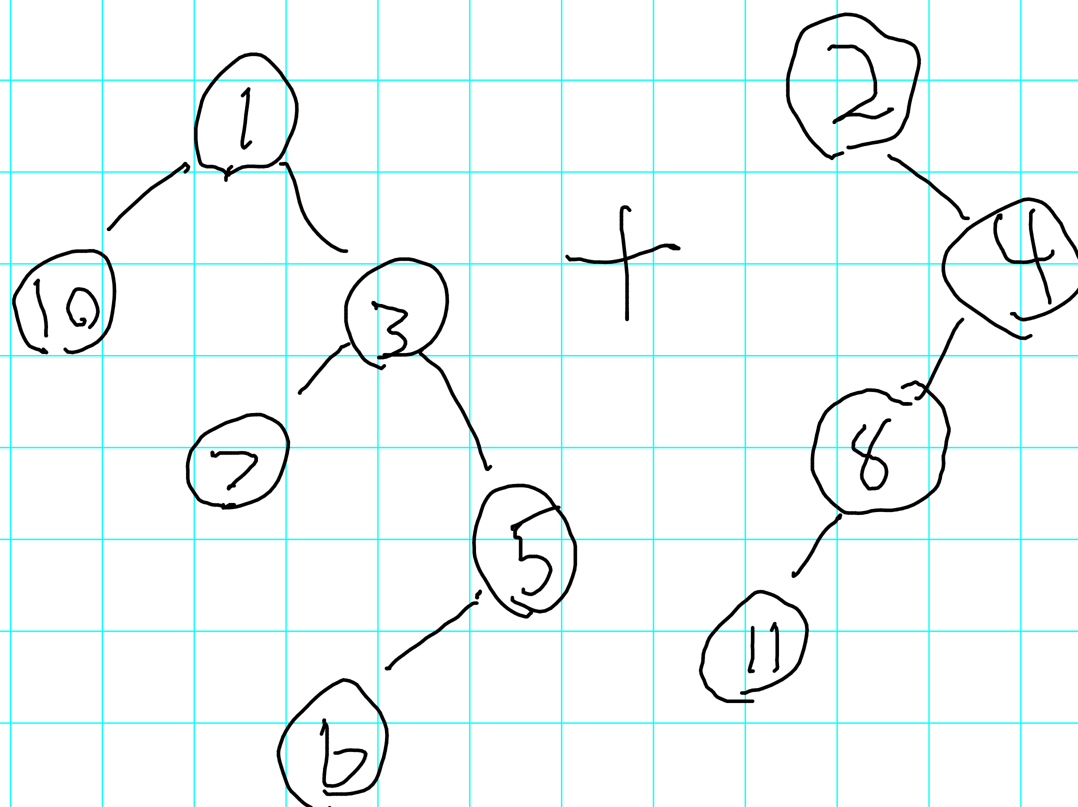
$$((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc)) + ((\bigcirc + \bigcirc) + (\bigcirc + \bigcirc))$$

$$(\bigcirc + \bigcirc + \bigcirc + \bigcirc) +$$
$$(\bigcirc + \bigcirc + \bigcirc + \bigcirc)$$

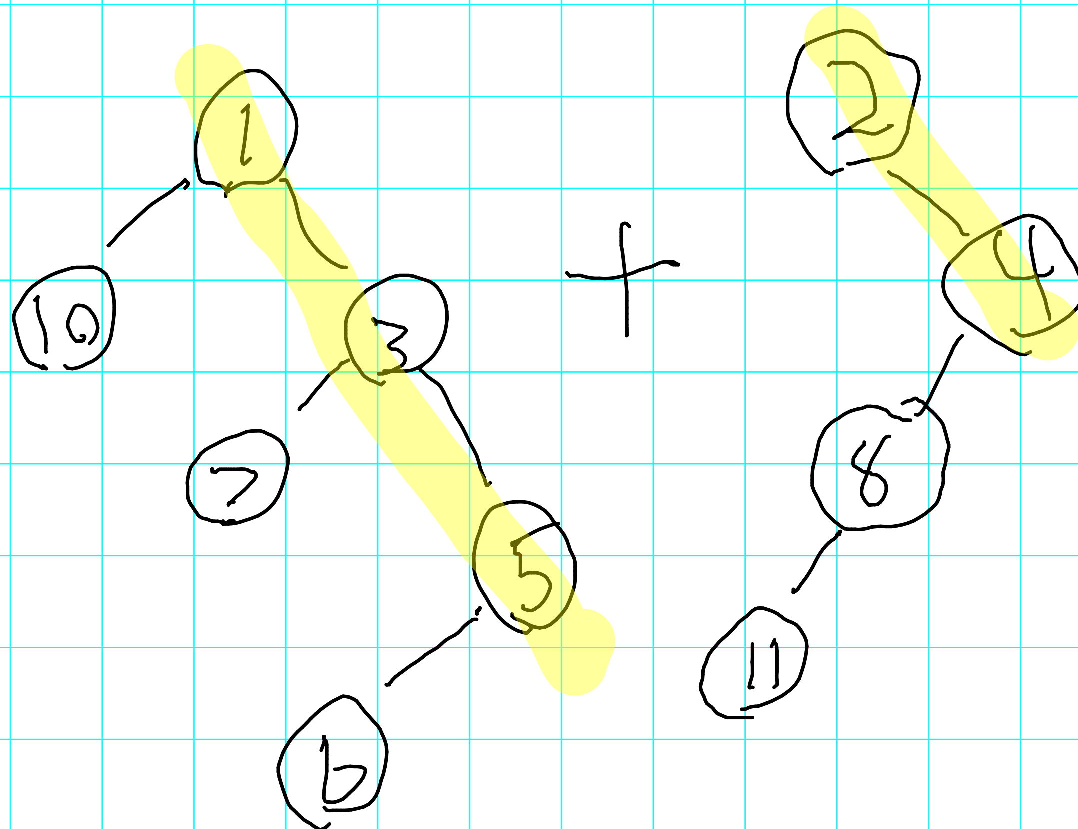
# Skew Heaps [Sleator + Tarjan 86]

- Another self adjusting heap
- Binary
- Great for teaching amortized analysis.

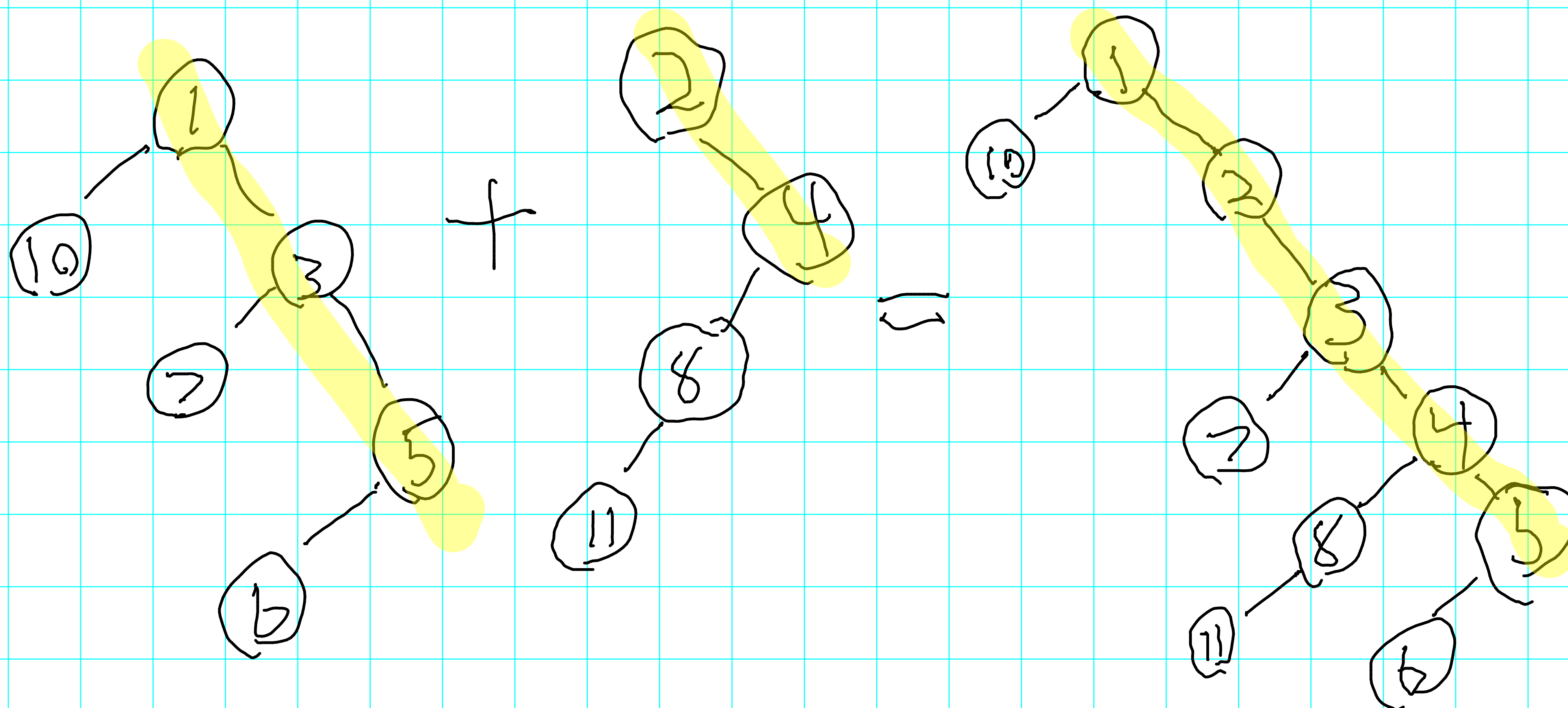
# Skew Heaps - Meld



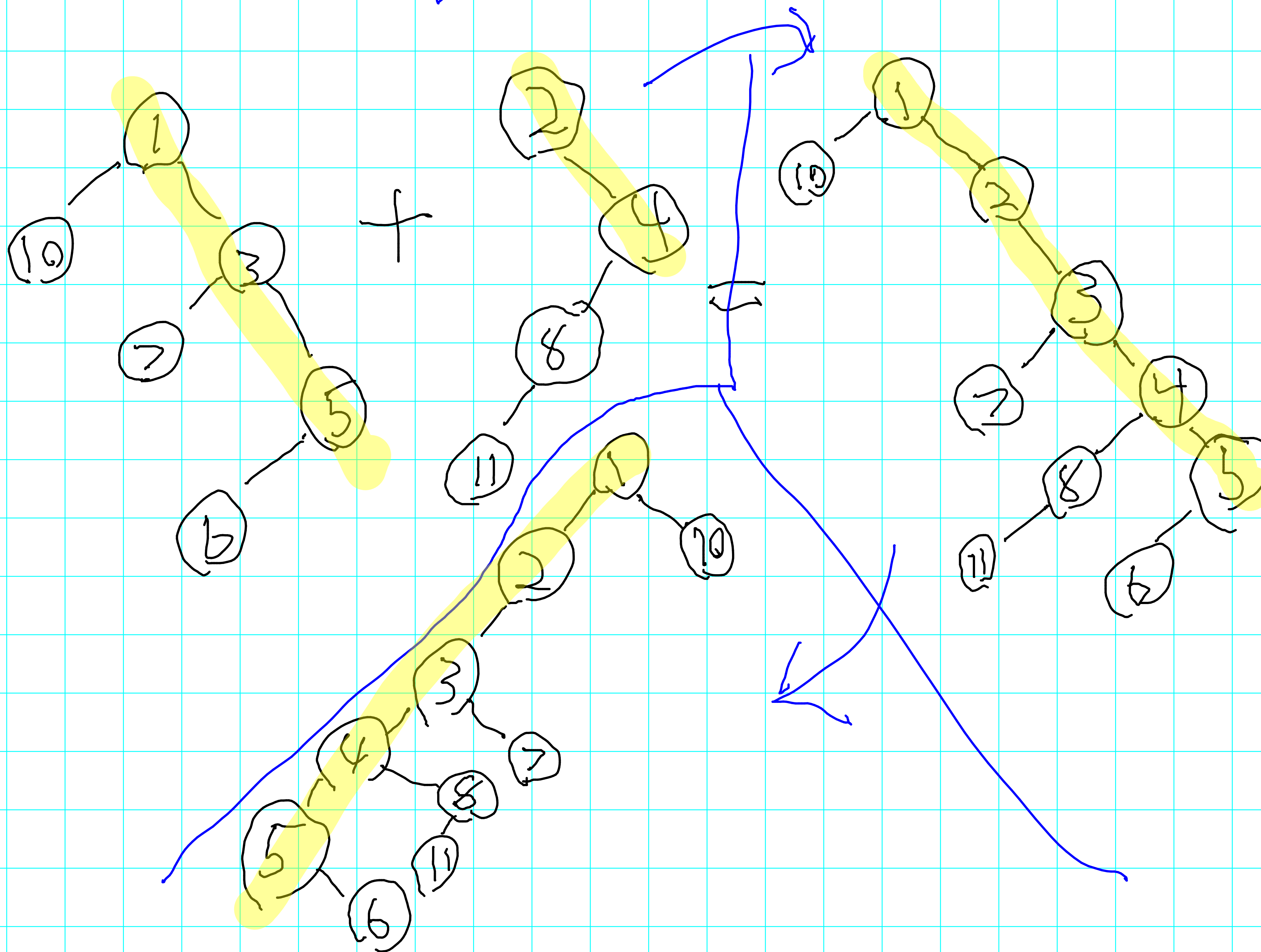
# Skew Heaps – Meld



# Skew Heaps - Meld



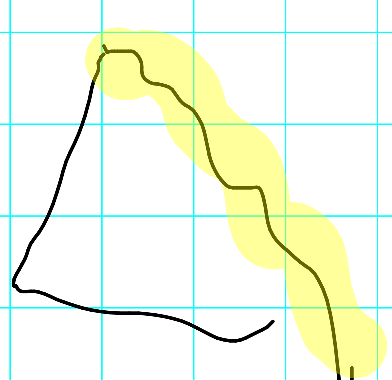
# Skew Heaps - Meld



# Analysis

Ins, Extract-min, meld
are $O(\log n)$

Put a coin on any node
that has a larger right
subtree than left



$\therefore$ All but $\log n$ on a right
path have coins, Fliering to
a left path increases coins by
at most $\log n$ — pathlength

# Analysis

- Not much work has been done on skew heaps.

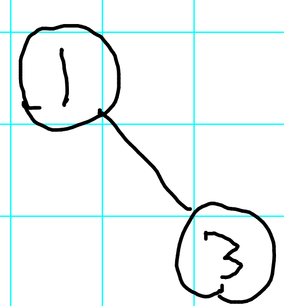
# Analysis

- Not much work has been done on skew heaps.

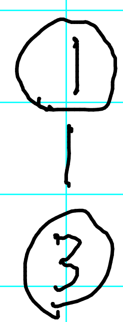
- Skew heaps are not so good.....

# My lazy friend

Skew Heap

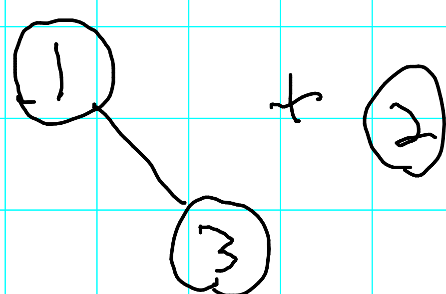



Lazy friend

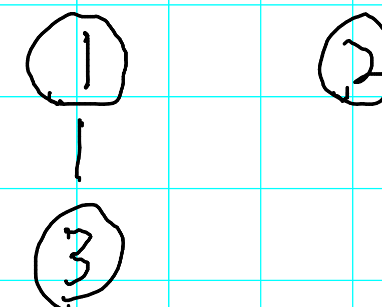


# My lazy friend

Skew Heap



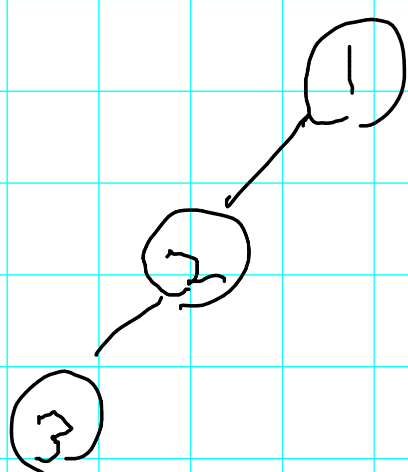
Lazy friend



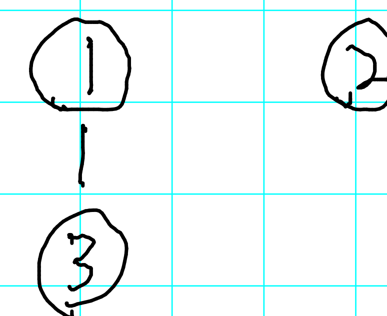
Insert 2

# My lazy friend

Skew Heap



Lazy friend

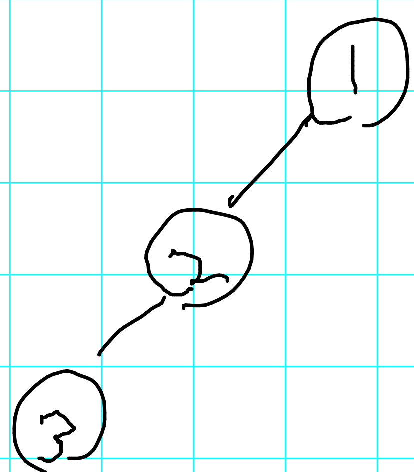


1:2 1:3

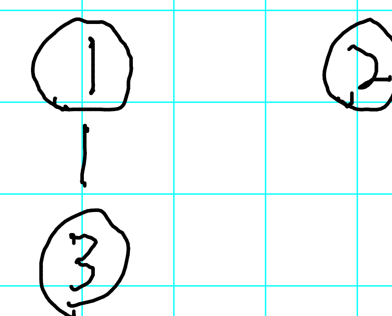
Insert 2

# My lazy friend

Skew Heap




Lazy friend

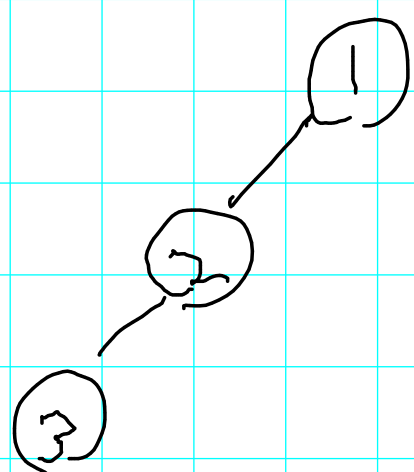



1:2 1:3

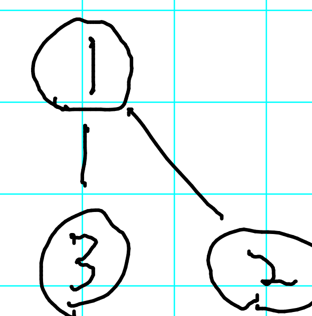
Find-min

# My lazy friend

Skew Heap




Lazy friend




~~1:2~~ 1:3

Find-min

# Lazy Friend

- A heap gives its lazy friend all of the comparisons it does
- The lazy friend, when it has a forest and needs one tree, does the comparisons in the list among its root

# Lazy Friend

The number of comparisons performed by a lazy friend is at most the number of the original heap

(total time may be different as how did our lazy friend find the right comparison)

# Skew Heap ≤ Pairing Heap

- The lazy friend of the skew heap is the odd-even pairing heap [Fredman 99]
- 

# Skew Heap ≤ Pairing Heap

- The lazy friend of the skew heap is the odd-even pairing heap

- On any sequence of operations, the skew heap is the same or slower than the pairing heap

The End