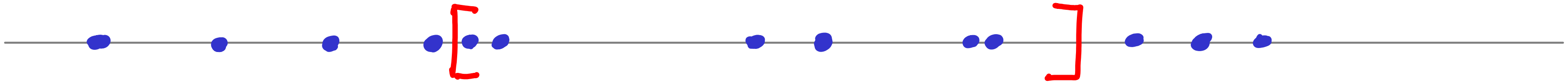
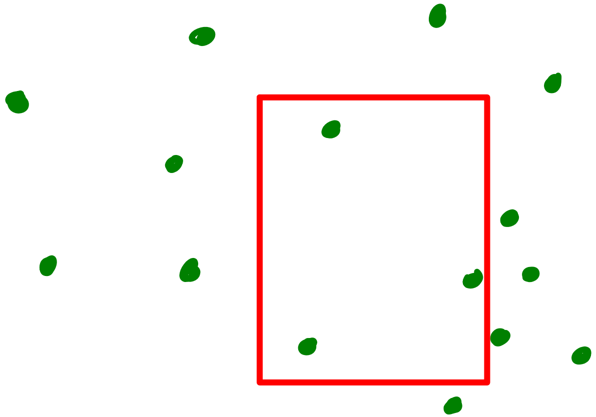
RANGE COUNTING

COUNT (or ENUMERATE) OBJECTS IN A GIVEN RANGE
(many times)

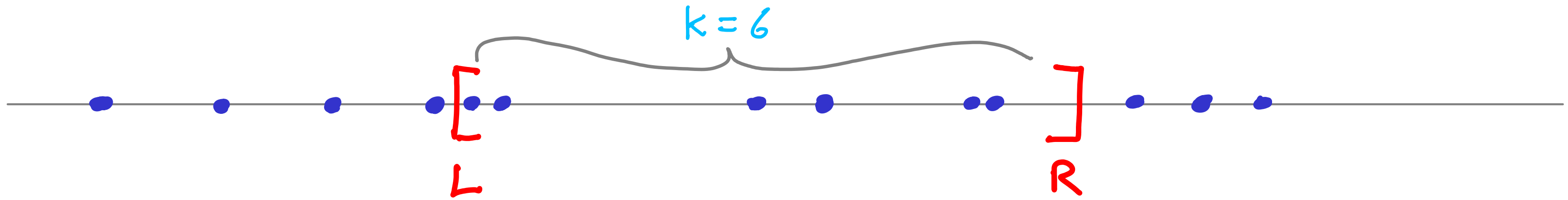
1D :



2D :



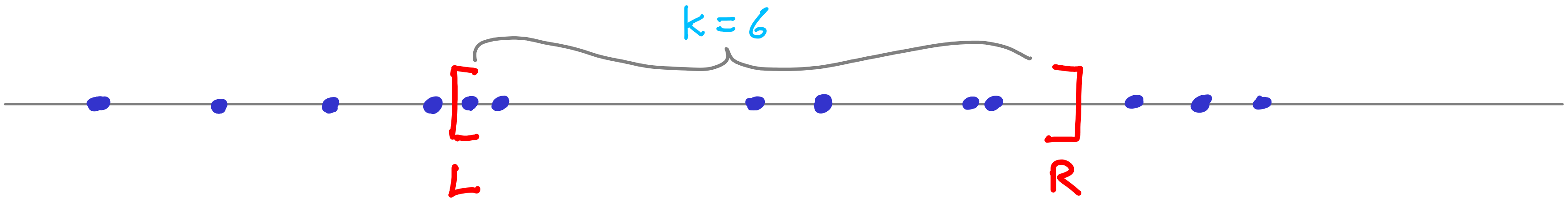
1D :



USE ARRAY: $O(\log n)$ to place L, R → to count.

$O(k + \log n)$ to enumerate/report.

1D :



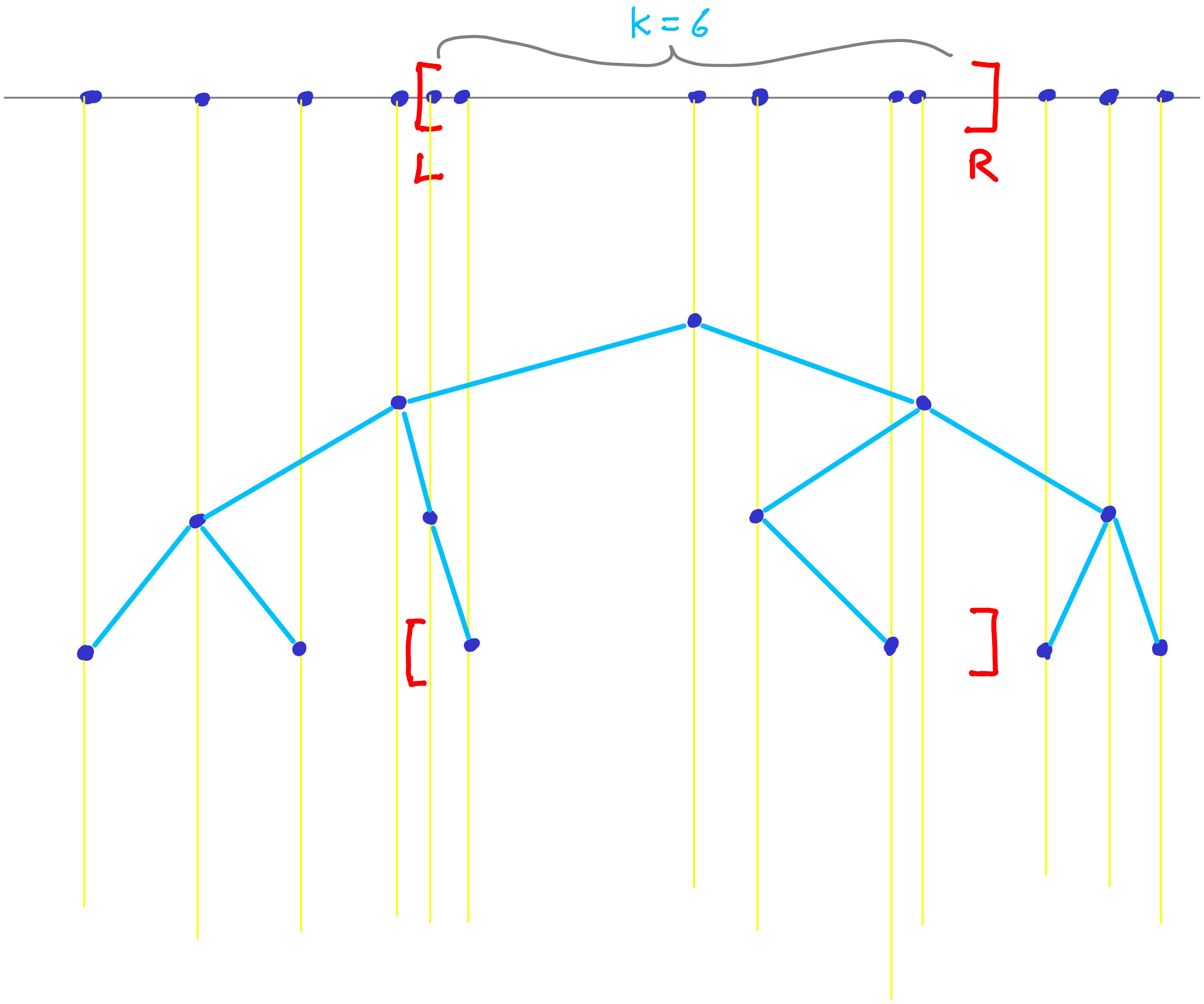
USE ARRAY: $O(\log n)$ to place L, R → to count.

$O(k + \log n)$ to enumerate/report.

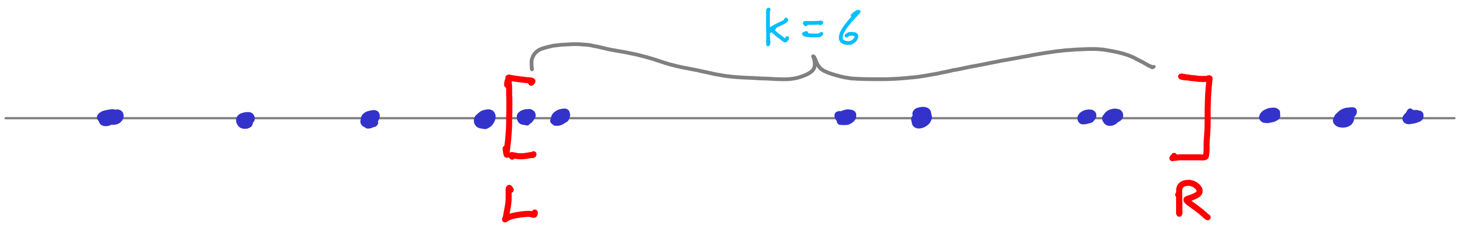
2 problems:

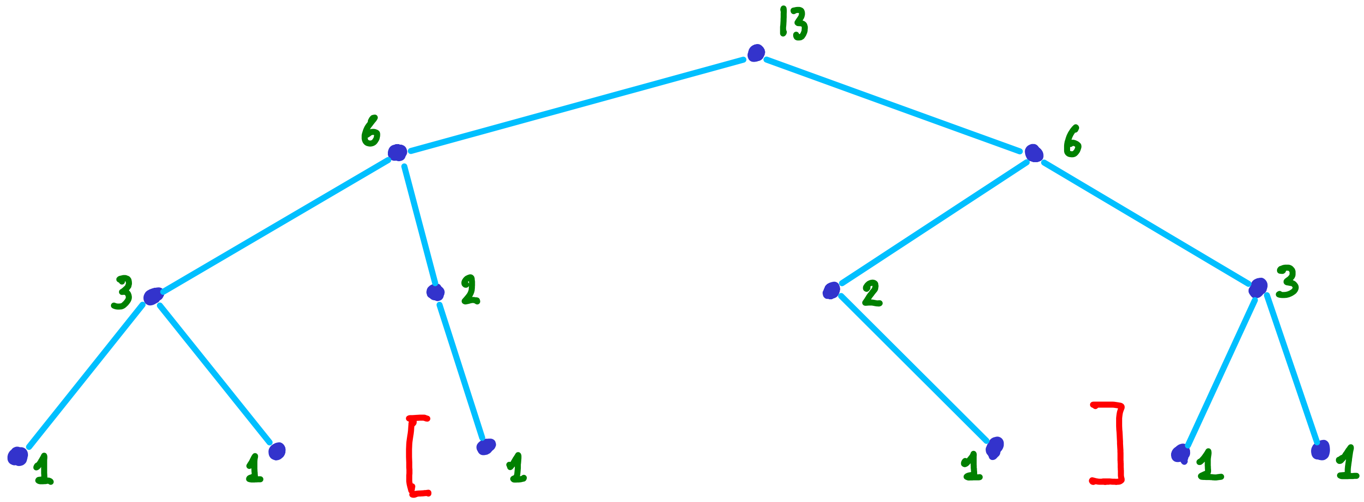
- doesn't generalize to 2D (no array)
- not dynamic ... insert, delete data : $O(n)$

1D :



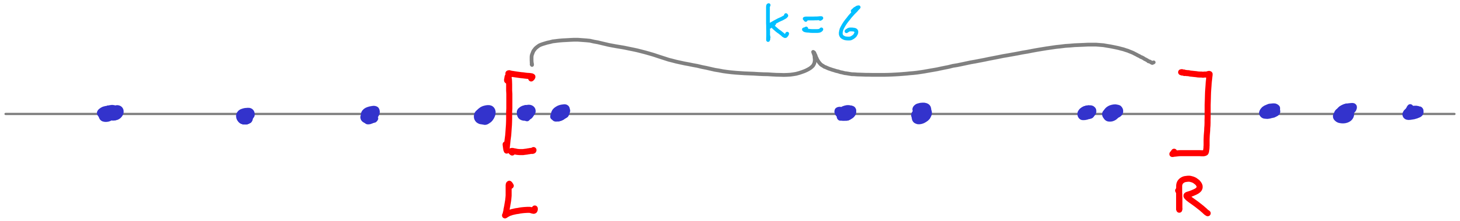
1D :

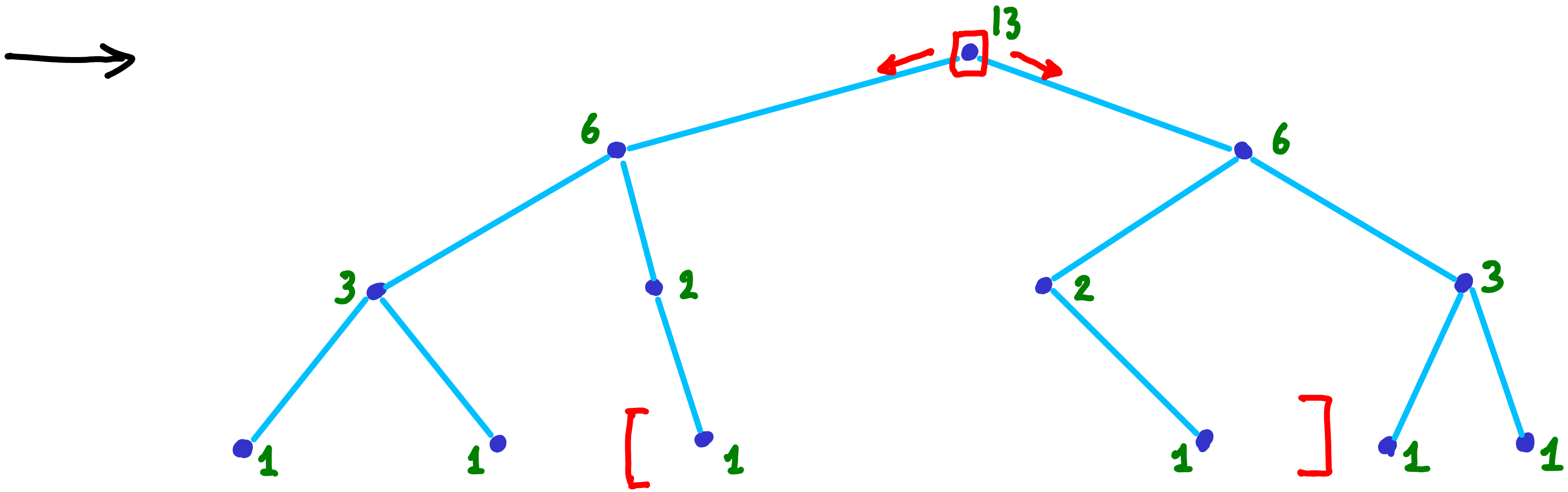




store size of subtree in each node

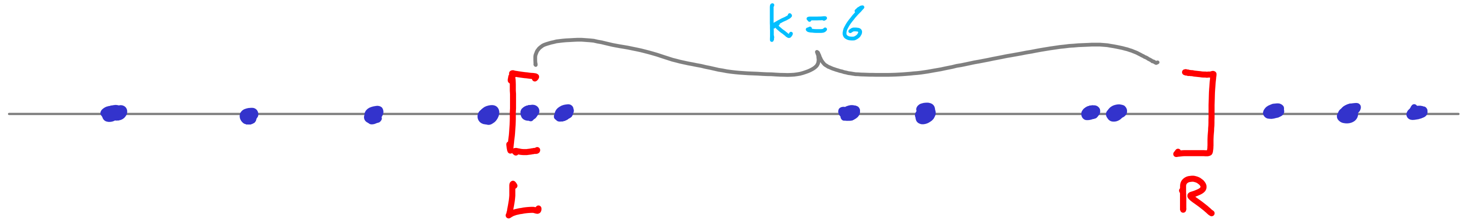
1D :

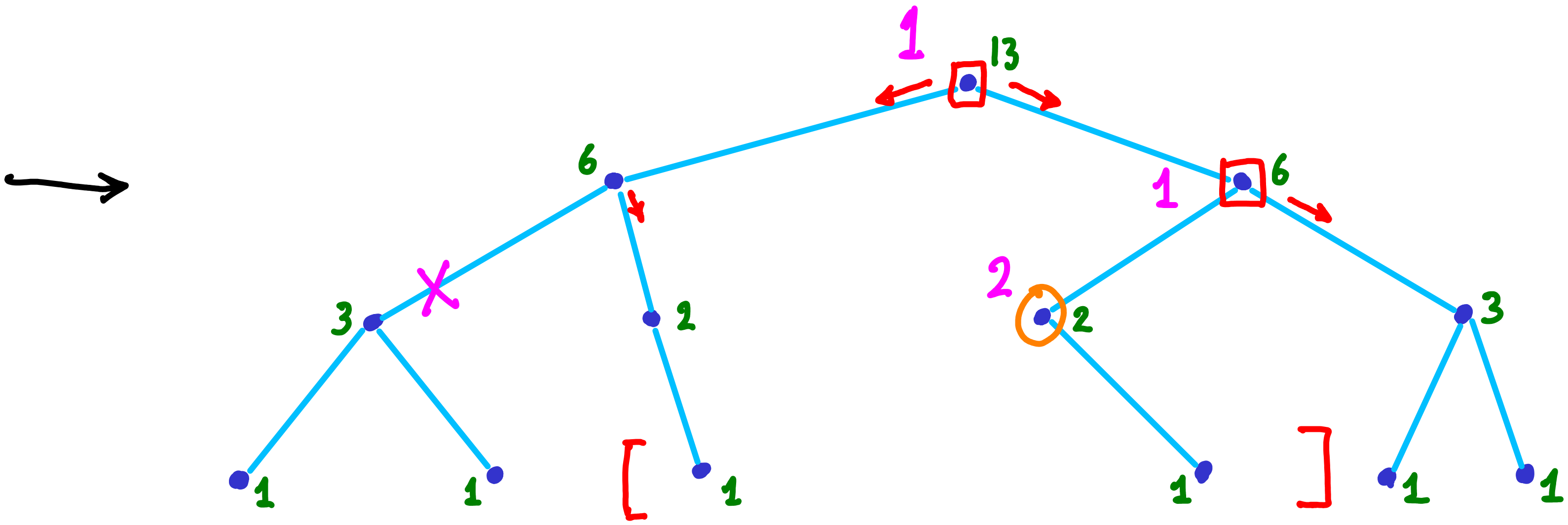




□ → count 1

1D :

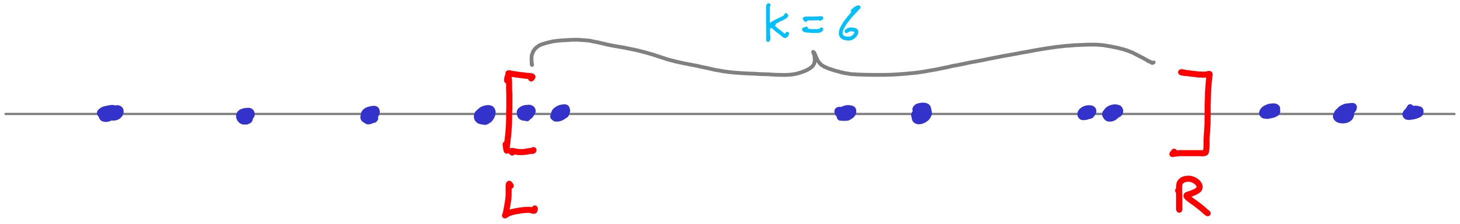


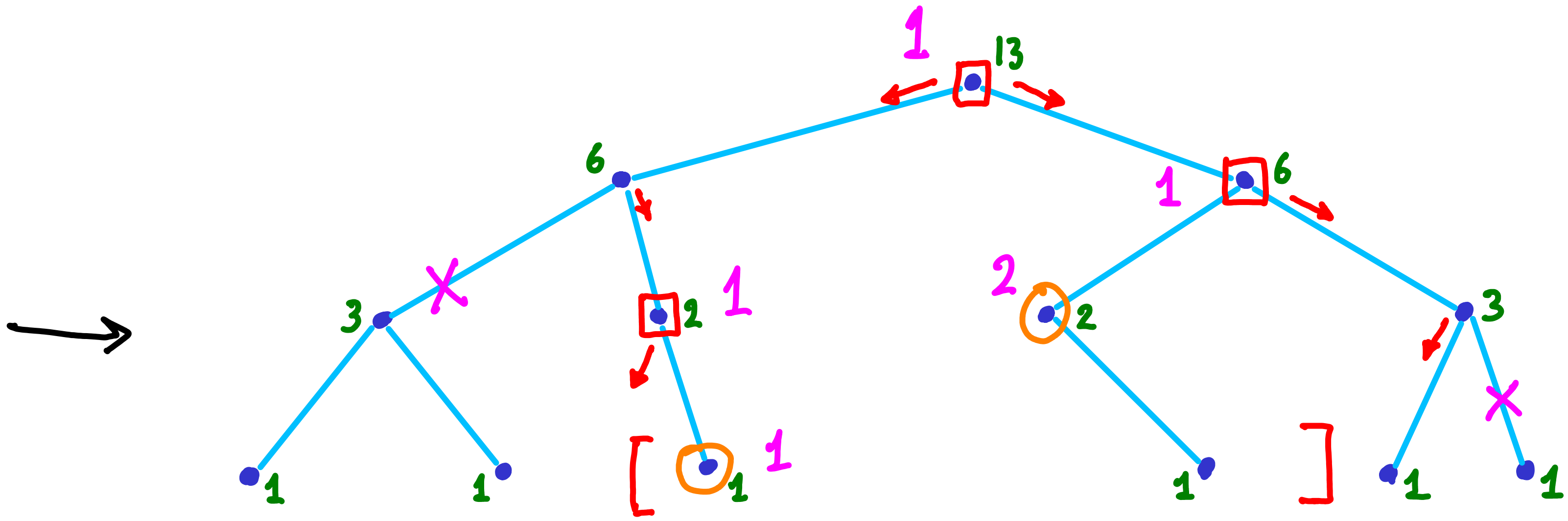


→ count 1

→ count subtree

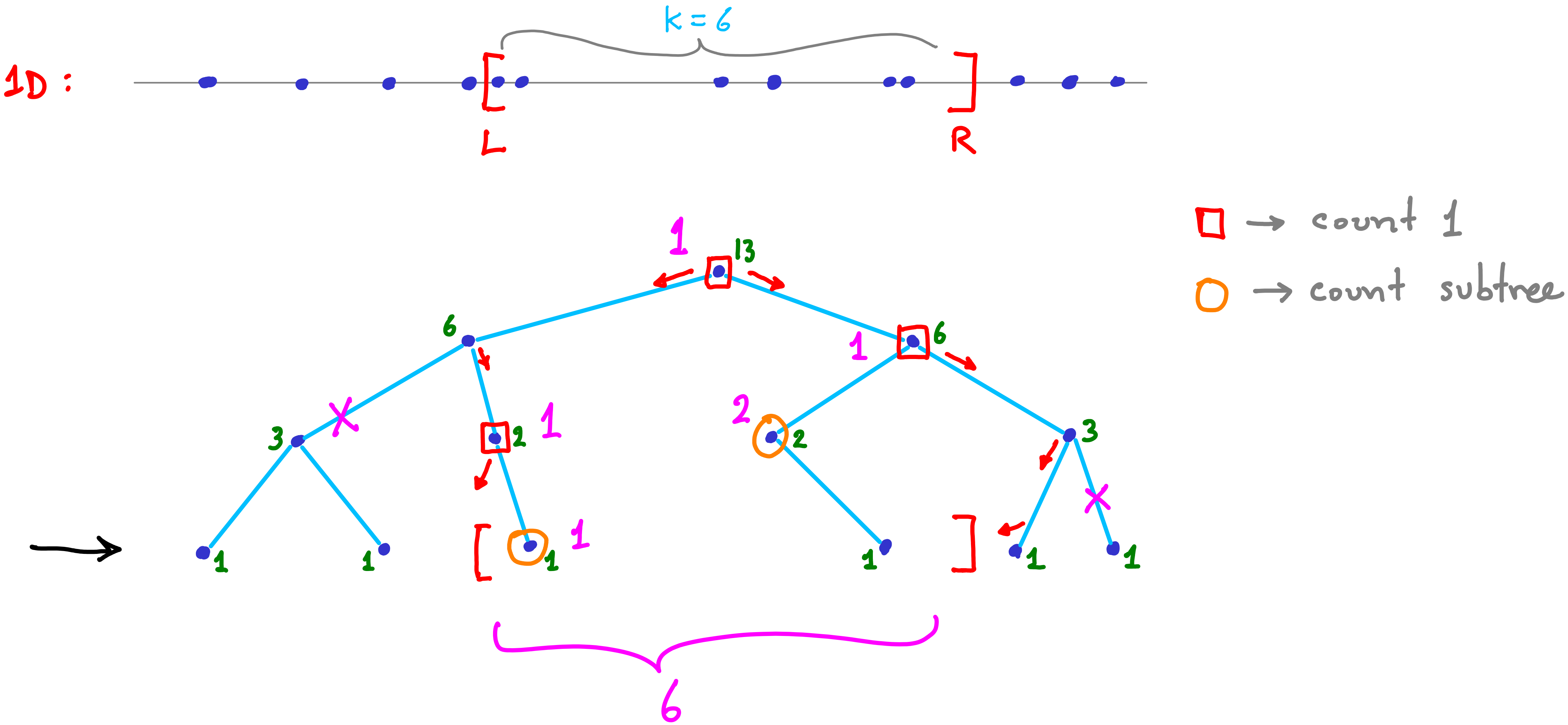
1D :

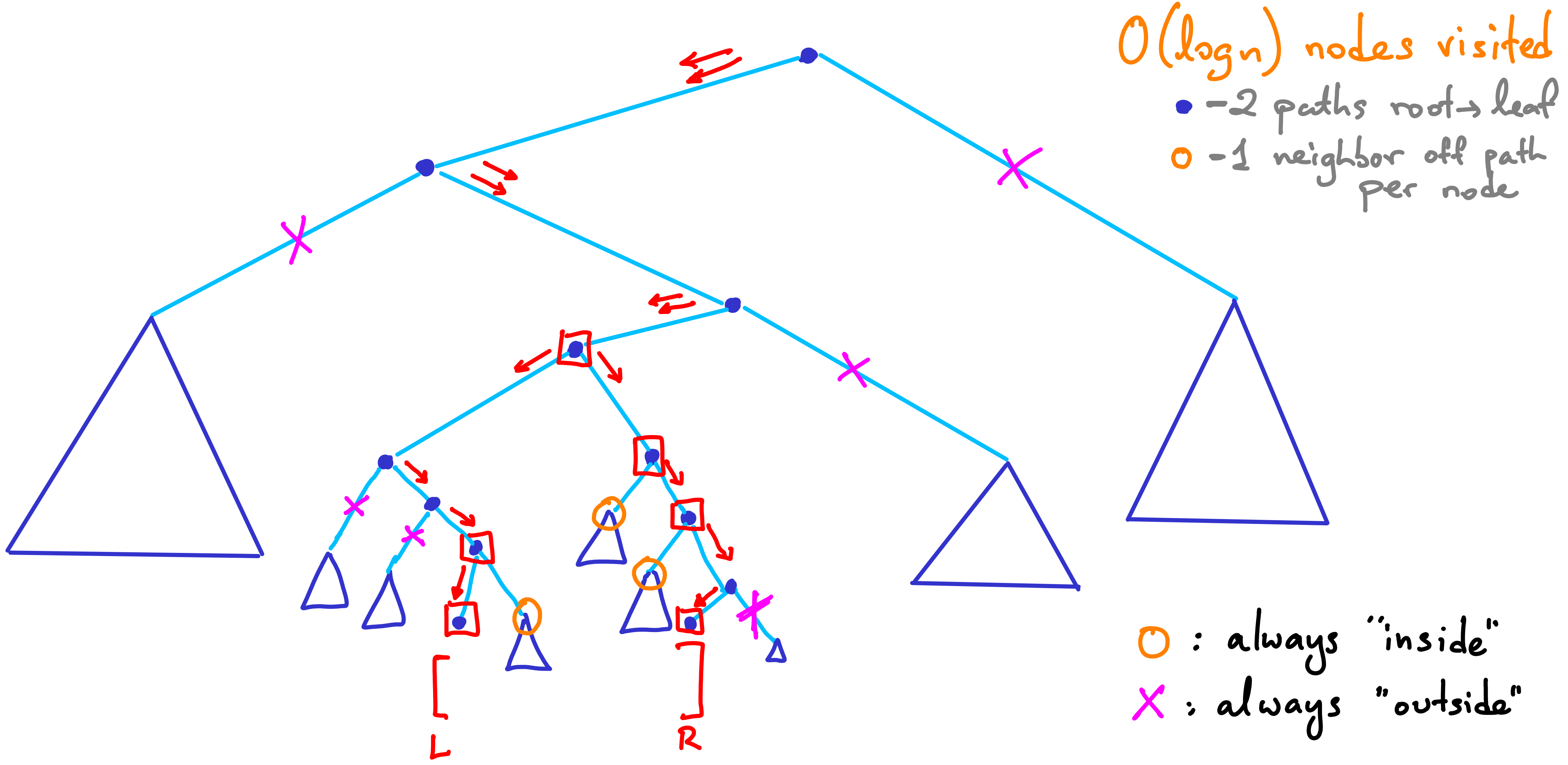




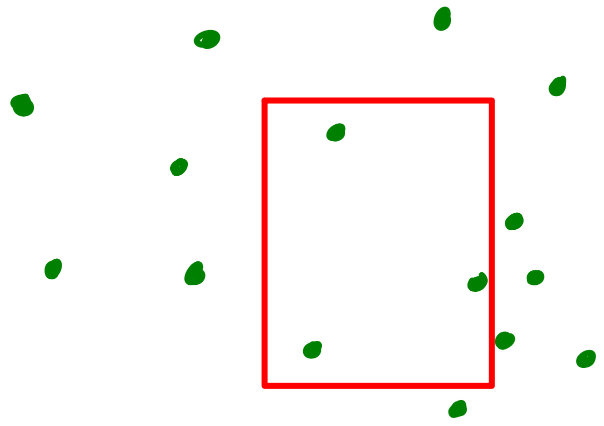
□ → count 1

○ → count subtree

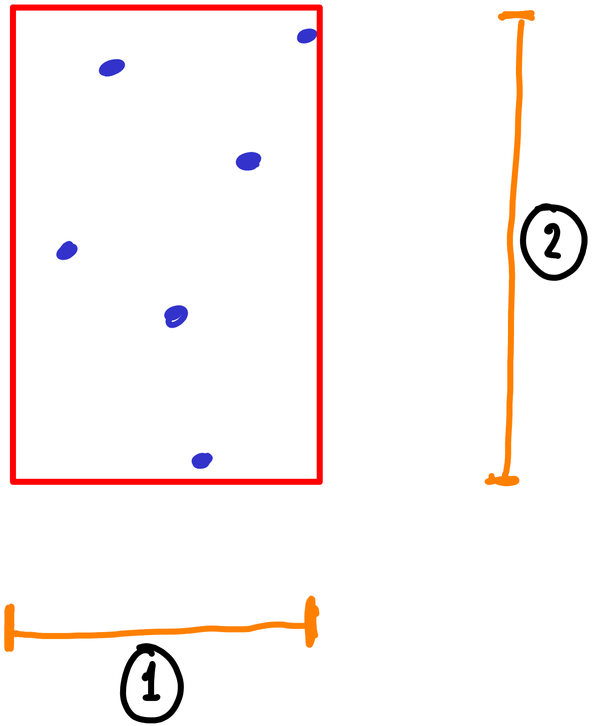

1D :
k = 6
L
R
1
13
6
6
1
3
2
1
2
2
3
1
1
1
1
1
1
1
6
□ → count 1
○ → count subtree


O(log n) nodes visited
• - 2 paths root → leaf
o - 1 neighbor off path per node
L
R
o : always "inside"
× : always "outside"

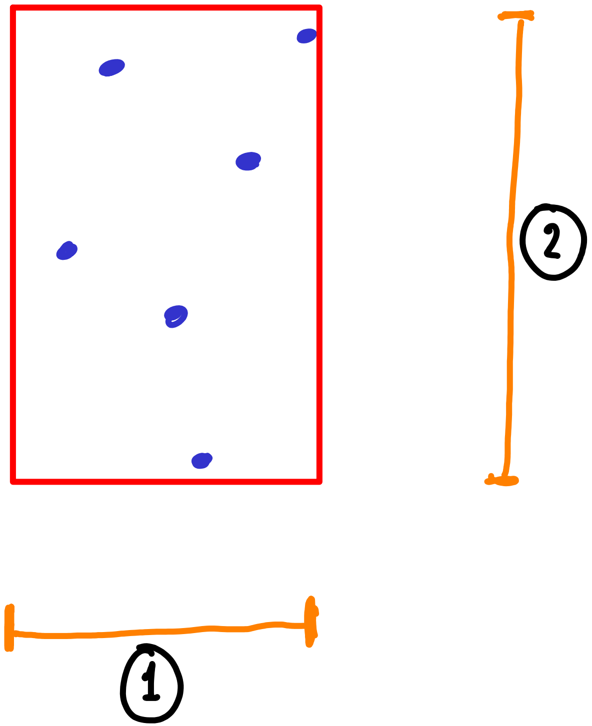
2D :



Intuitive idea for 2D range counting : search X, then Y



Intuitive idea for 2D range counting : search X, then Y

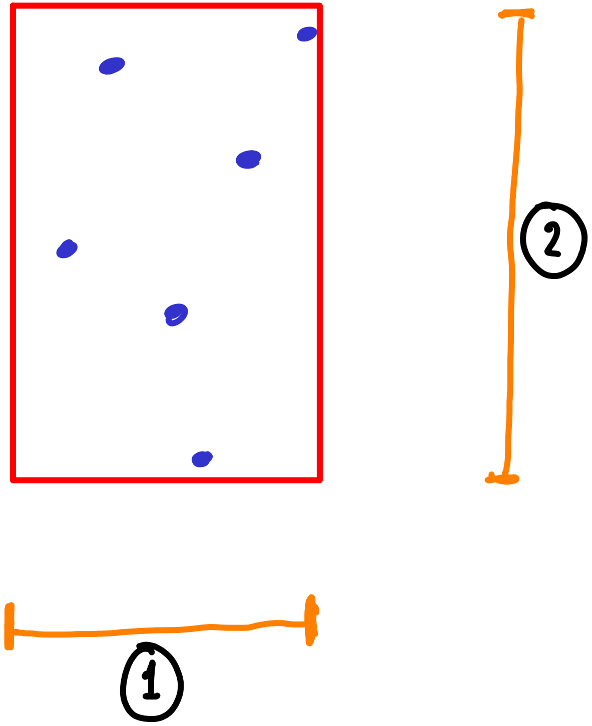


Expensive to Y-sort all X-ranges

$O(\log n) \cdot O(n^2)$

by dealing with ranges in careful order

Intuitive idea for 2D range counting : search X, then Y



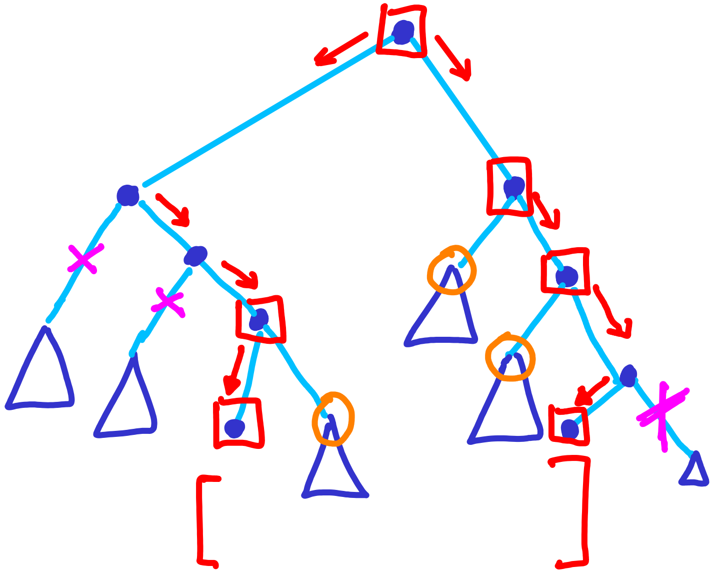
Expensive to Y-sort all X-ranges

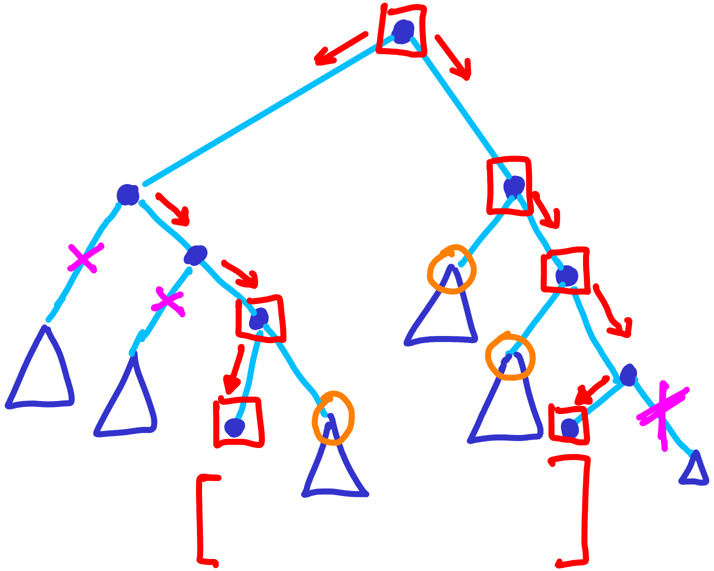
$O(\log n) \cdot O(n^2)$

by dealing with ranges in careful order

Also expensive to store all X-ranges by brute-force

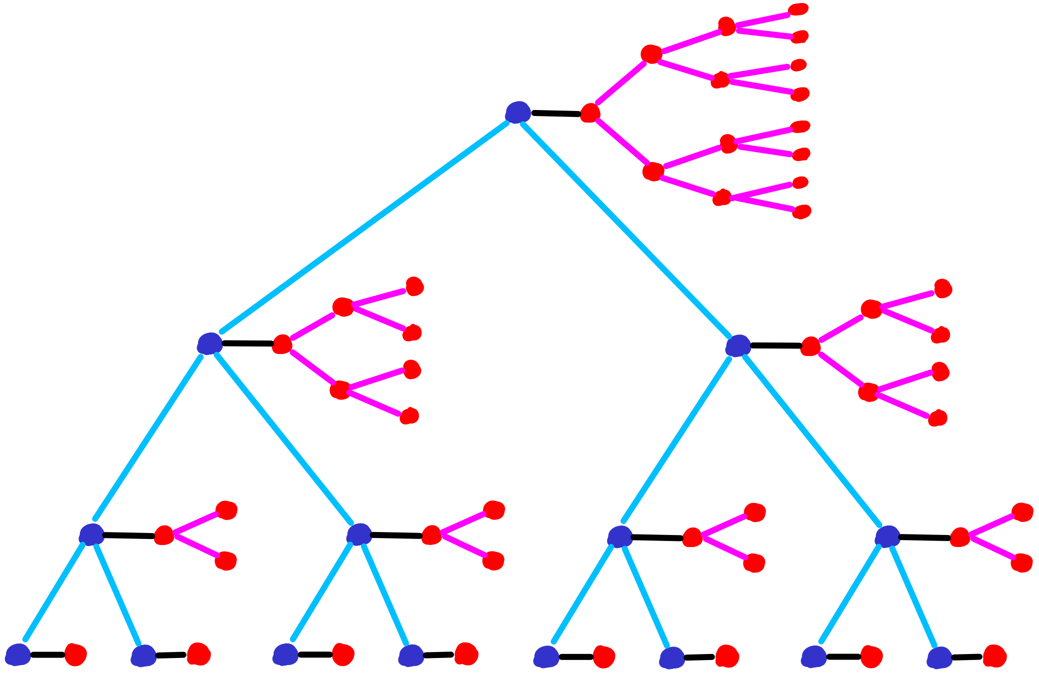
Every X-range is represented by O(logn) nodes



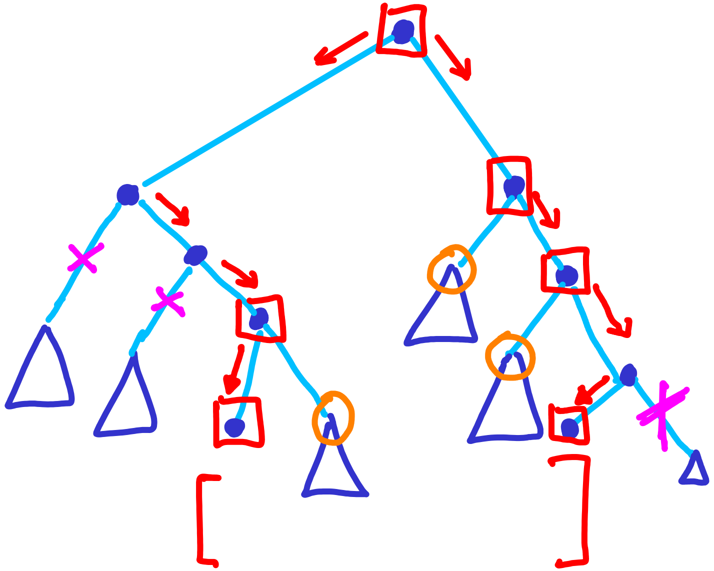


Every X-range is represented by O(logn) nodes

For each node, create a new (aux.) tree containing all nodes of subtree, sorted by Y.

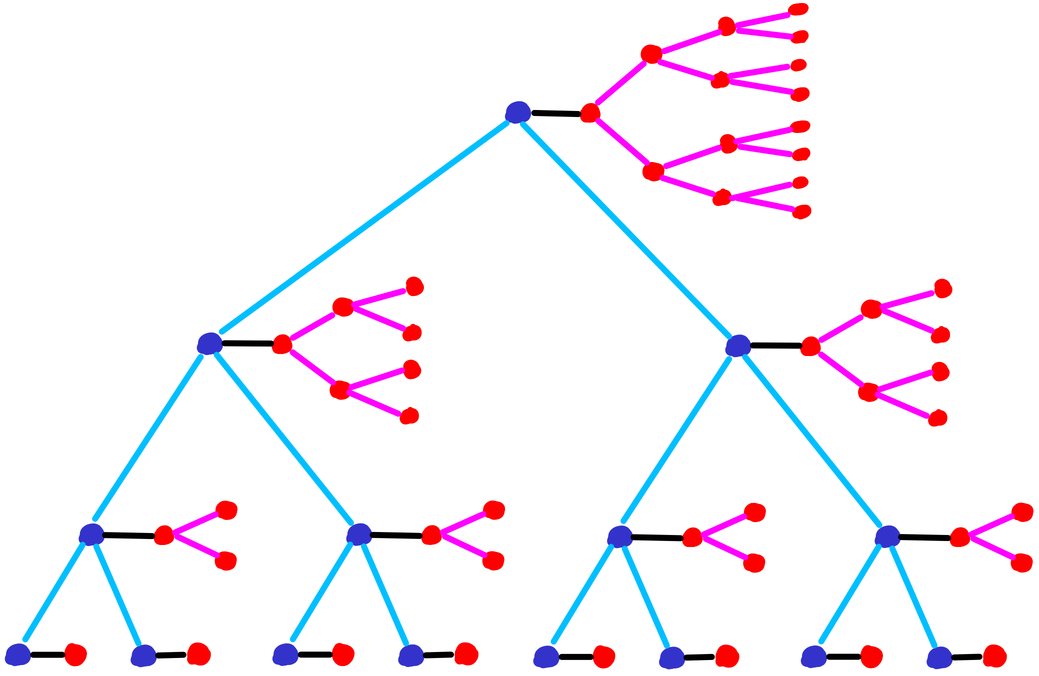


Size?



Every X-range is represented by $O(\log n)$ nodes

For each node, create a new (aux.) tree containing all nodes of subtree, sorted by Y.



size of aux trees

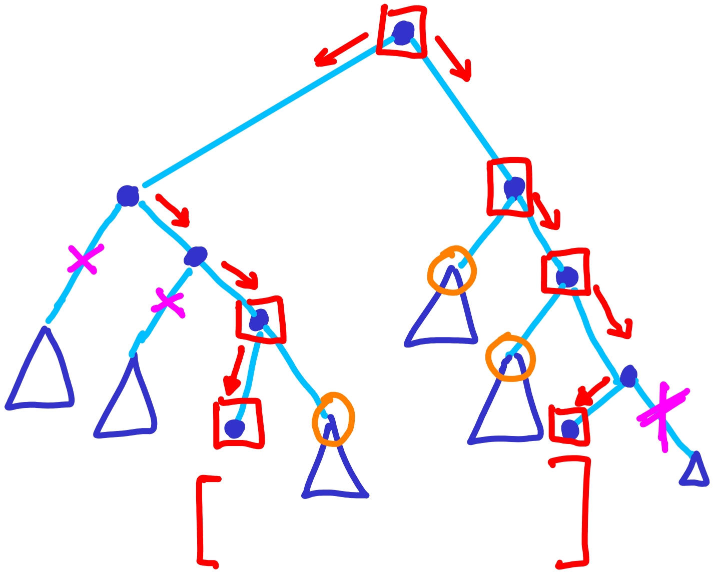
$$1 \cdot n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + 8 \cdot \frac{n}{8} + \dots\ n \cdot 1$$

root

leaves

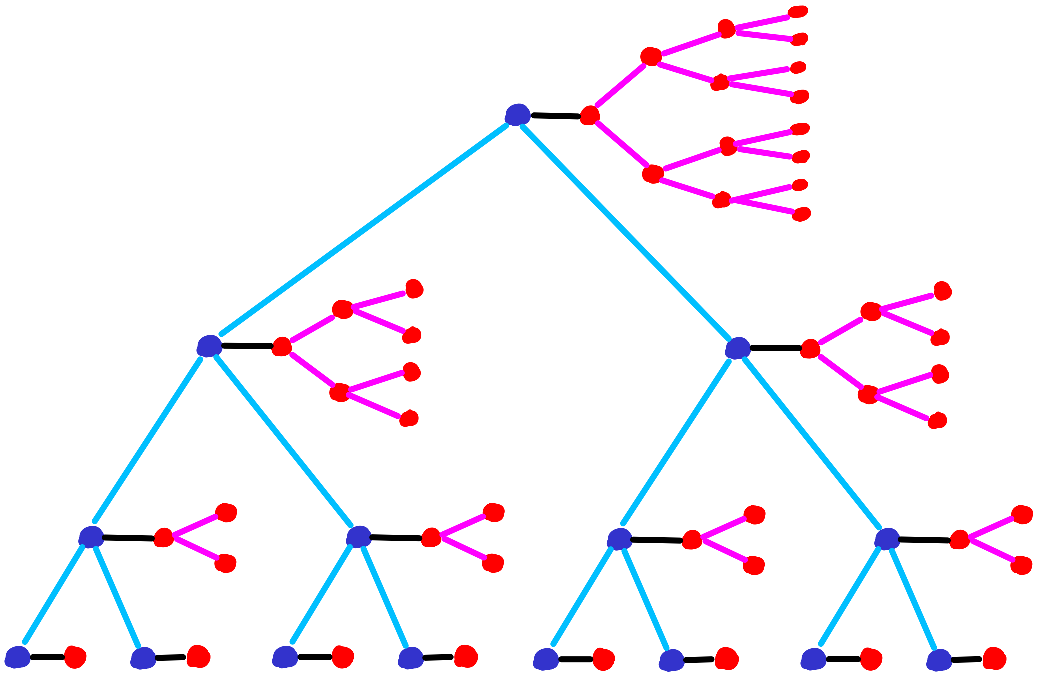
number of aux. trees

$= \Theta(n \log n$ space$)$



Every X-range is represented by O(logn) nodes

For each node, create a new (aux.) tree containing all nodes of subtree, sorted by Y.



size of aux trees

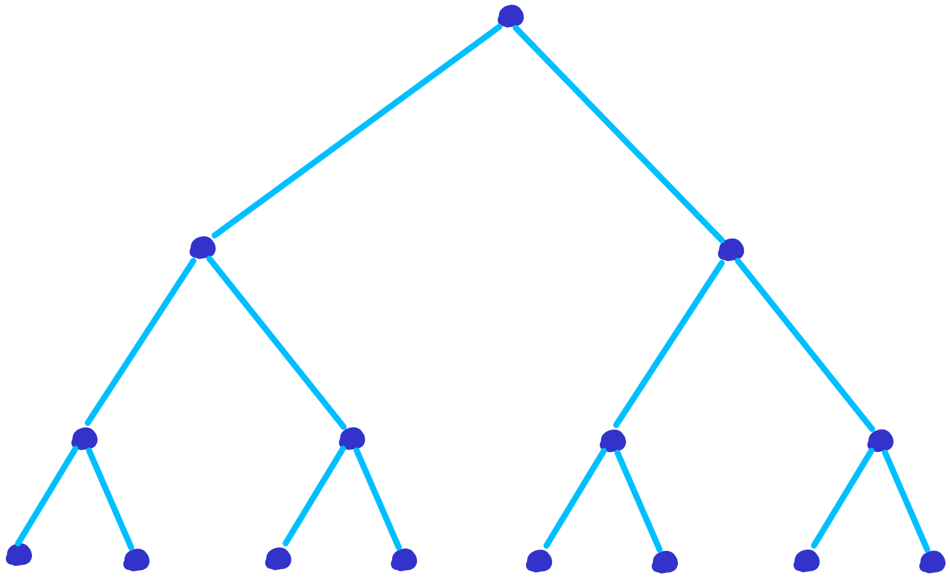
$$\underbrace{1}_{\text{root}} \cdot n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + 8 \cdot \frac{n}{8} + \cdots \underbrace{n}_{\text{leaves}} \cdot 1$$

number of aux. trees

$= \Theta(n \log n \text{ space})$

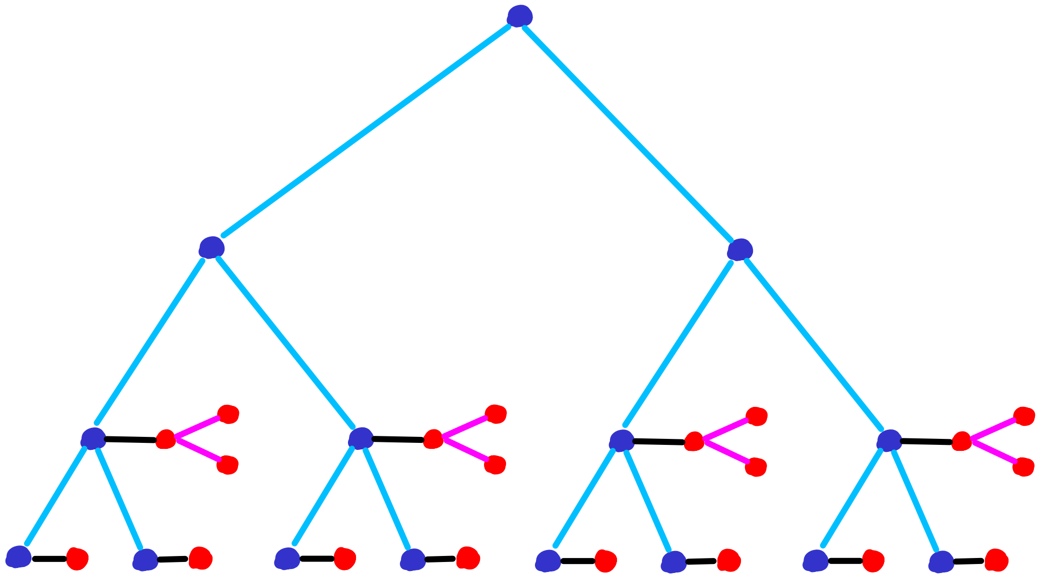
Every node is represented in O(logn) aux. subtrees

Build primary tree : $\Theta(n \log n)$ time



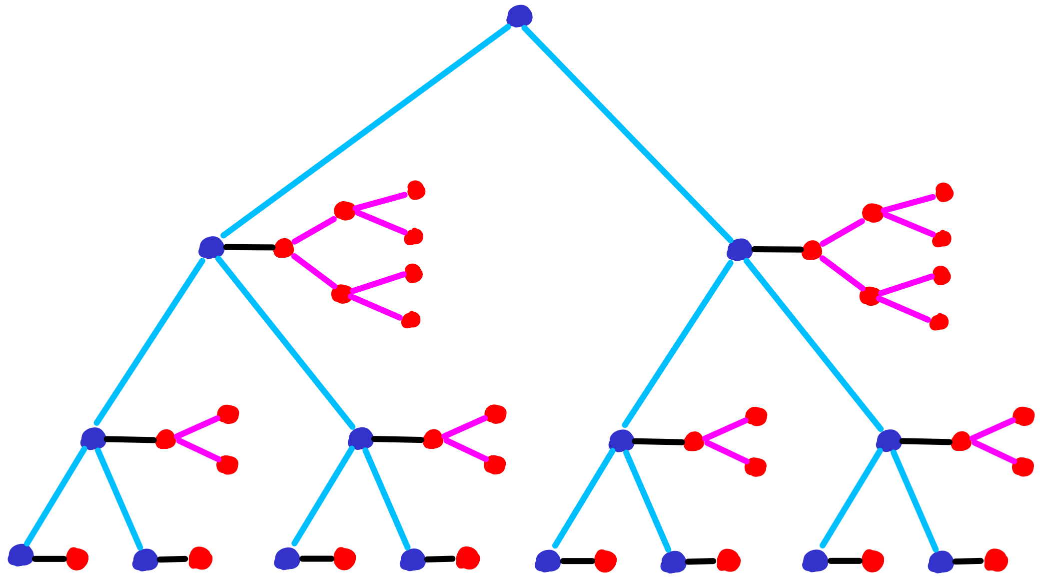
Build primary tree : $\Theta(n \log n)$ time

Build all aux. trees, bottom-up (merge)



Build primary tree : $\Theta(n\log n)$ time

Build all aux. trees, bottom-up (merge) : $\Theta(n\log n)$

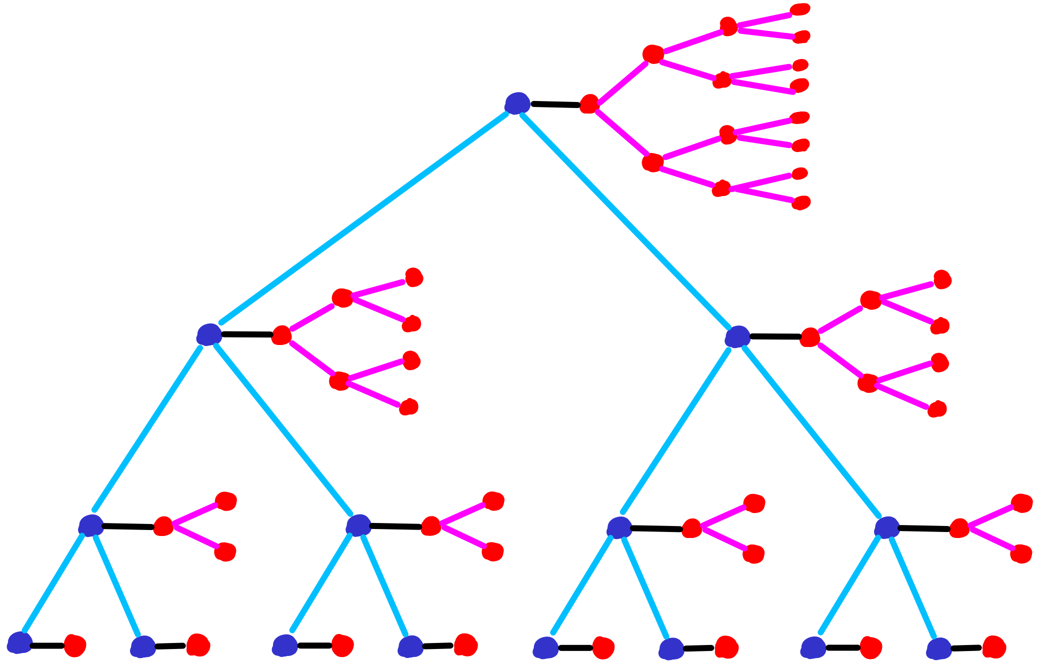


Build primary tree : $\Theta(n\log n)$ time

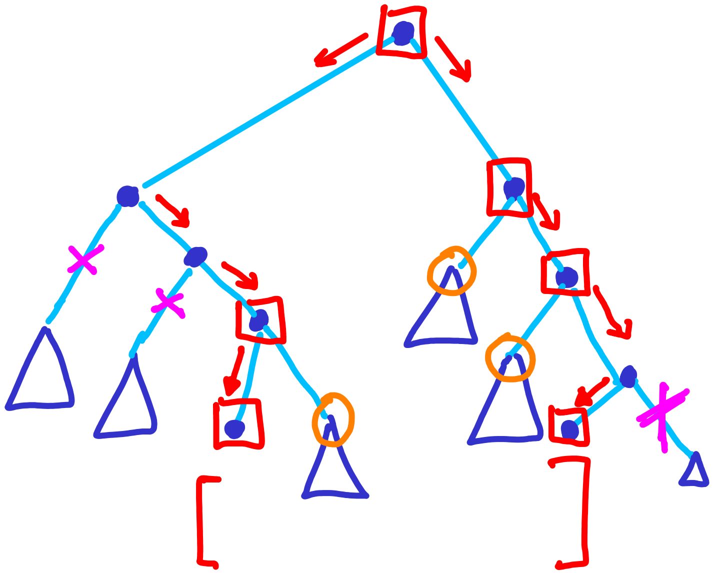
Build all aux. trees, bottom-up (merge) : $\Theta(n\log n)$

[if built independently: $T(n) = 2\cdot T(\frac{n}{2}) + \Theta(n\log n)$

$n\log n + 2(\frac{n}{2}\log\frac{n}{2}) + 4(\frac{n}{4}\log\frac{n}{4}) + \cdots \quad = \Theta(n\log^2 n)$]



(X,Y)-Query → Search X: identify O(log n) nodes $\boxed{\bullet}$ & ○

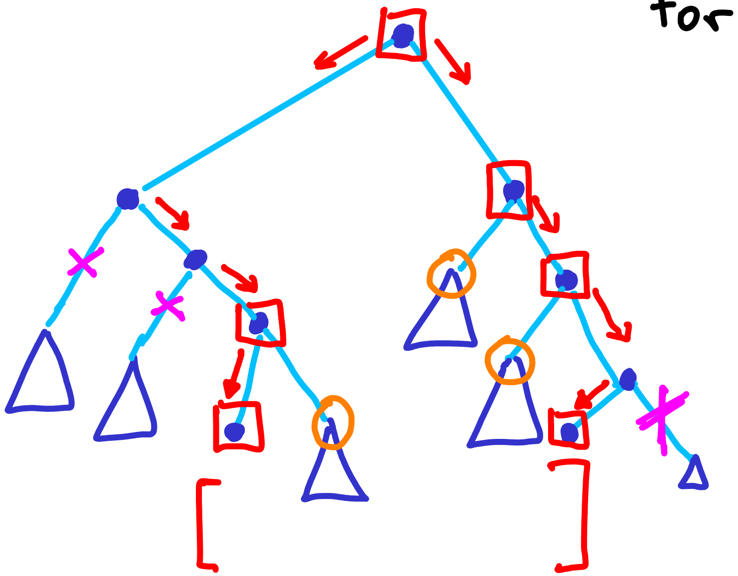


(X,Y)-Query → Search X: identify O(logn) nodes ▣ & ○

For each ▣ check if y-range is ok. O(1)

TOTAL WORK

O(logn)

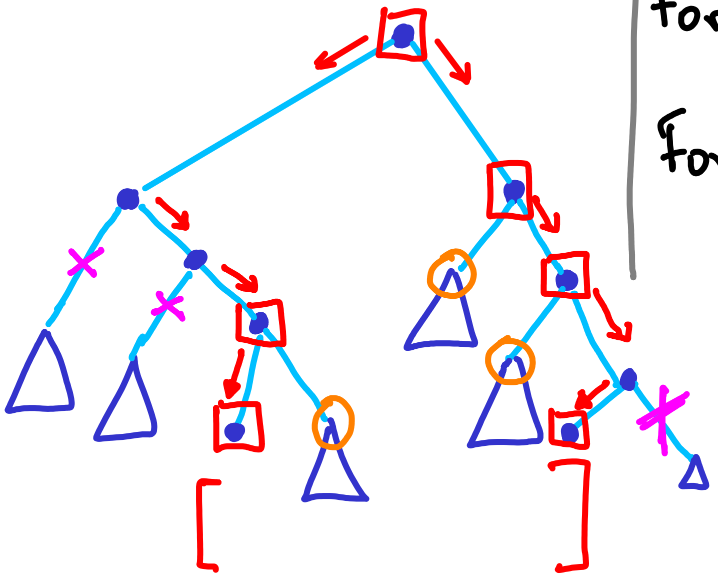


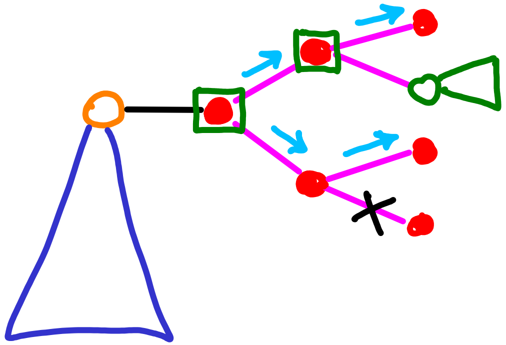
(X,Y)-Query → Search X: identify $O(\log n)$ nodes $\boxed{\cdot}$ & ○

TOTAL WORK

For each $\boxed{\cdot}$ check if y-range is ok. $O(1)$ — $O(\log n)$

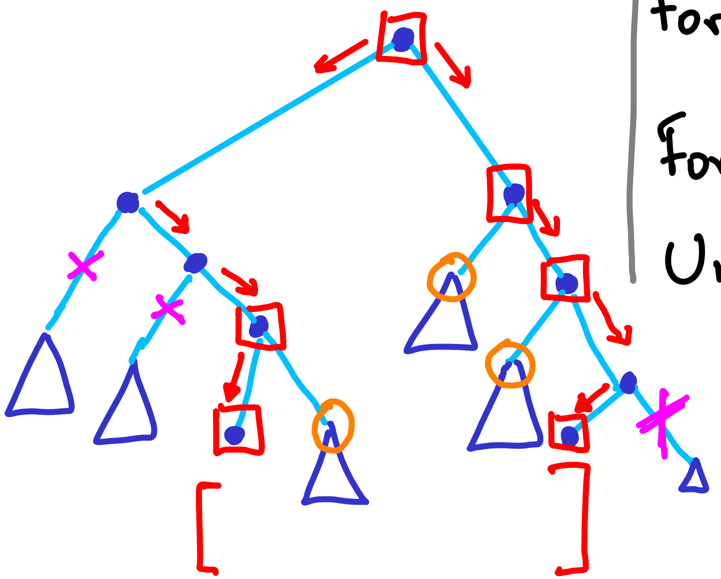
For each ○ check y-range of aux. tree $O(\log n)$ — $O(\log^2 n)$

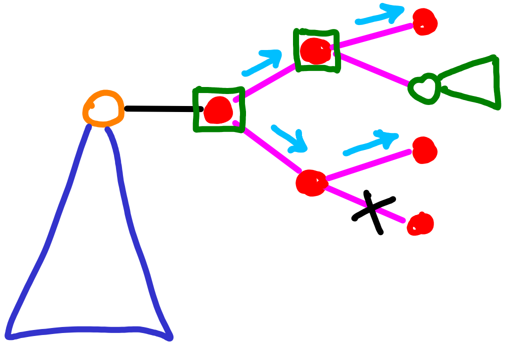




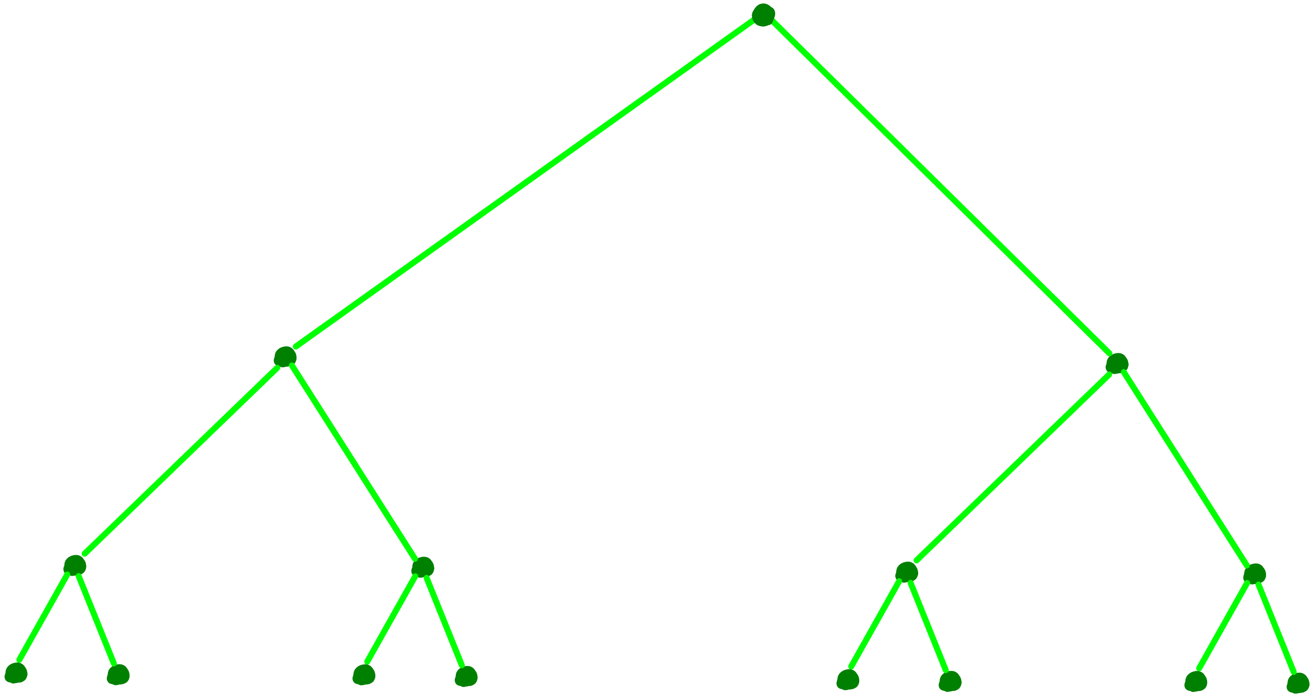
(X,Y)-Query → Search X: identify $O(\log n)$ nodes □ & ○

		TOTAL WORK
For each □ check if y-range is ok.	$O(1)$	$O(\log n)$
For each ○ check y-range of aux.tree	$O(\log n)$	$O(\log^2 n)$
Union results		$O(\log n)$

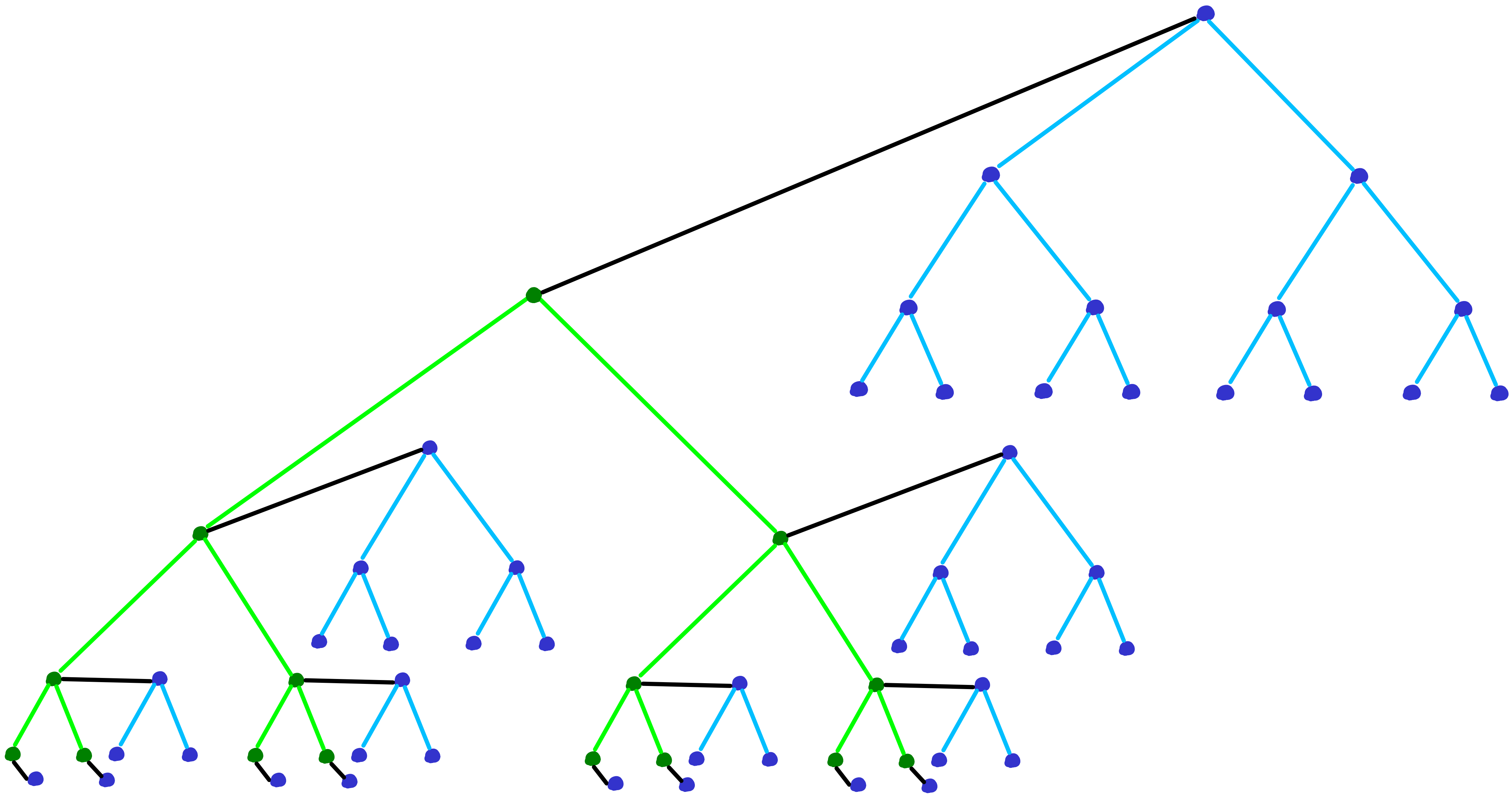




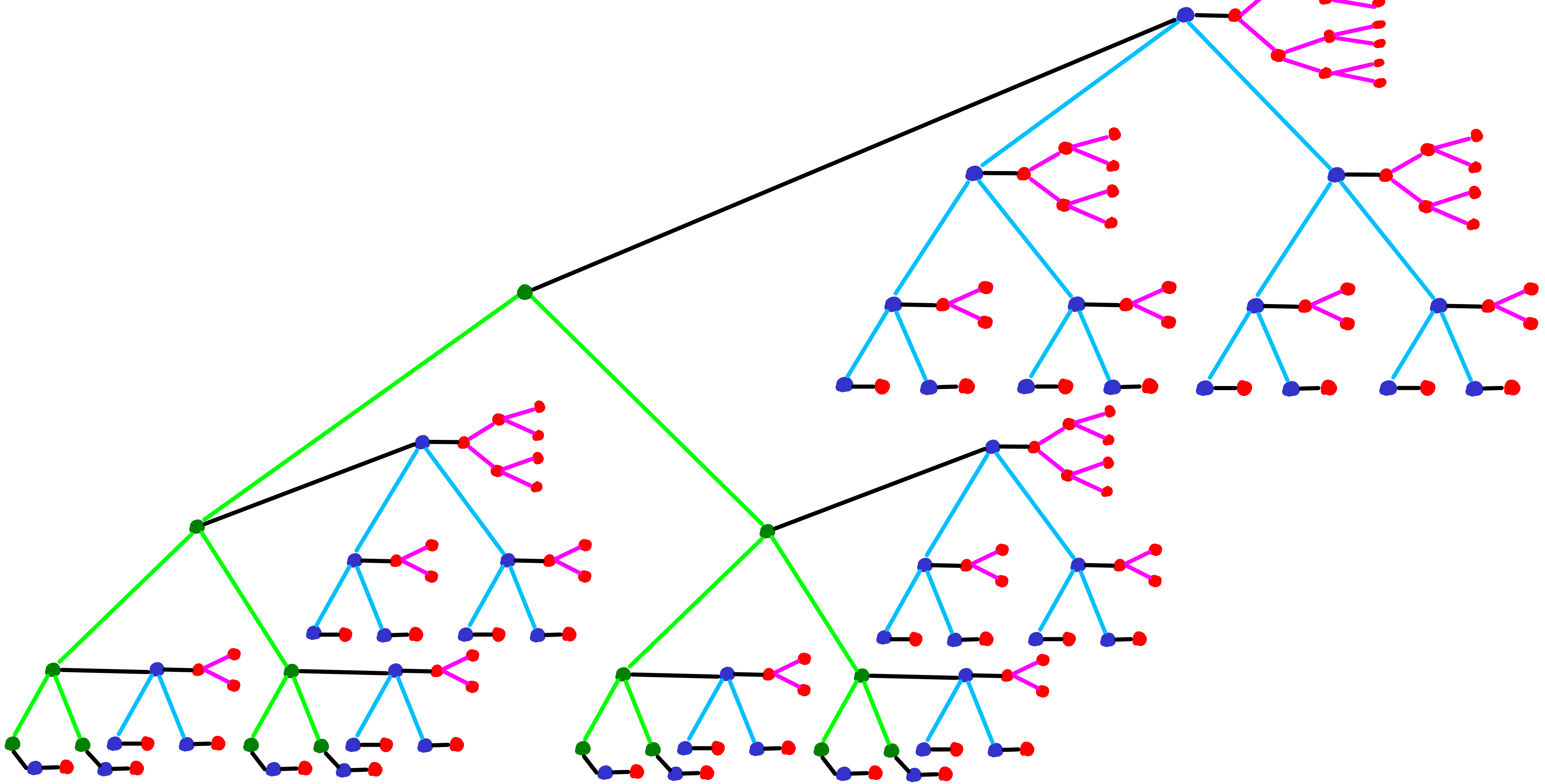
1D



2D : tree of trees



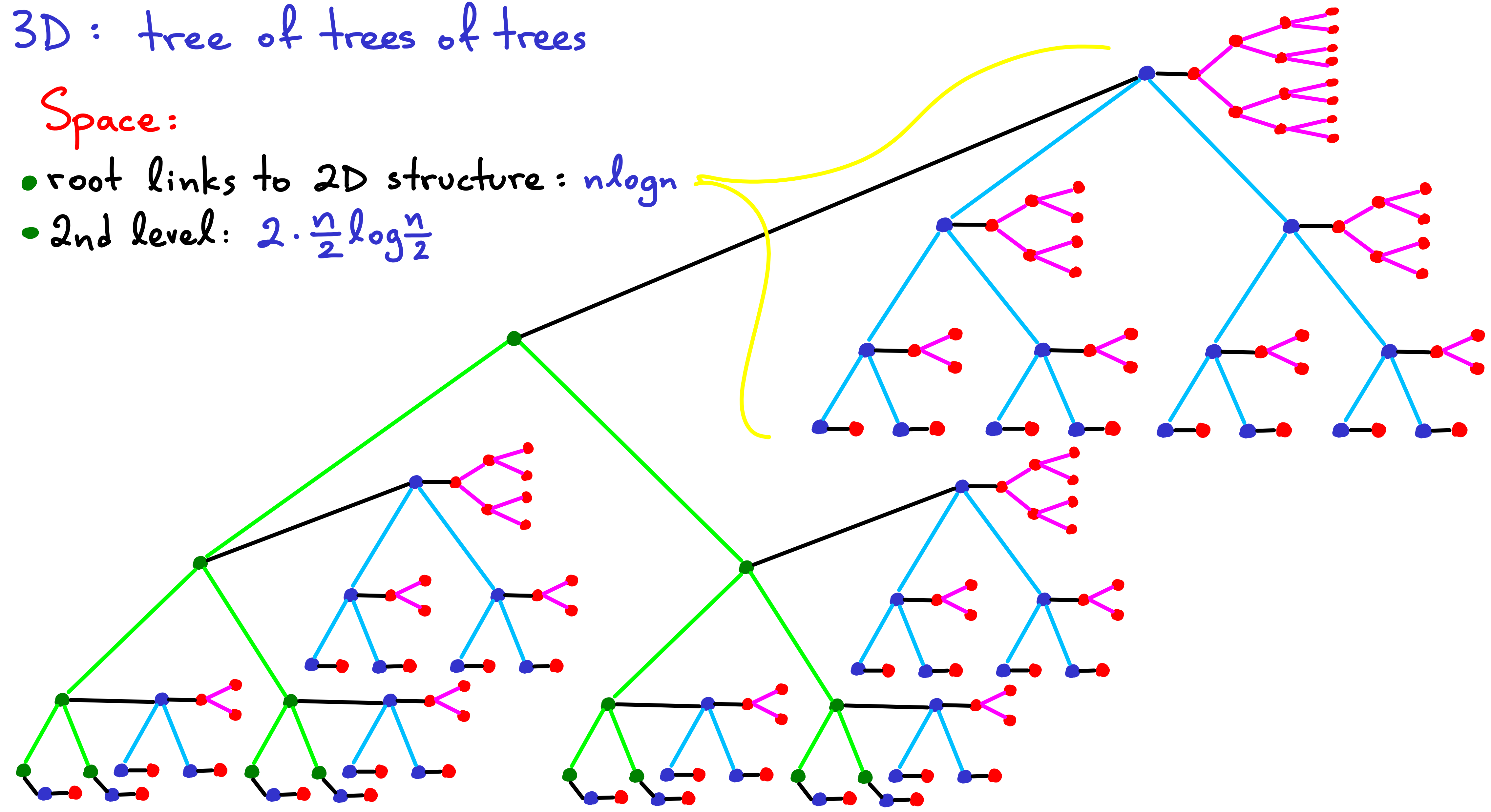
3D : tree of trees of trees



3D : tree of trees of trees

Space:

- root links to 2D structure : $n \log n$
- 2nd level : $2 \cdot \frac{n}{2} \log \frac{n}{2}$

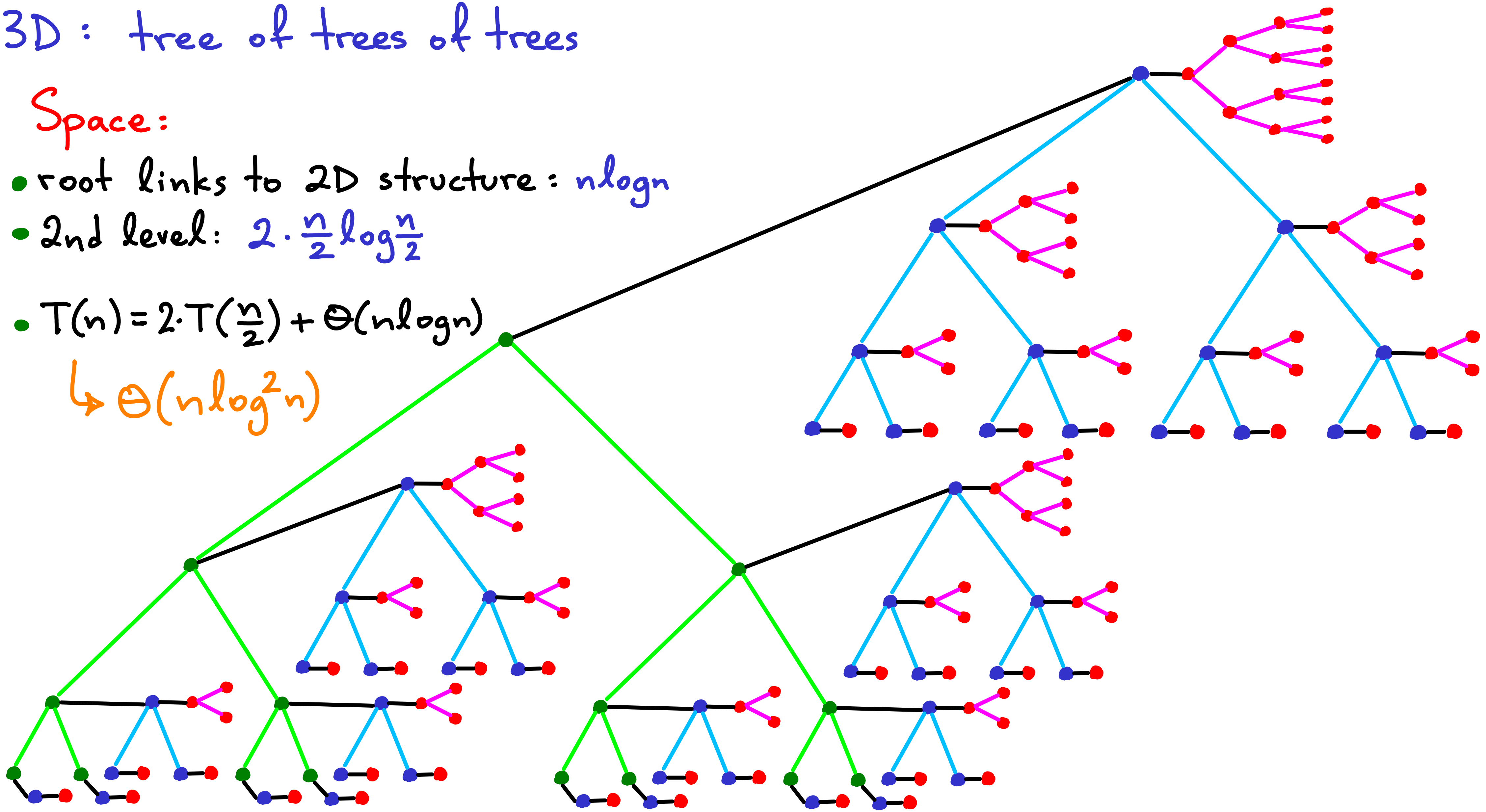


3D : tree of trees of trees

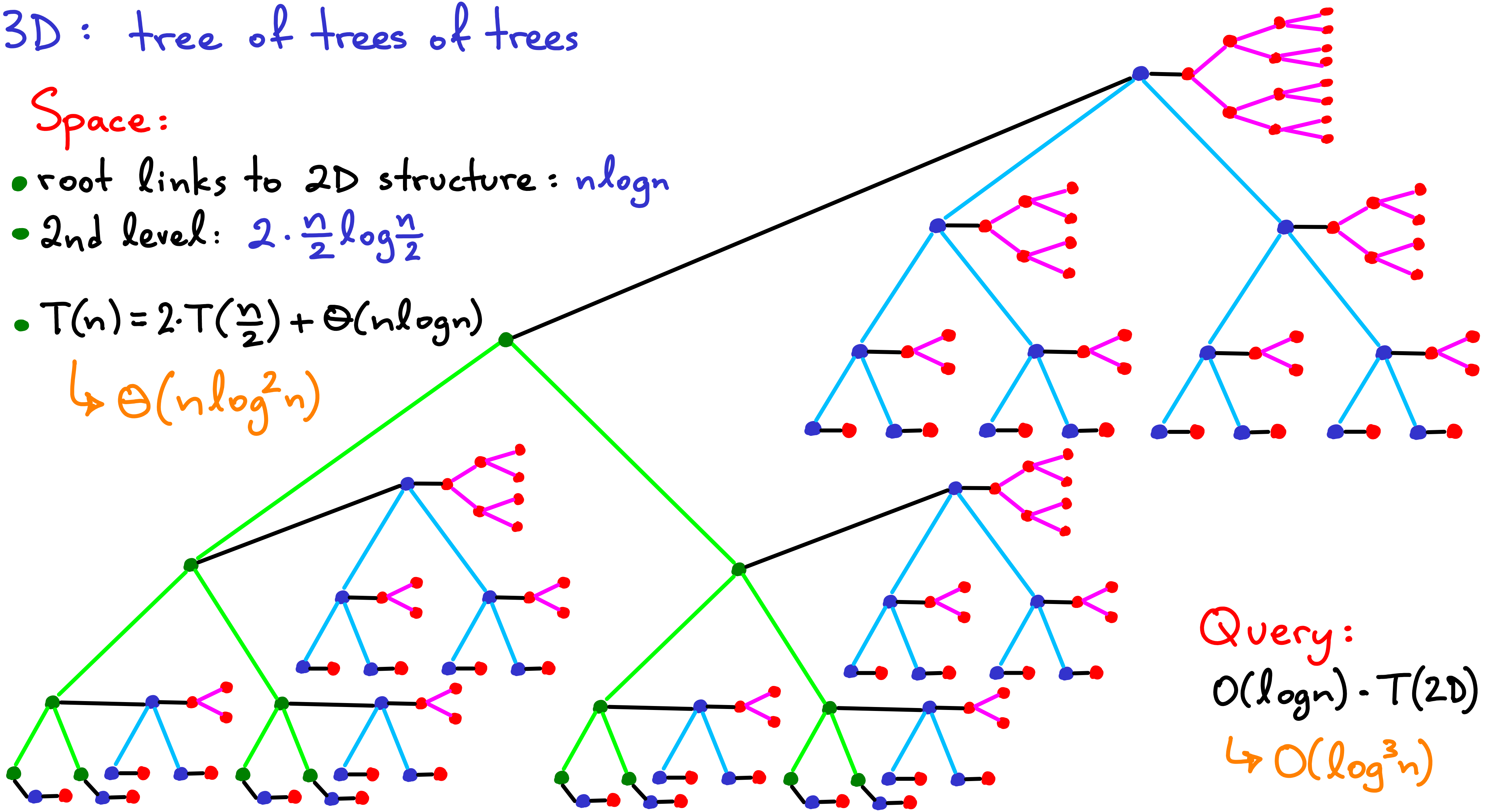
Space:

- root links to 2D structure : $n \log n$
- 2nd level: $2 \cdot \frac{n}{2} \log \frac{n}{2}$
- $T(n) = 2 \cdot T(\frac{n}{2}) + \Theta(n \log n)$

$\hookrightarrow \Theta(n \log^2 n)$



3D : tree of trees of trees



Space:

- root links to 2D structure : $n \log n$
- 2nd level: $2 \cdot \frac{n}{2} \log \frac{n}{2}$
- $T(n) = 2 \cdot T(\frac{n}{2}) + \Theta(n \log n)$

  $\hookrightarrow \Theta(n \log^2 n)$

Query:

$O(\log n) \cdot T(2D)$

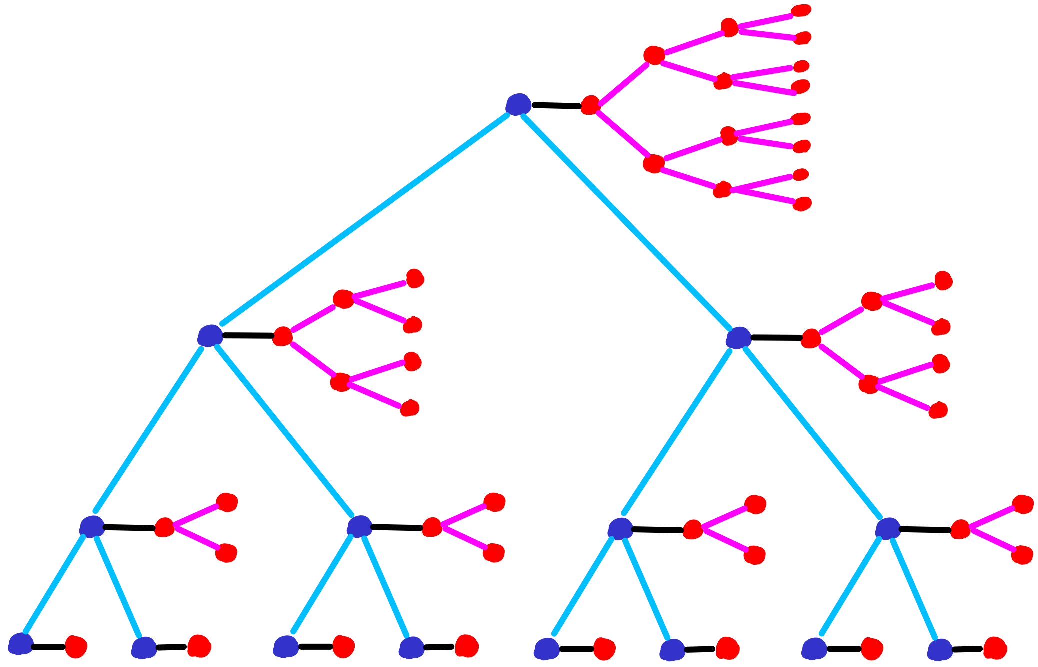
$\hookrightarrow O(\log^3 n)$

RANGE COUNTING

add $\Theta(R)$ to report R items

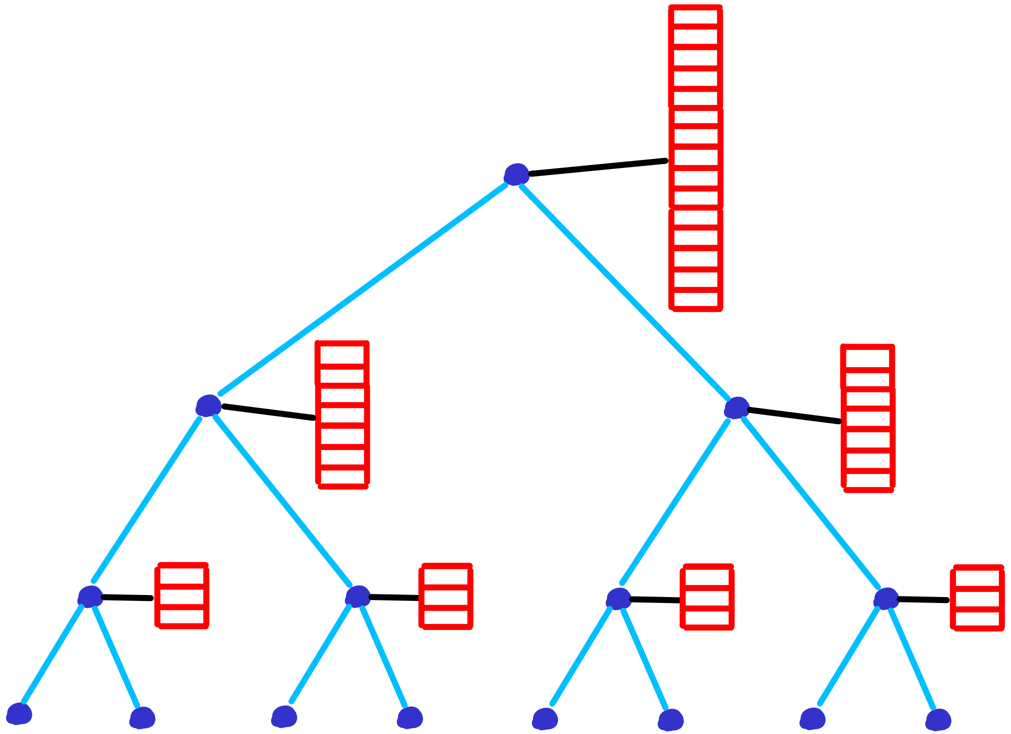
	size of structure	query time
1D Tree	$\Theta(n)$	$\Theta(\log n)$
2D Tree of trees	$\Theta(n \log n)$	$O(\log^2 n)$
$k \geqslant 3$ tree (of trees)$^{k-1}$	$O(n \log^{k-1} n)$	$O(\log^k n)$

Back to 2D: tree of trees (just static)



Back to 2D: tree of trees (just static)

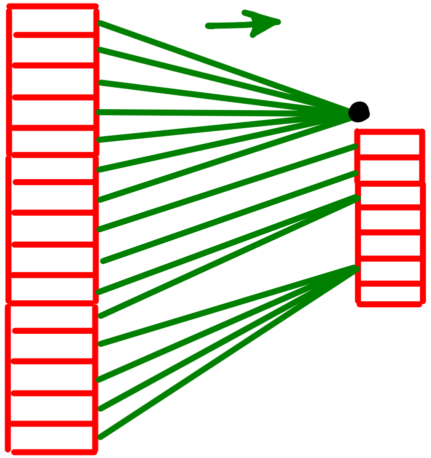
Replace aux. trees with arrays → doesn't change search idea (or time)

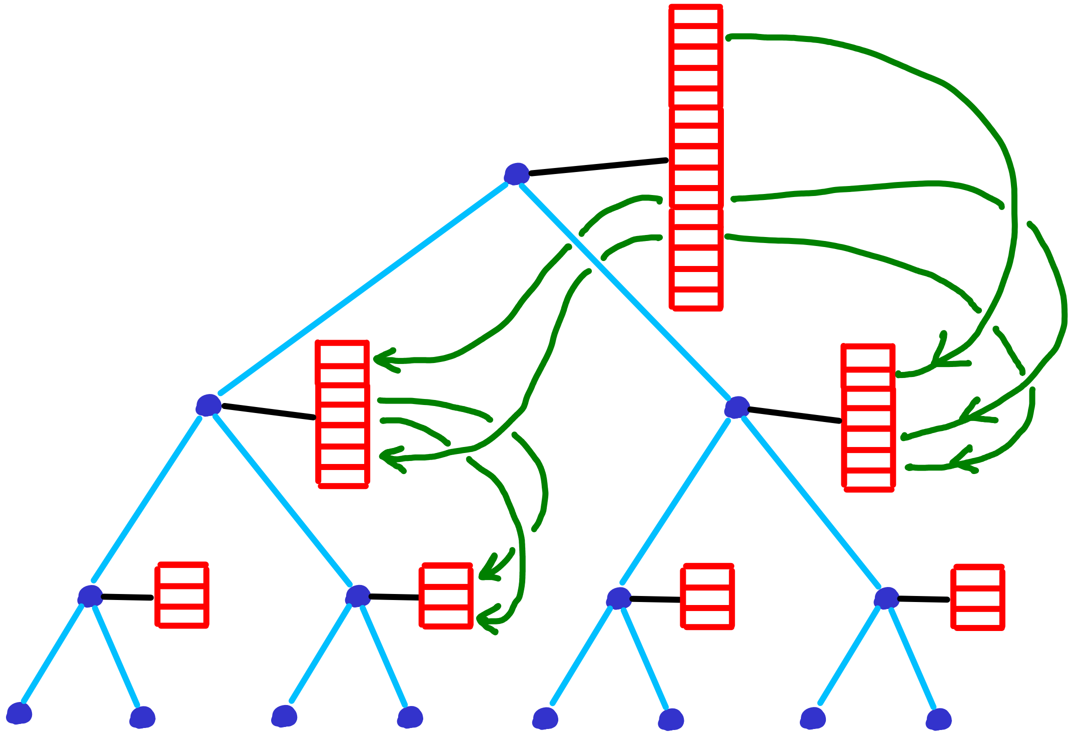


Back to 2D: tree of trees (just static)

Replace aux. trees with arrays → doesn't change search idea (or time)

For every array,
add a pointer from every key
to the position it would be at
in each child array

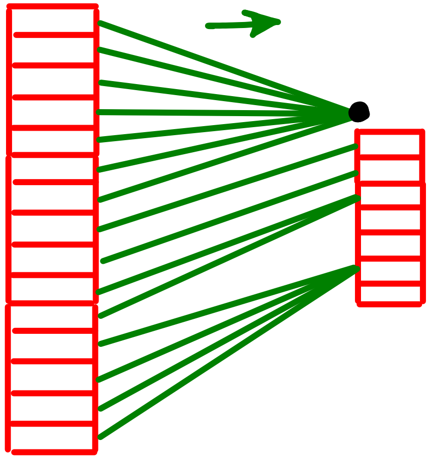




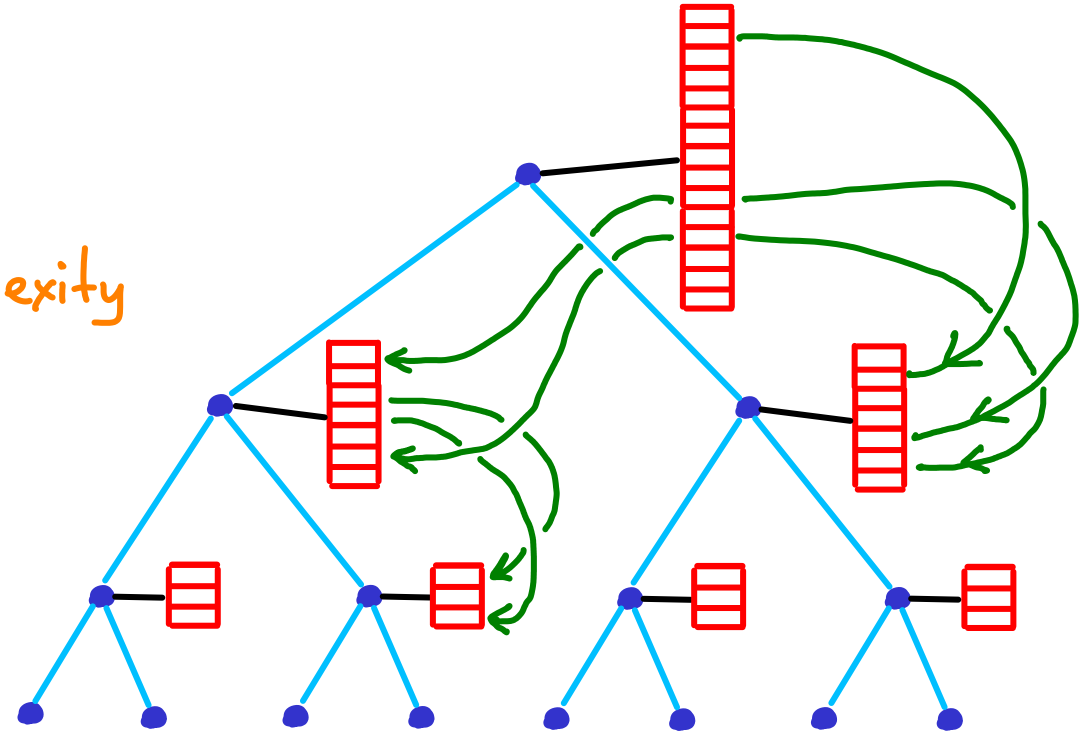
Back to 2D: tree of trees (just static)

Replace aux. trees with arrays → doesn't change search idea (or time)

For every array,
add a pointer from every key
to the position it would be at
in each child array

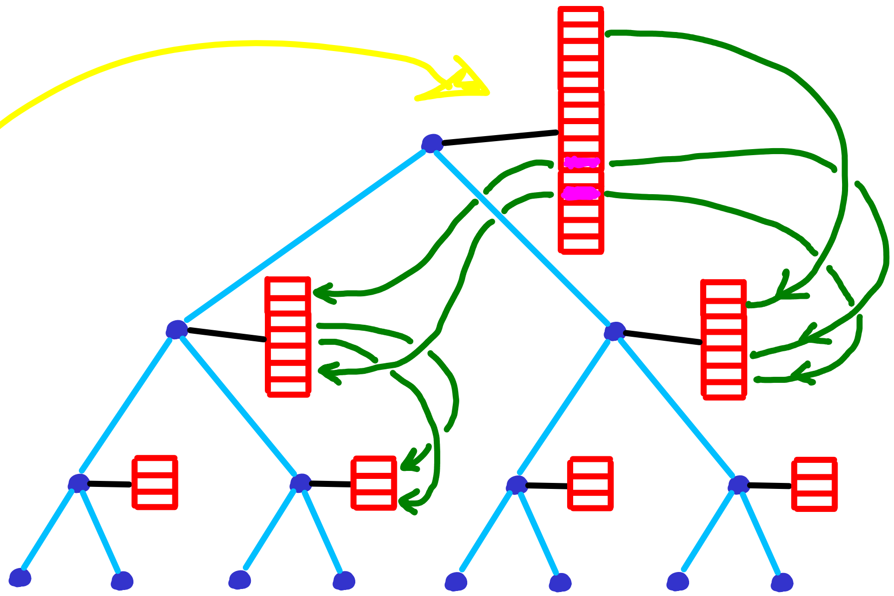


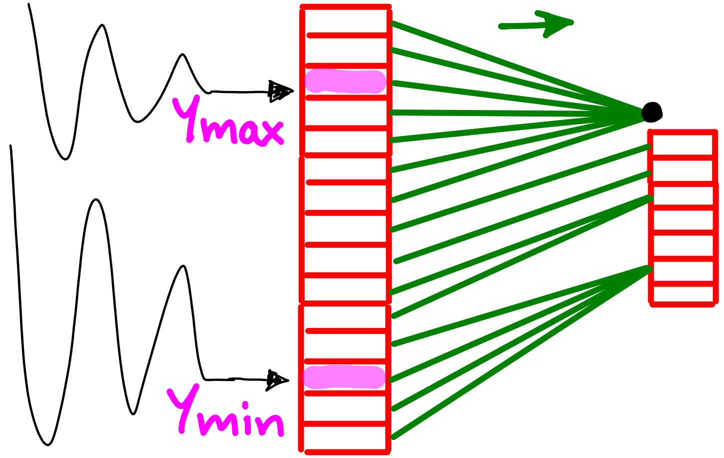
Same space complexity



New (X,Y)-search:

1) binary search by Y at root (array)
↳ mark Ymax & Ymin



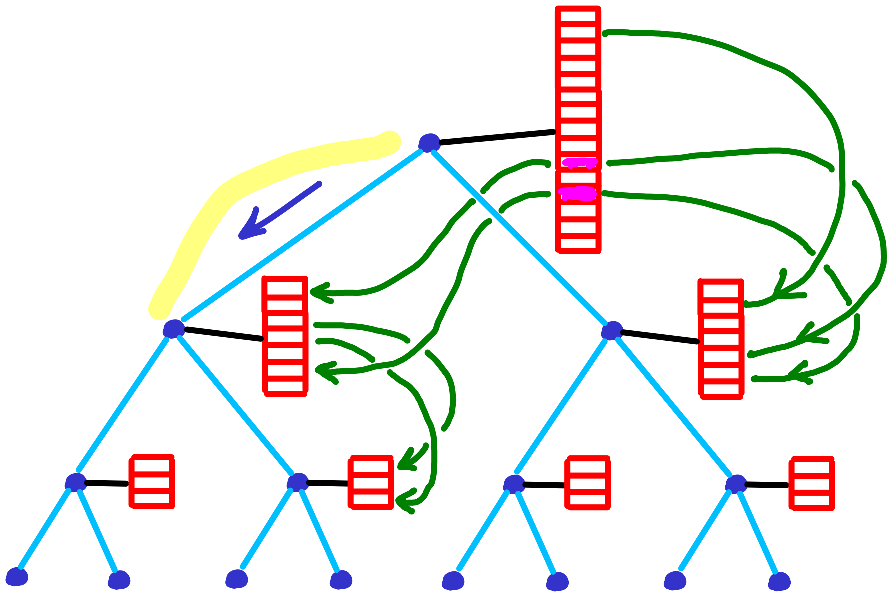


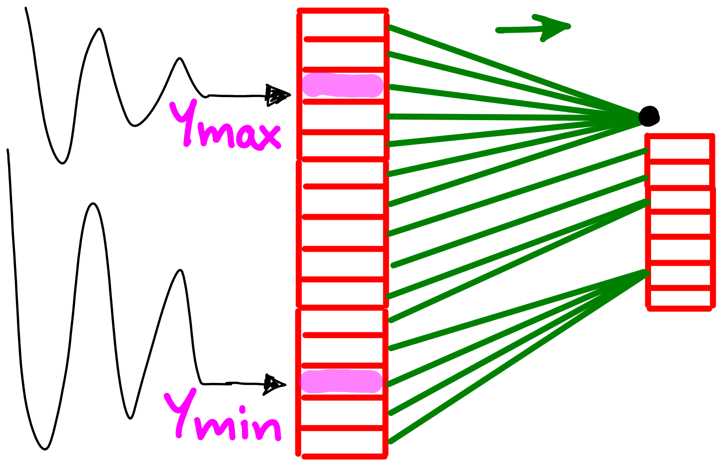
New (X,Y)-search:

1) binary search by Y at root (array)

↳ mark Ymax & Ymin

2) do regular search by X ...



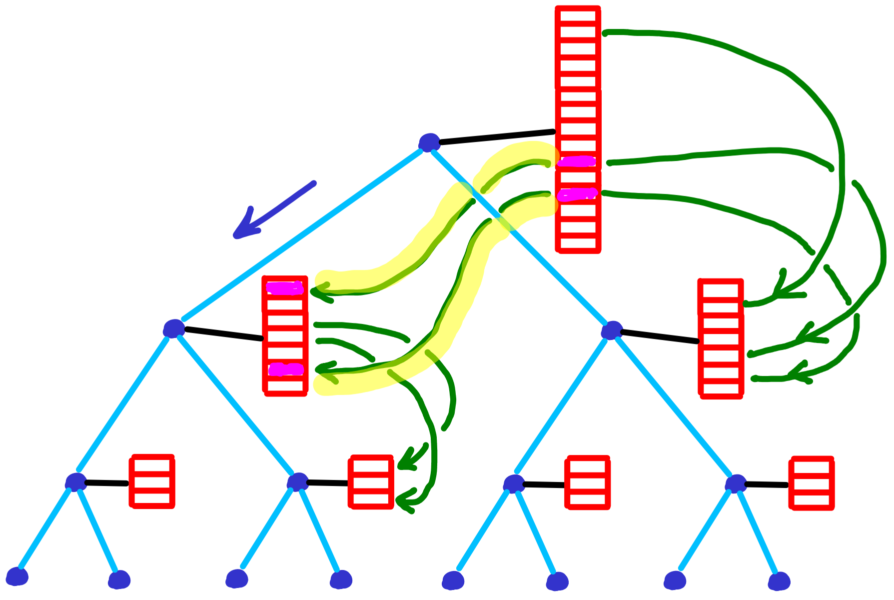


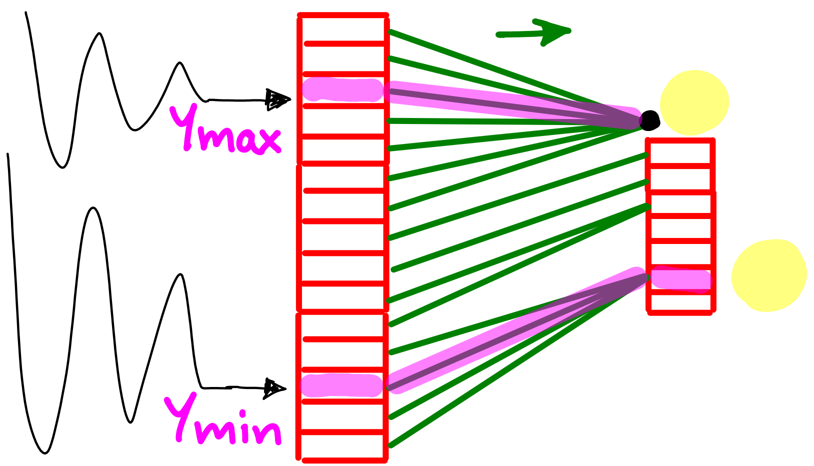
New (X,Y)-search:

1) binary search by Y at root (array)
↳ mark Ymax & Ymin

2) do regular search by X but also:

- for every node visited,
  follow pointers to child array
  ↳ tells us #points in Y range

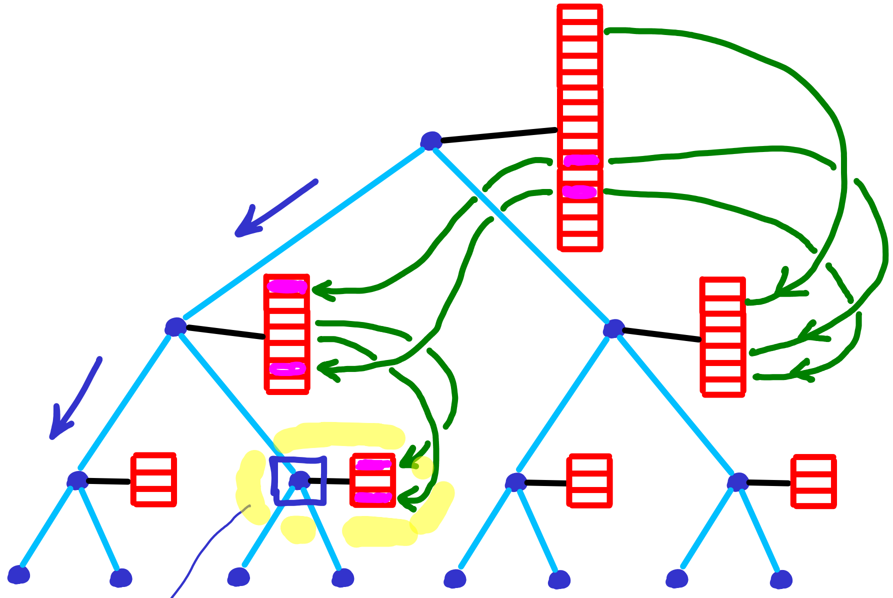




New (X,Y)-search:

1) binary search by Y at root (array)

↳ mark Ymax & Ymin



2) do regular search by X but also:

- for every node visited,

follow pointers to child array

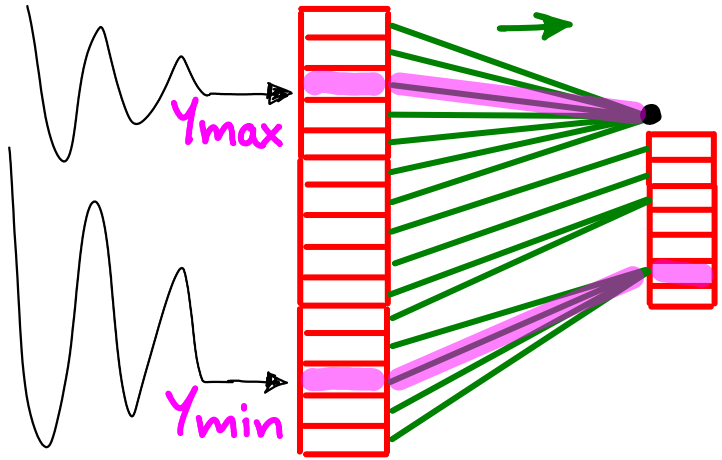
↳ tells us #points in Y range

- when we have a subtree entirely in X-range

↳ we get #pts also in Y-range

in O(1) time

instead of O(logn)



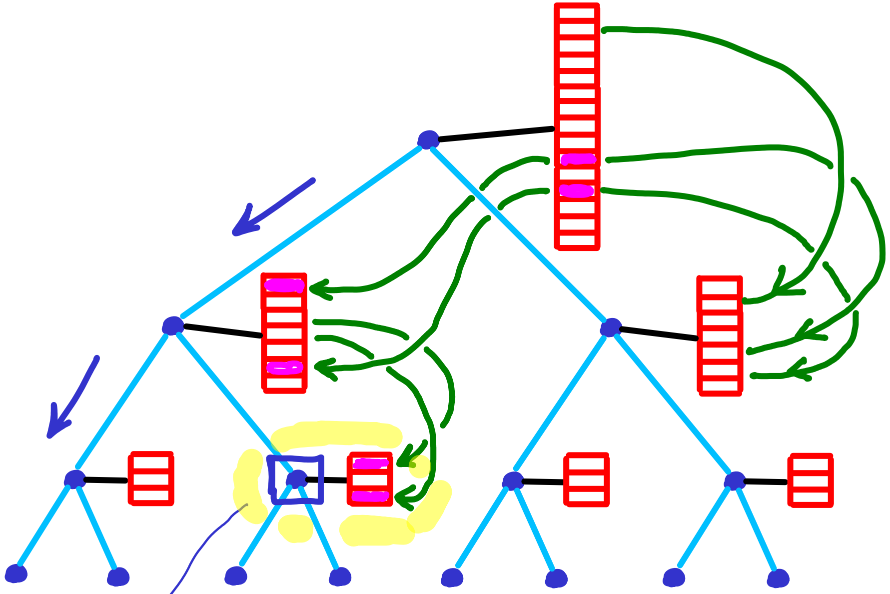
New (X,Y)-search:

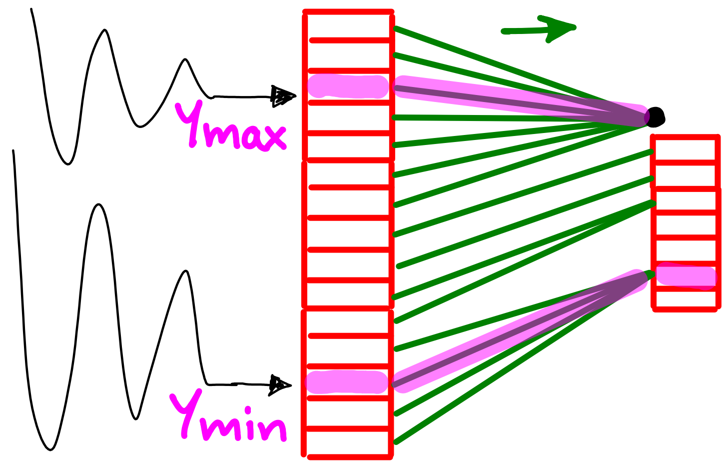
1) binary search by Y at root (array)
   ↳ mark Ymax & Ymin

2) do regular search by X but also:

- for every node visited,
  follow pointers to child array
  ↳ tells us #points in Y range

- when we have a subtree entirely in X-range
  ↳ we get #pts also in Y-range
  in O(1) time
  instead of O(log n)





Time: O(log n)

Works only for last level : time = $O(\log^{d-1} n)$

Can be made dynamic

Can be improved

projects

RANGE COUNTING

add $\Theta(R)$ to report R items

	size of structure	query time
1D Tree	$\Theta(n)$	$\Theta(\log n)$
2D Tree of trees	$\Theta(n \log n)$	$O(\log^2 n) \rightarrow \Theta(\log n)$

RANGE COUNTING

add $\Theta(R)$ to report R items

	size of structure	query time
1D Tree	$\Theta(n)$	$\Theta(\log n)$
2D Tree of trees	$\Theta(n \log n)$	$O(\log^2 n) \rightarrow \Theta(\log n)$
$k \geq 3$ tree (of trees)$^{k-1}$	$O(n \log^{k-1} n)$	$O(\log^k n) \rightarrow O(\log^{k-1} n)$

RANGE COUNTING

add $\Theta(R)$ to report R items

	size of structure	query time
1D Tree	$\Theta(n)$	$\Theta(\log n)$
2D Tree of trees	$\Theta(n \log n)$	$O(\log^2 n) \rightarrow \Theta(\log n)$
$k \geqslant 3$ tree (of trees)$^{k-1}$	$O(n \log^{k-1} n) \rightarrow O\left(n \cdot \frac{\log^{k-1} n}{\log\log n}\right)$	$O(\log^k n) \rightarrow O(\log^{k-1} n)$

RANGE COUNTING

add $\Theta(R)$ to report R items

	size of structure	query time
1D Tree	$\Theta(n)$	$\Theta(\log n)$
2D Tree of trees	$\Theta(n \log n)$	$O(\log^2 n) \rightarrow \Theta(\log n)$
$k \geq 3$ tree (of trees)$^{k-1}$	$O(n \log^{k-1} n) \rightarrow O\left(n \cdot \frac{\log^{k-1} n}{\log\log n}\right)$	$O(\log^k n) \rightarrow O(\log^{k-1} n)$
paper by B. Chazelle:	$\Theta(n \log^k n)$	$O(\log^{k-2} n)$

RANGE COUNTING

add $\Theta(R)$ to report R items

	size of structure	query time
$k=2$ k-d tree (see separate pdf)	$\Theta(n)$	$O(\sqrt{n})$
Tree of trees	$\Theta(n\log n)$	$O(\log n)$
Dimension $k \geq 3$		
k-d tree	$\Theta(kn)$	$O(k\cdot n^{1-\frac{1}{k}})$
$(\text{tree of})^{k-1}$ trees	$O(n\log^{k-1} n)$	$O(\log^{k-1} n)$