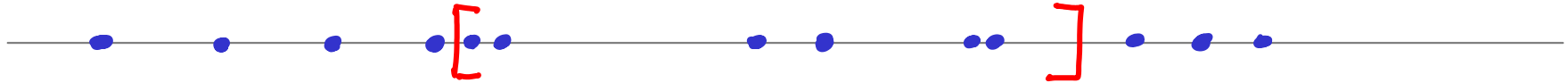
RANGE COUNTING

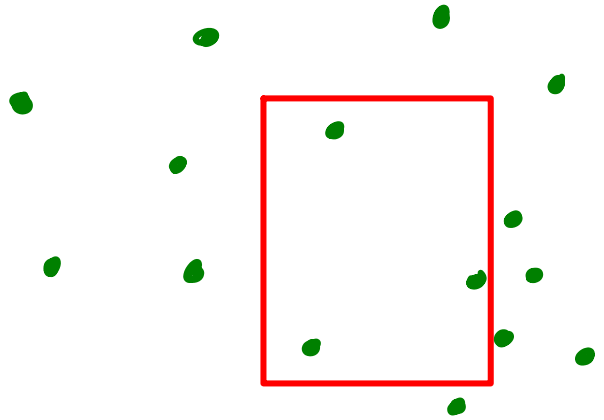
COUNT (or ENUMERATE) OBJECTS IN A GIVEN RANGE

(many times)

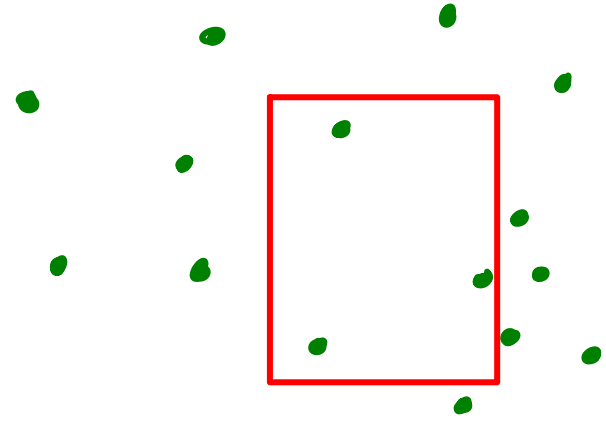
1D :



2D :



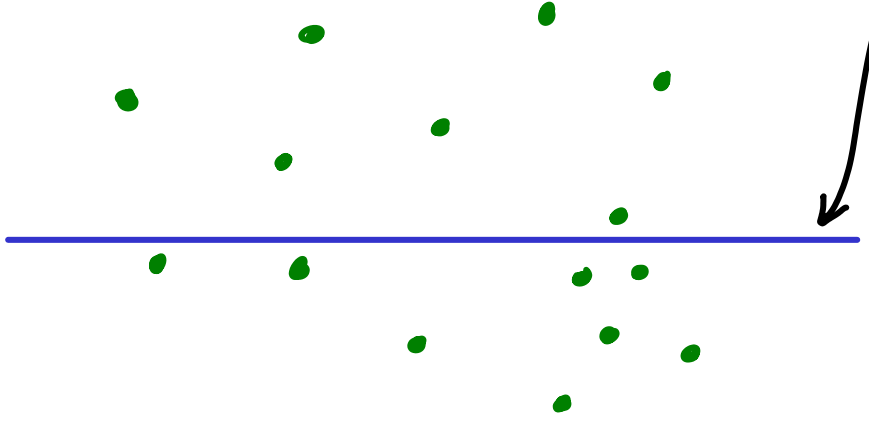
2D :

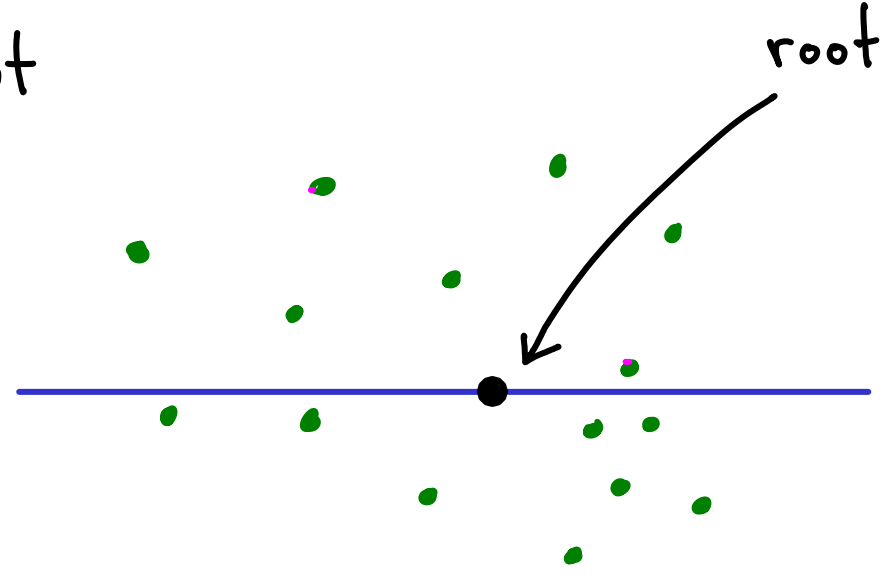


K-d tree (k=2)

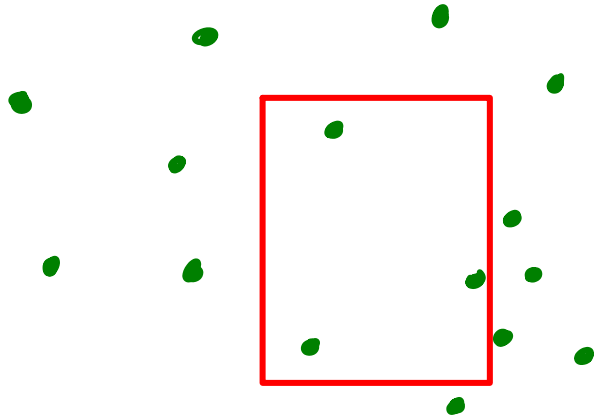
Recursively cut at median

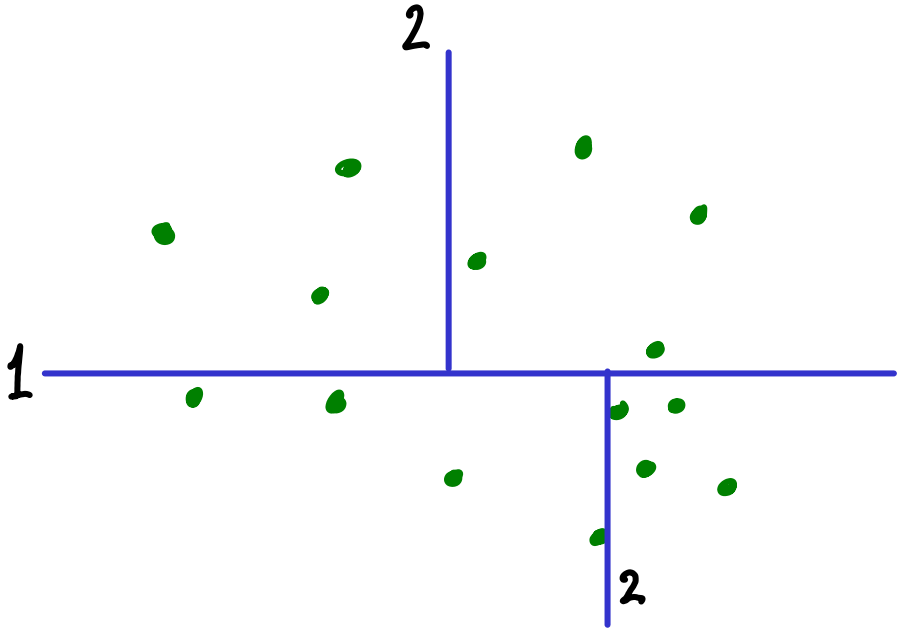
alternating ↔ vs ↕

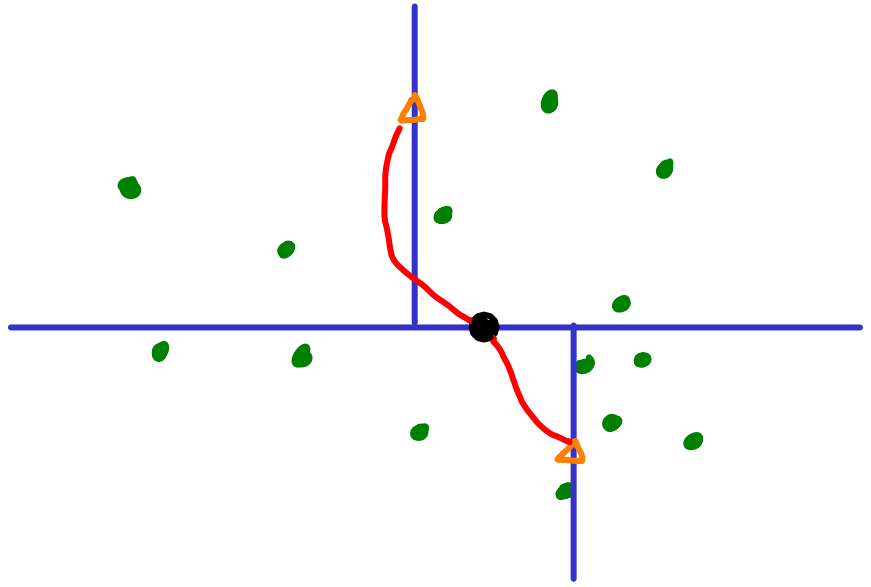




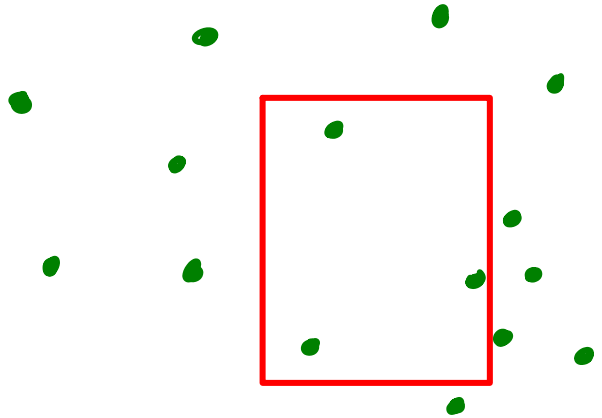
2D :

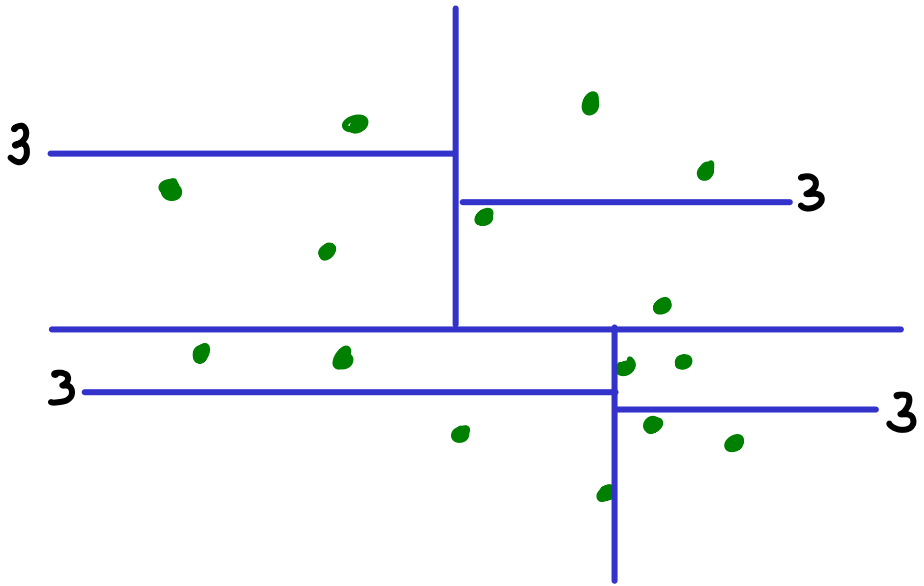


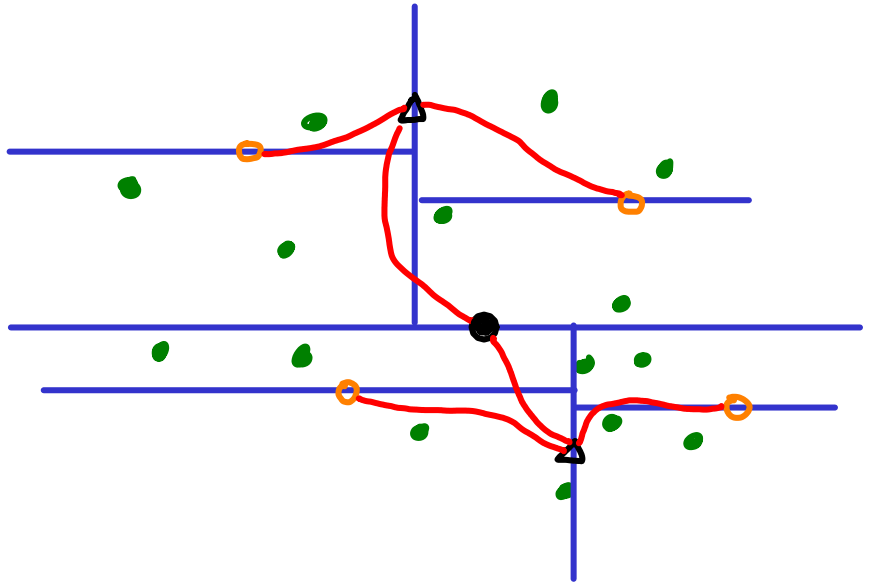




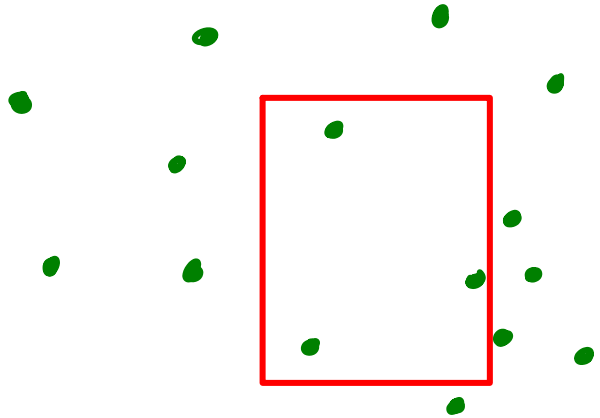
2D :

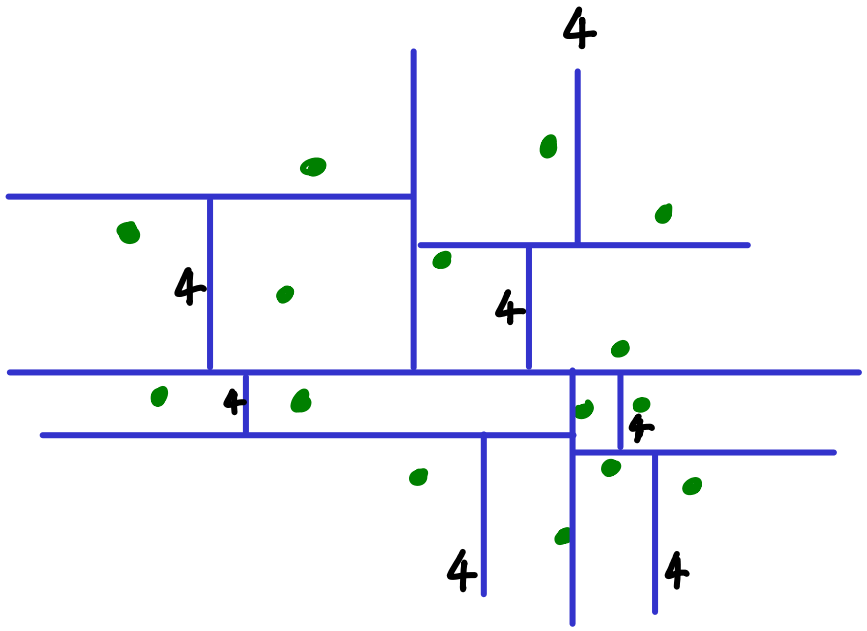


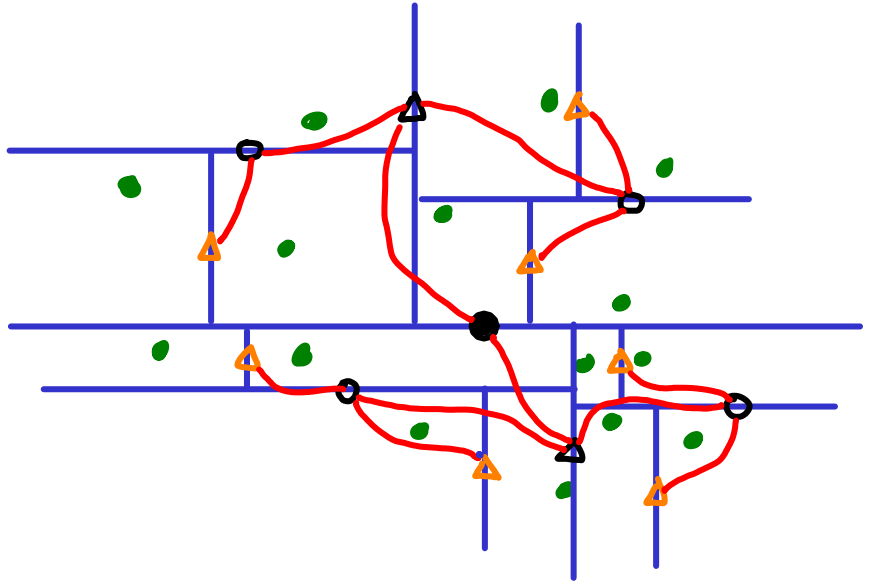




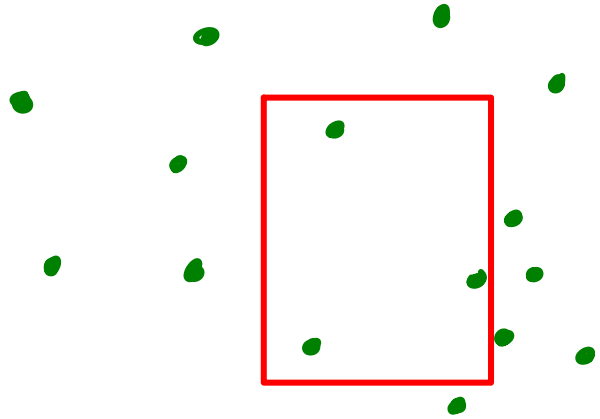
2D :

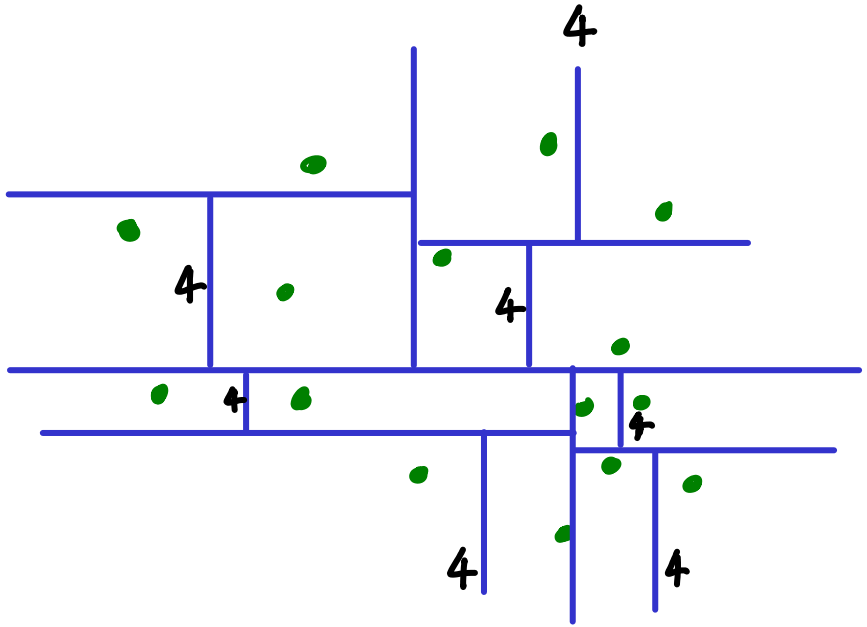




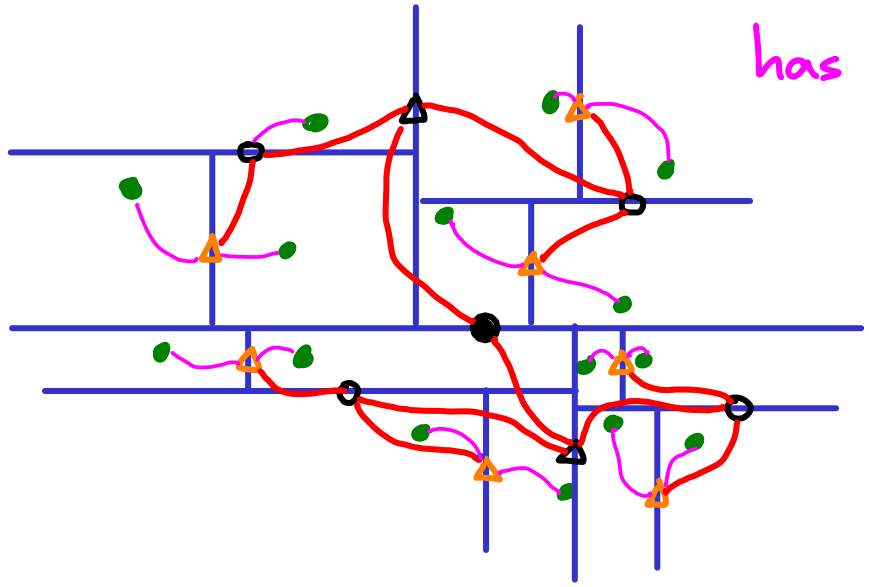


2D :

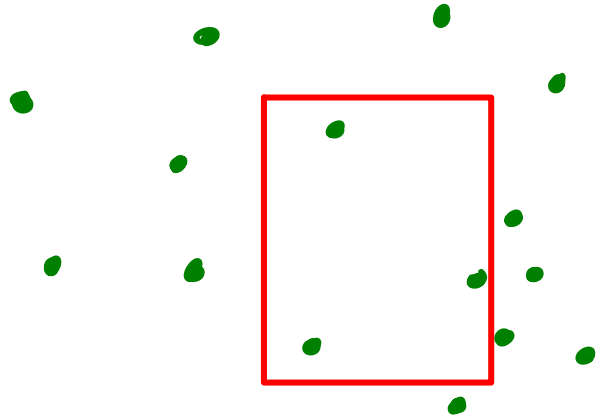




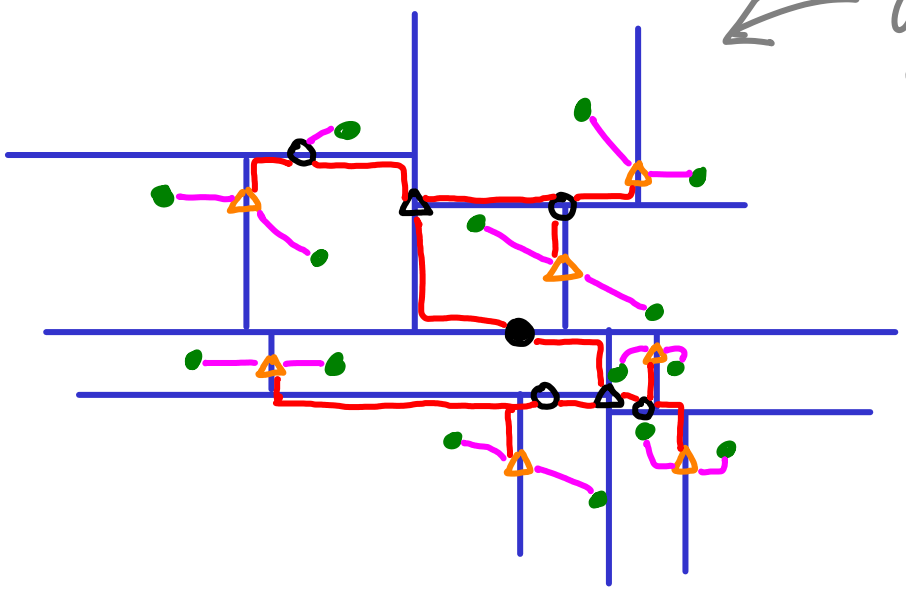
... until every cell has 1 point

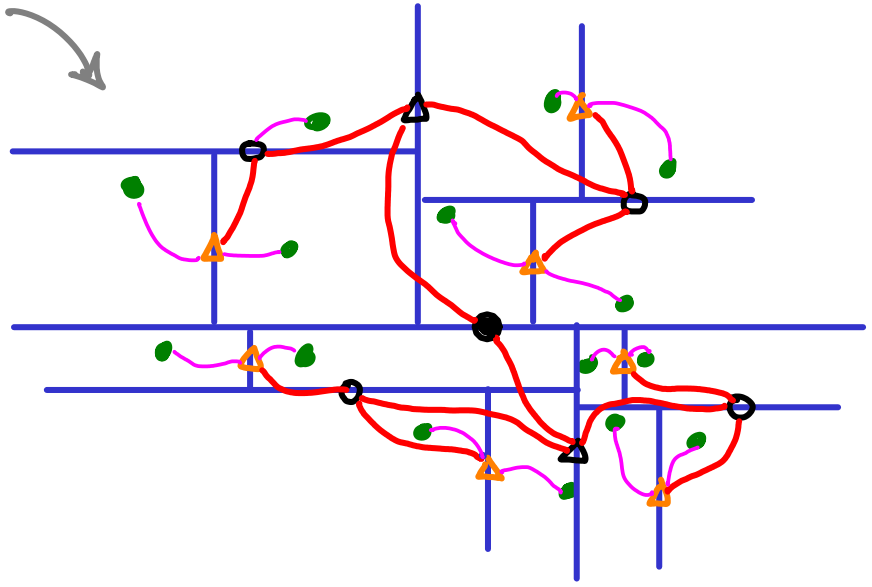


2D :



graph drawing

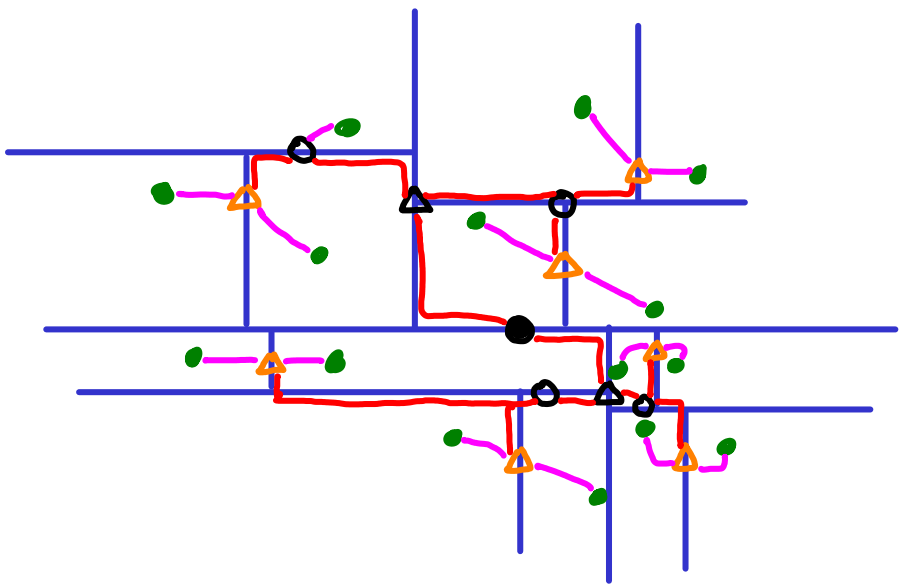


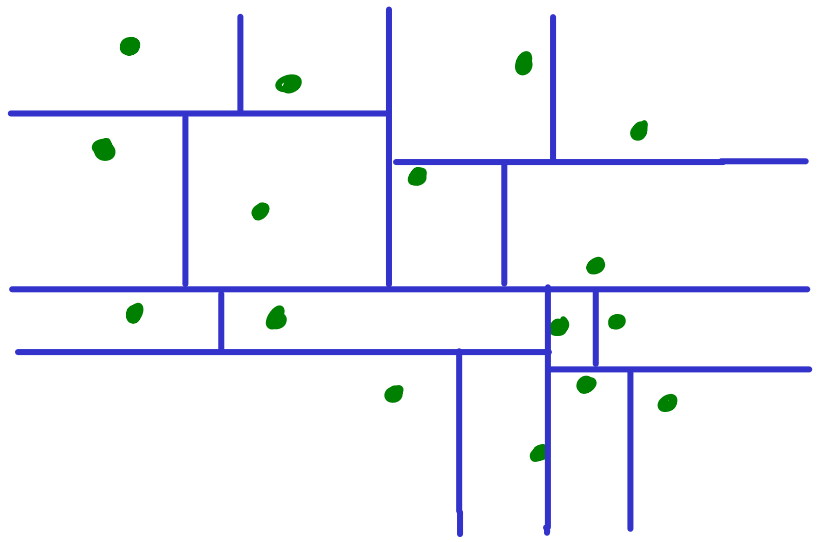


Construction time:

- per level: disjoint median-finding : $\Theta(n)$ total
- $\Theta(\log n)$ levels

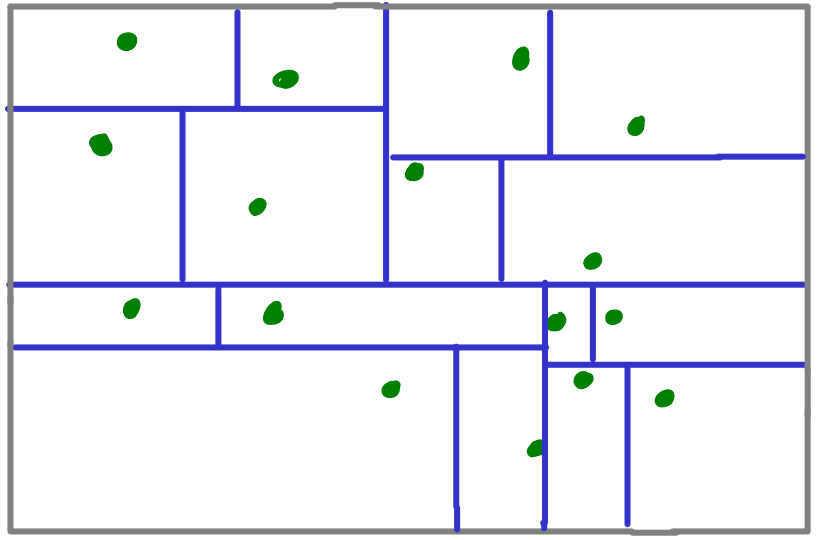
$O(n \log n)$

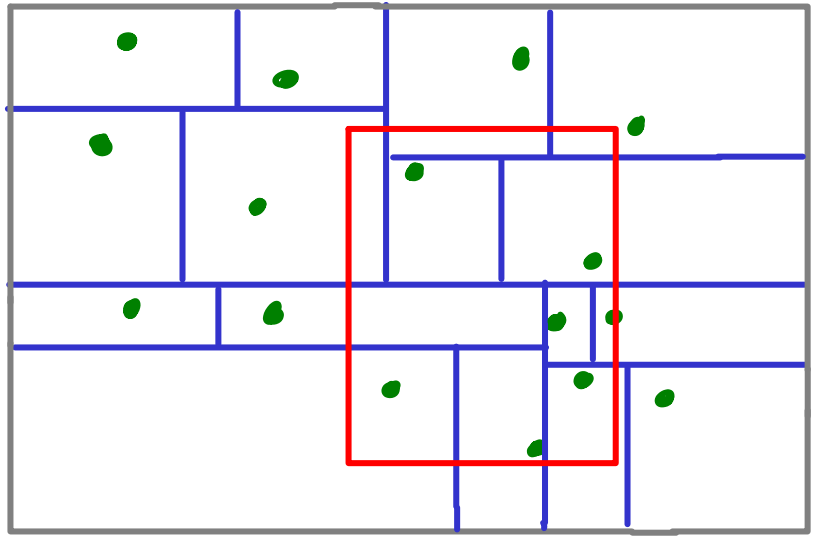




add bounding box :

every leaf is a rectangle



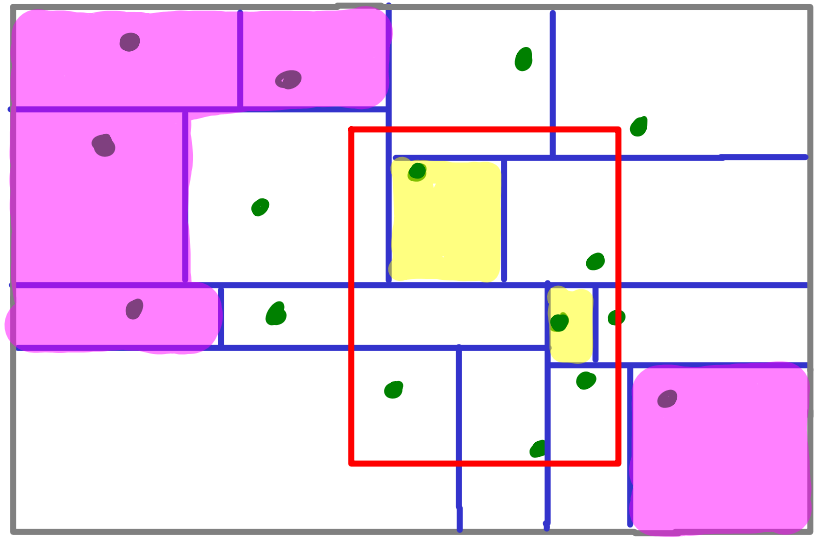


Range-count query:

brute force: for every leaf (rectangle),
check if data point is inside query box

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing

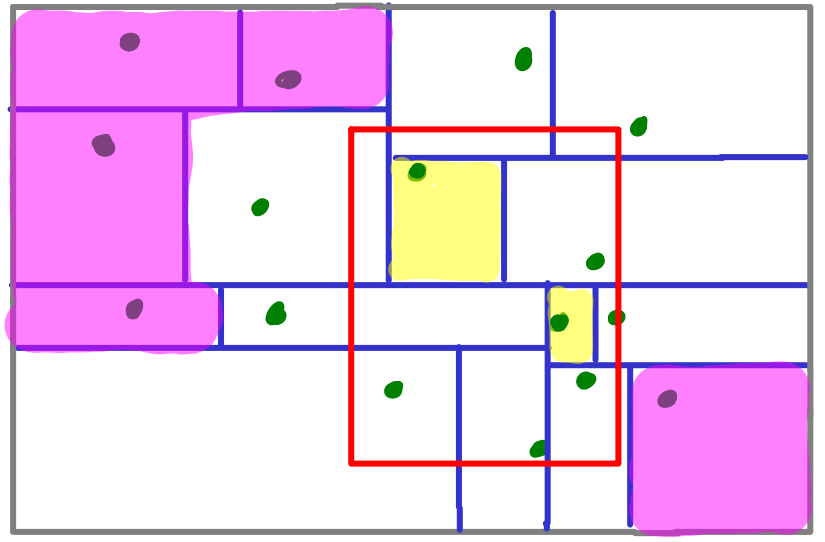


Range-count query:

brute force: for every leaf (rectangle),
check if data point is inside query box

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing



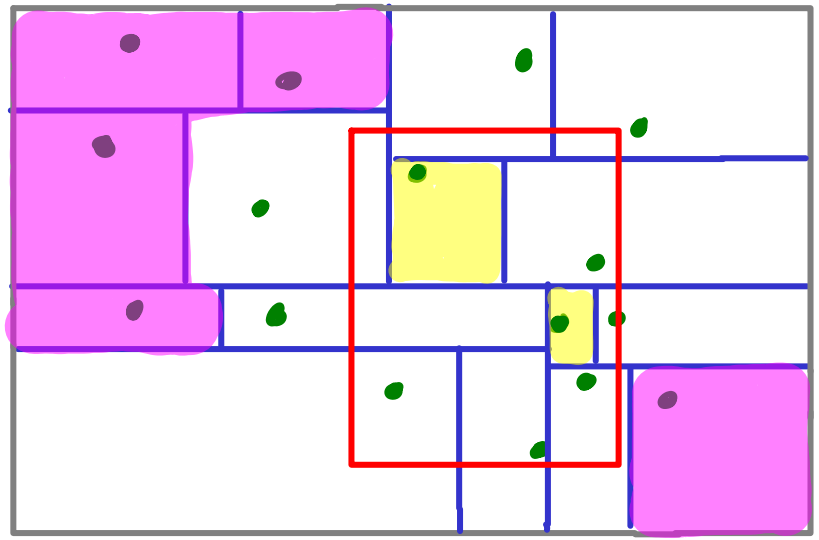
Range-count query:

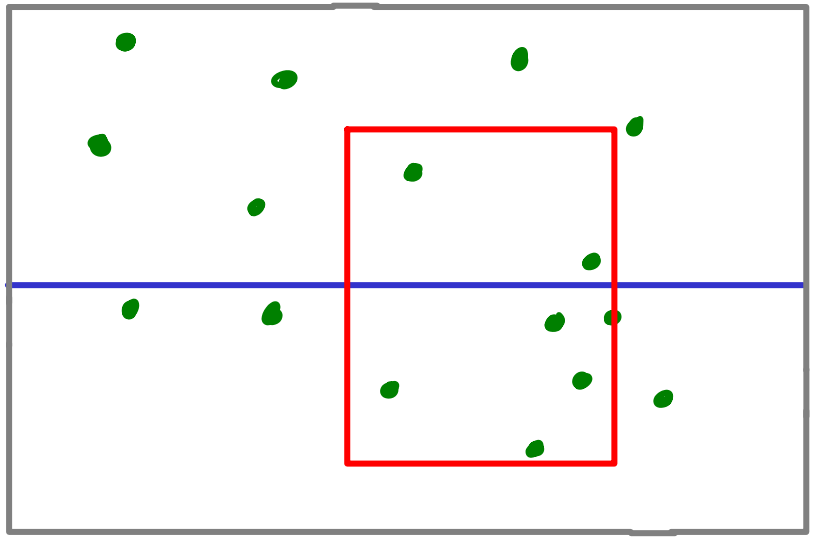
brute force: for every leaf (rectangle),
check if data point is inside query box

We will only check "crossing" leaves explicitly

3 TYPES OF LEAVES (RECTANGLES)

- inside □
- outside □
- crossing □

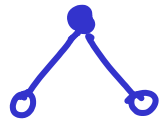




ALGO

step1 : compare □ to the 2 rectangles of 1st partition

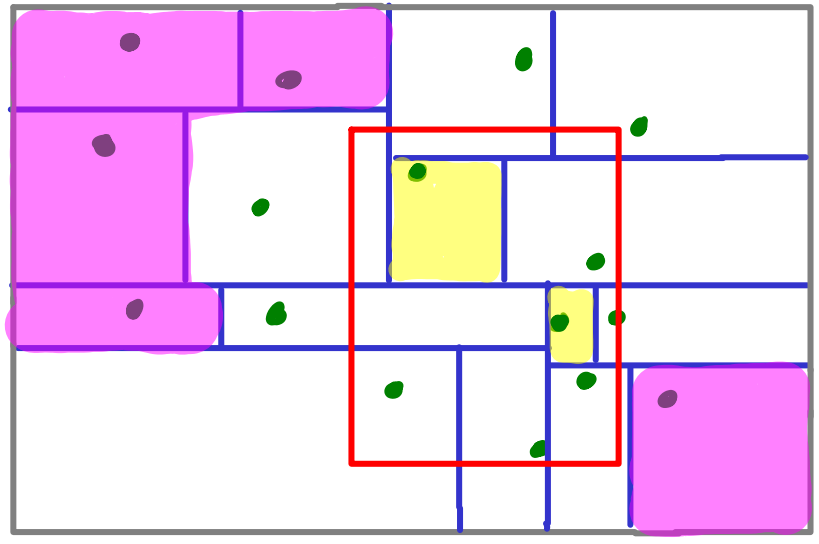
proceed on whichever side intersects □

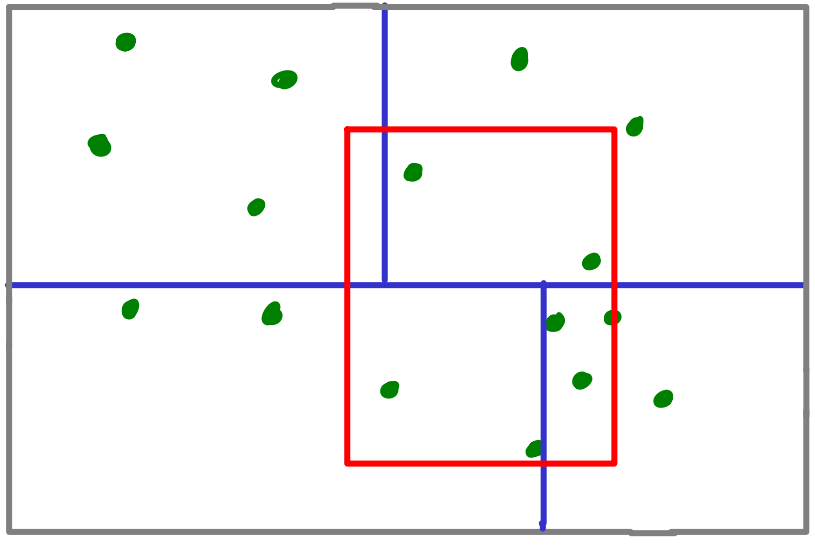


(in this case, both)

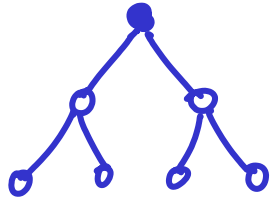
3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing



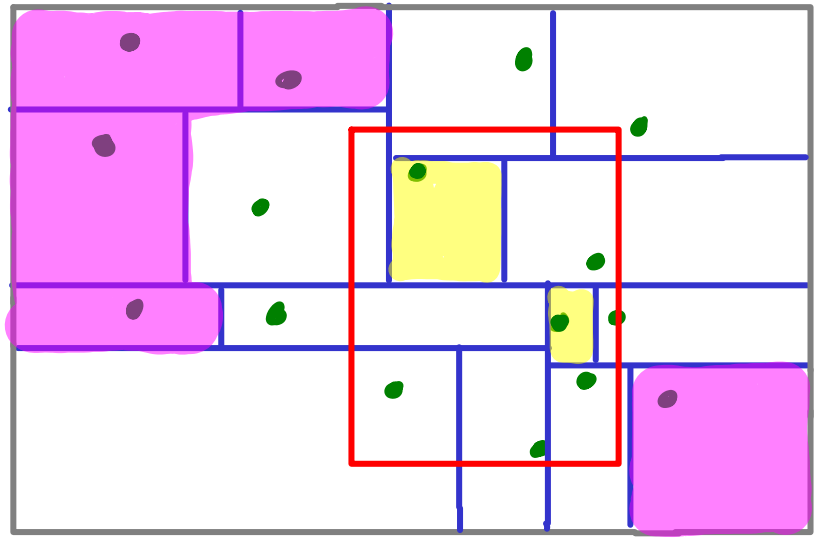


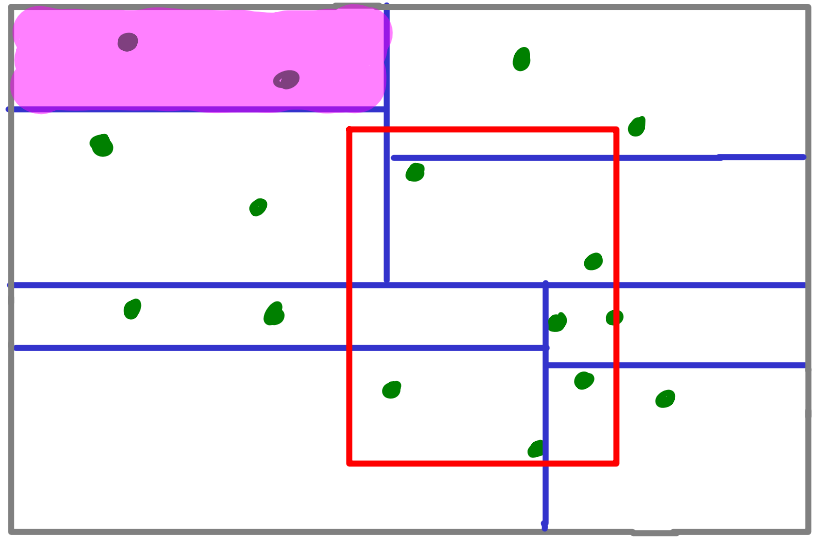
step 2 : compare □ to 4 rectangles (children)



3 TYPES OF LEAVES (RECTANGLES)

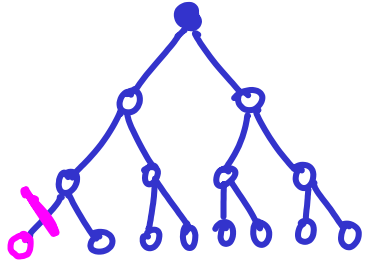
- inside
- outside
- crossing





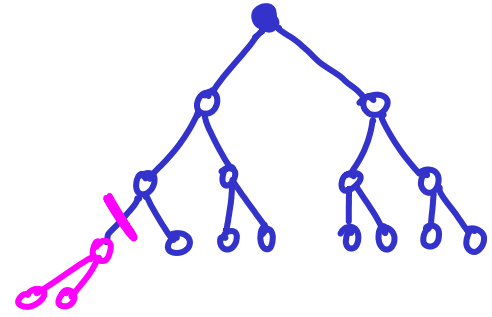
step 3 : finally one rectangle doesn't overlap.

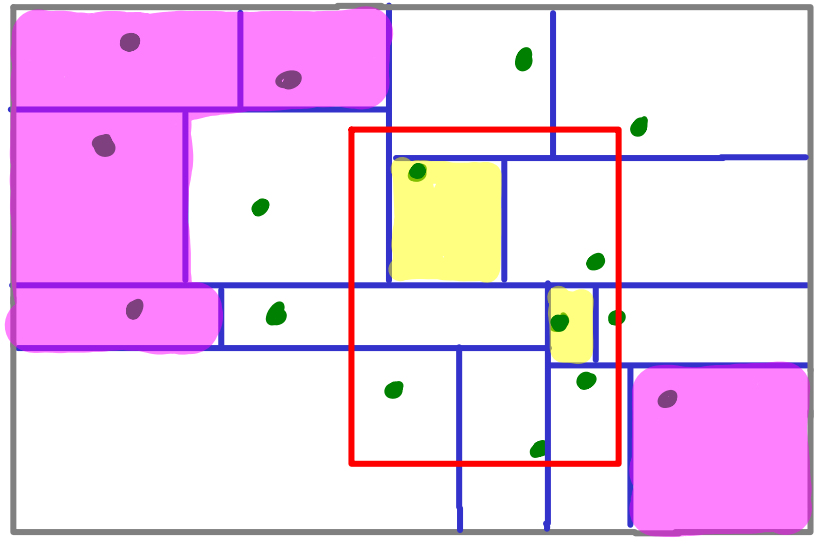
Ignore it and its children

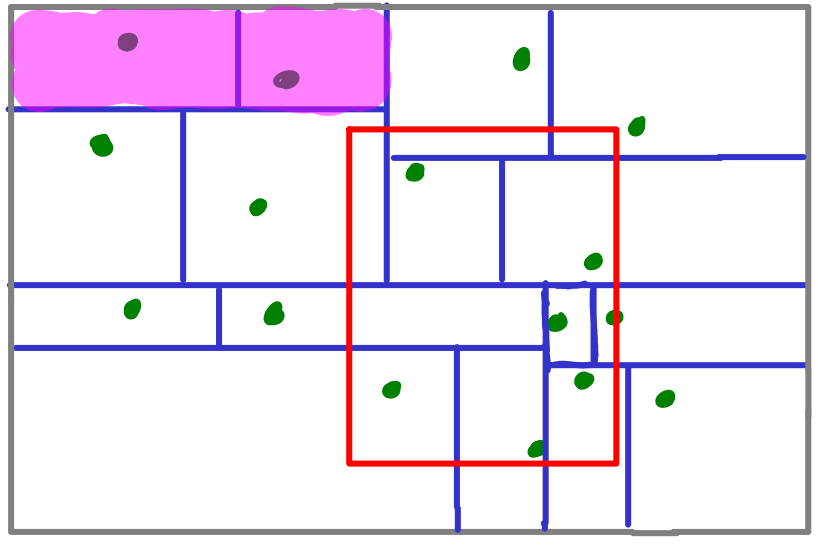


3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing





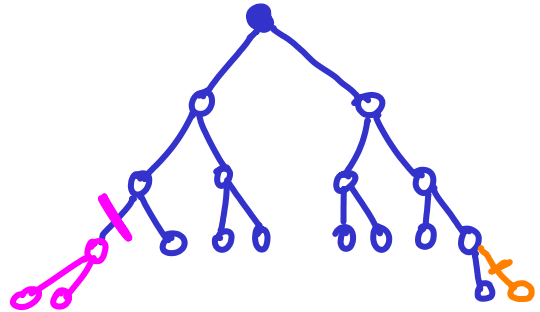


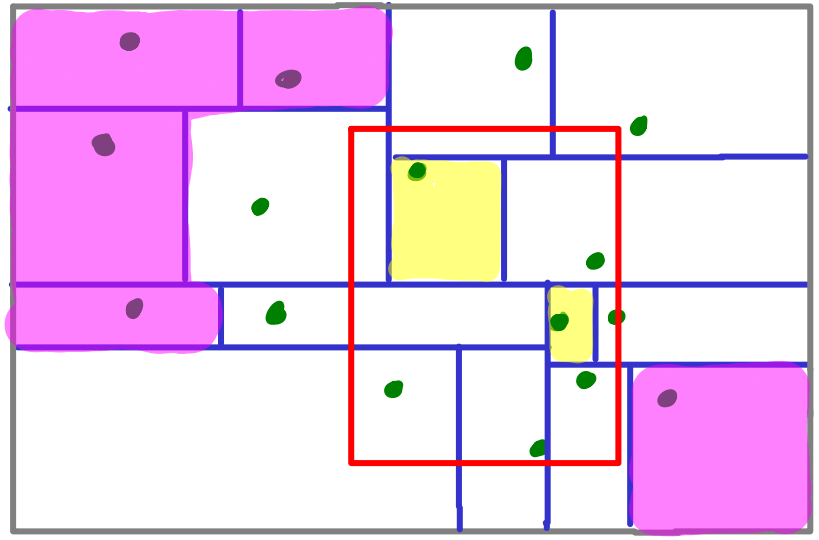
Level 4:

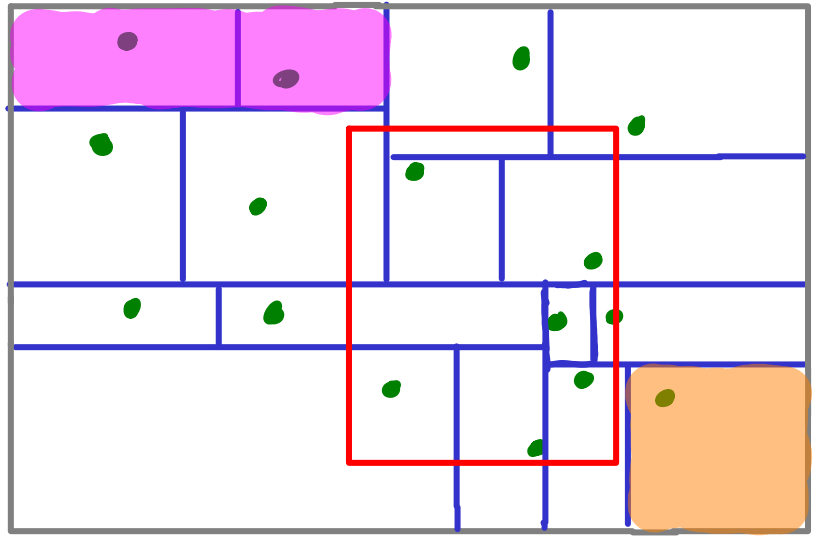
- no comparison of □ to ■

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing





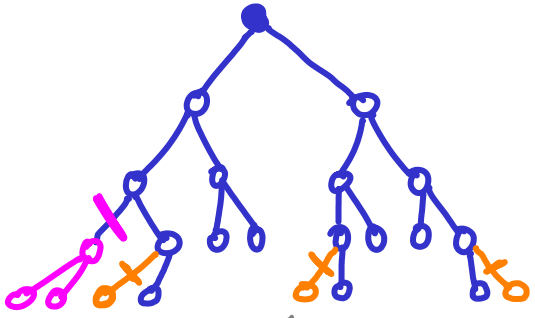


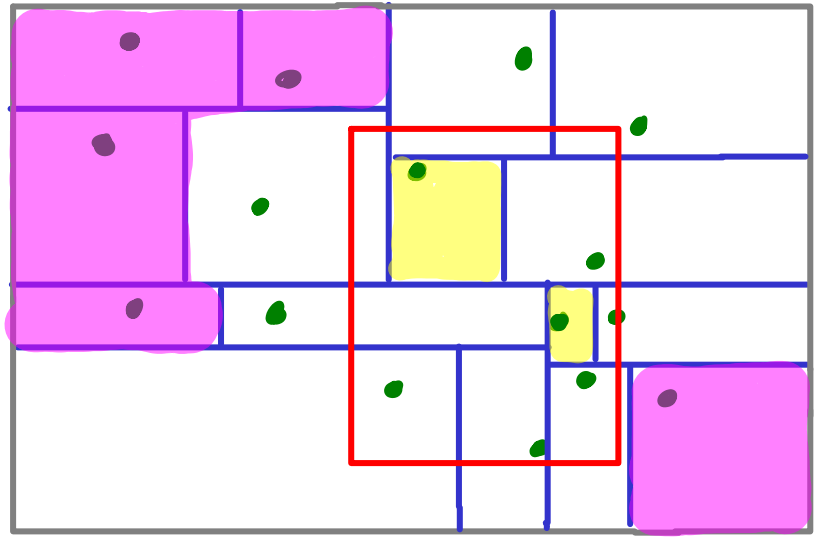
Level 4:

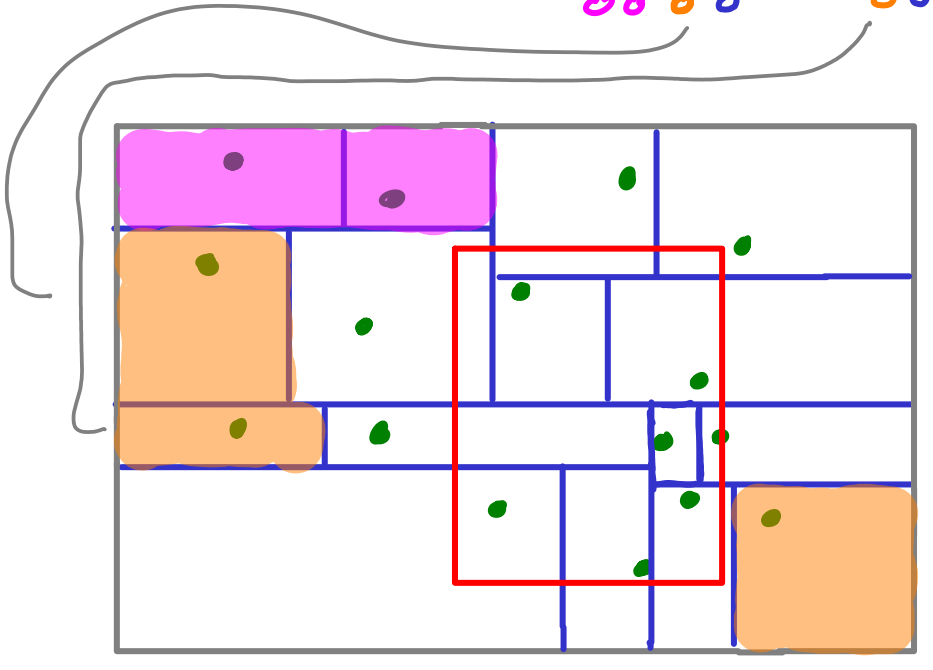
- no comparison of □ to ■
- ignore new external boxes: ■

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing





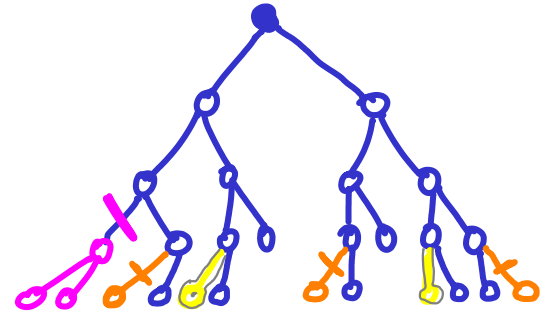


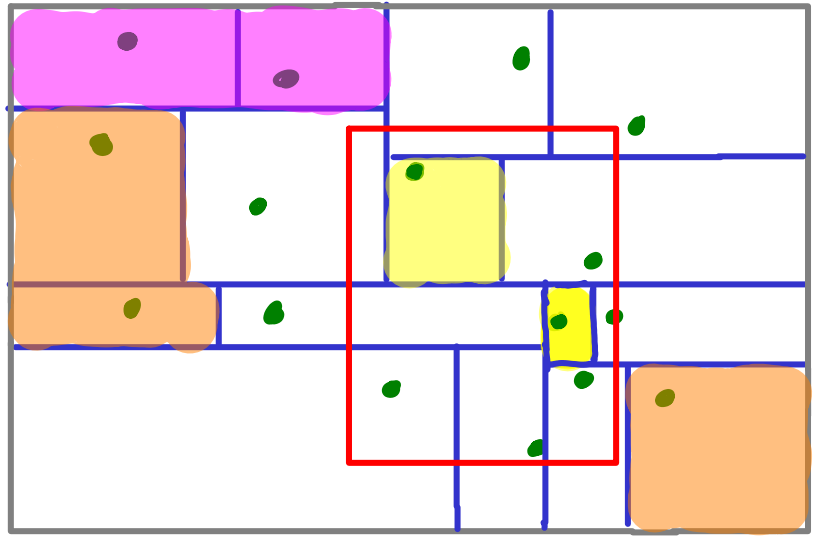
Level 4:

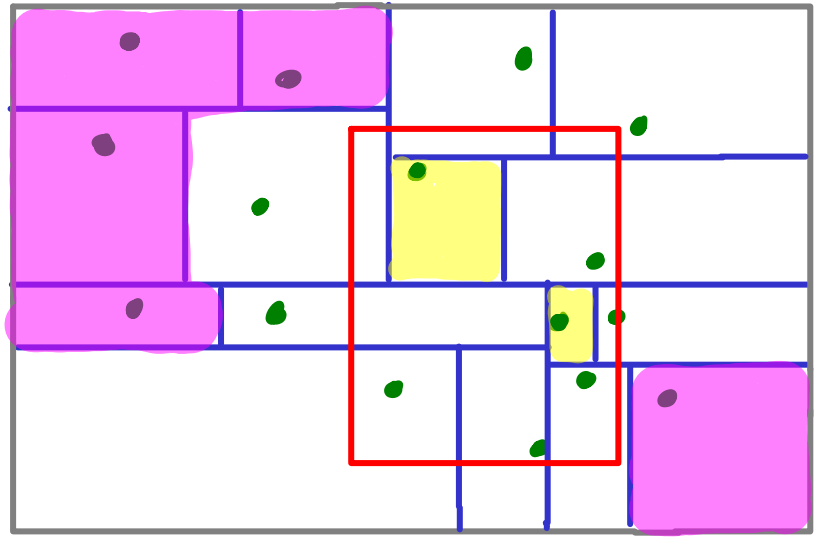
- no comparison of to
- ignore new external boxes:

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing







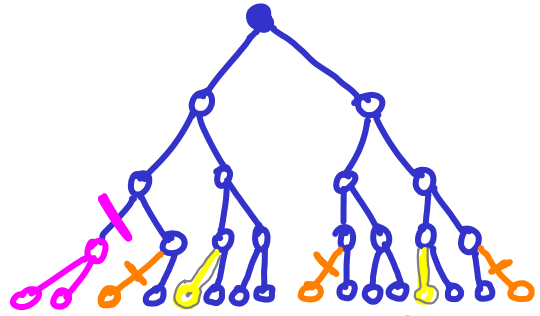
Level 4:

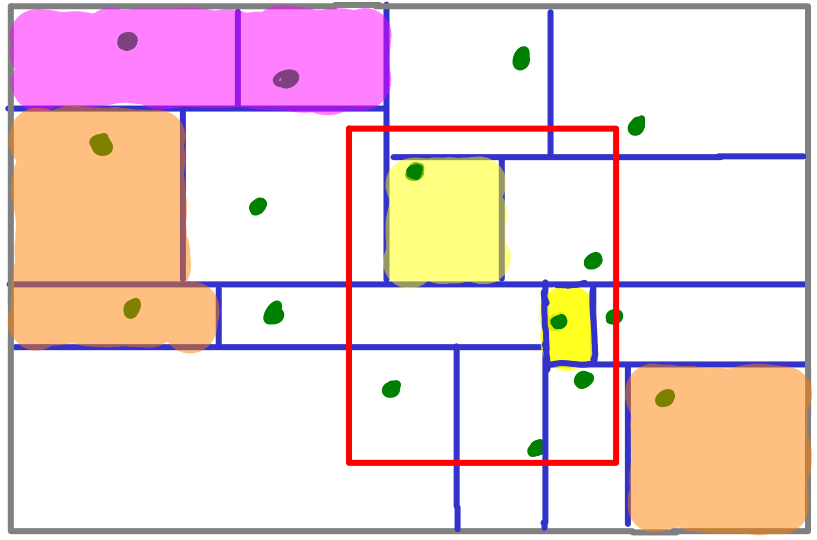
- no comparison of to
- ignore new external boxes:
- found internal

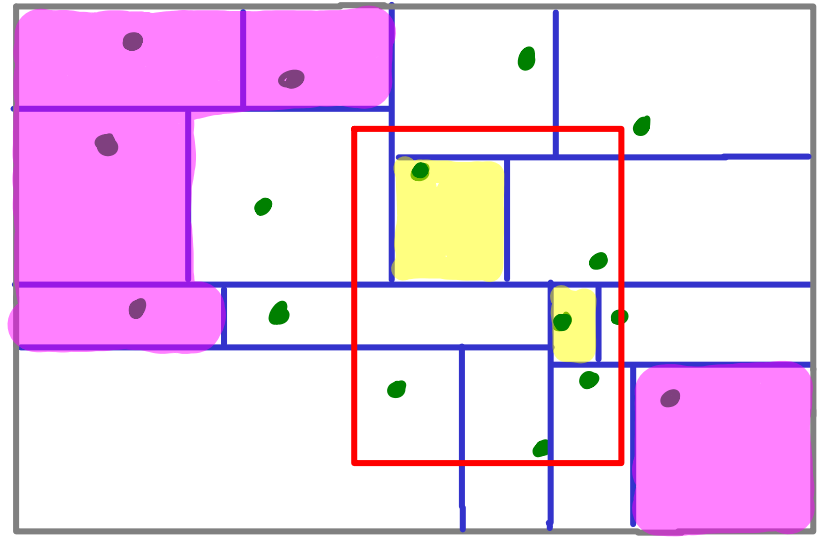
↳ count total points inside & ignore levels below

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing







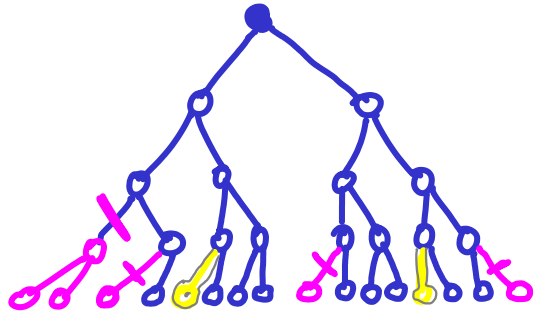
Level 4:

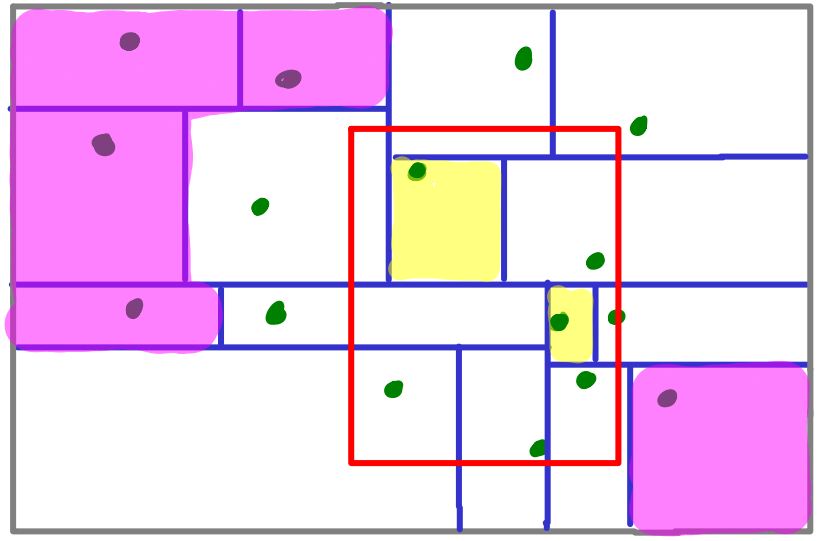
- no comparison of to
- ignore new external boxes:
- found internal

↳ count total points inside & ignore levels below

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing



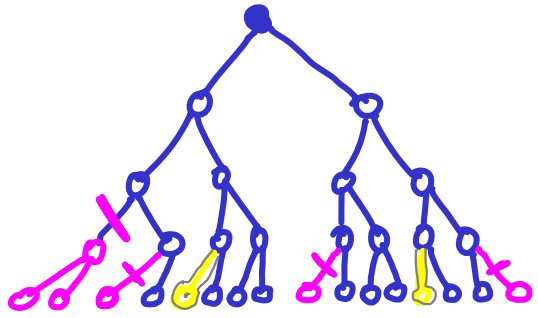


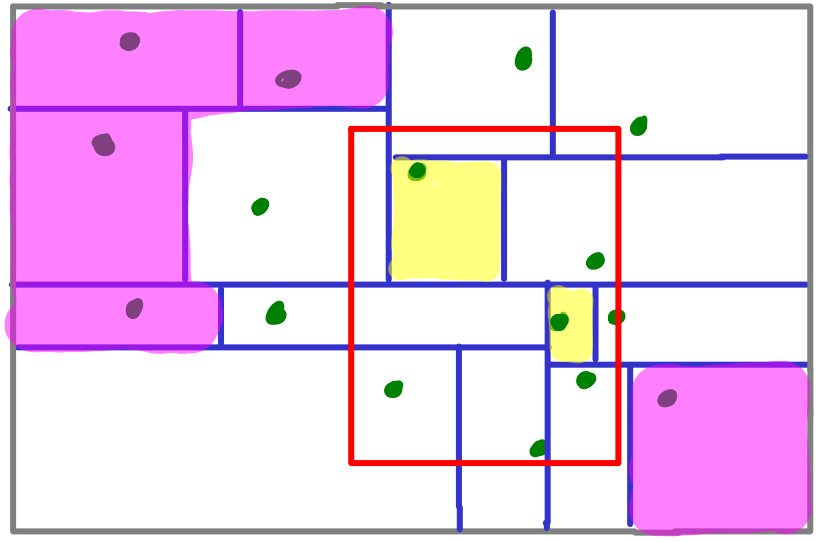
Similar to 1D, we implicitly count points in fully contained subtrees
& ignore points in external subtrees

(rectangle ~ subtree)

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing



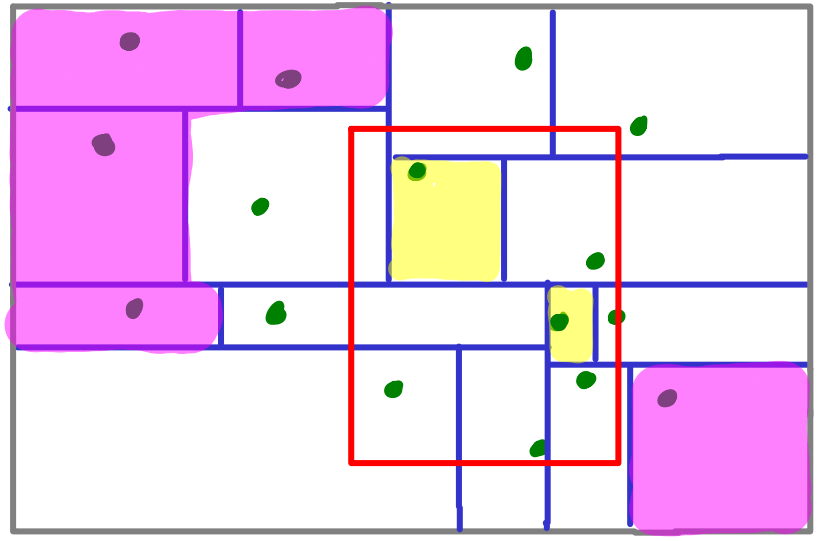


Similar to 1D, we implicitly count points in fully contained subtrees & ignore points in external subtrees

Unlike 1D, we can branch a lot

3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing



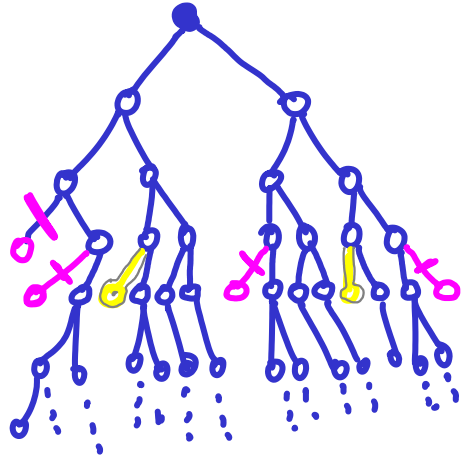
Dominating factor in worst case work: # crossing leaves

CLAIM

Other leaves are either

- in an ignored rectangle
- already counted (ancestor)

represented by nodes adjacent to paths to crossing leaves



3 TYPES OF LEAVES (RECTANGLES)

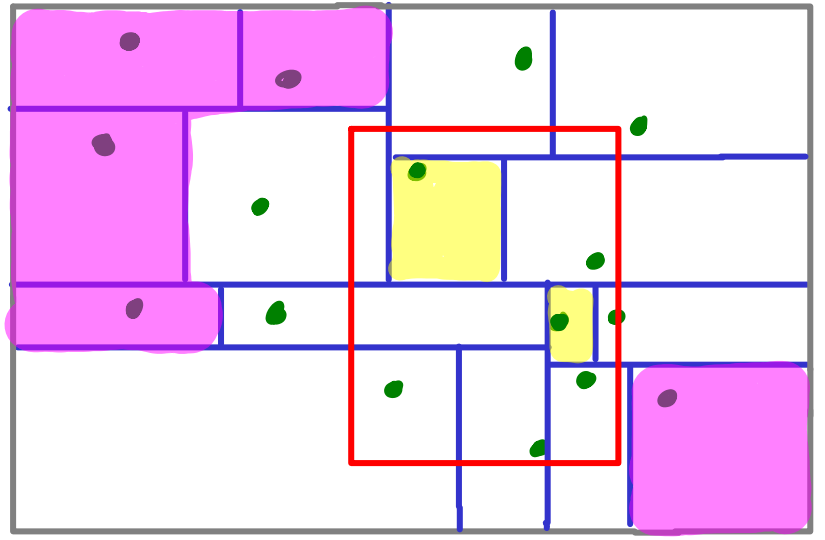
- inside □
- outside □
- crossing □

Dominating factor in worst case work: # crossing leaves

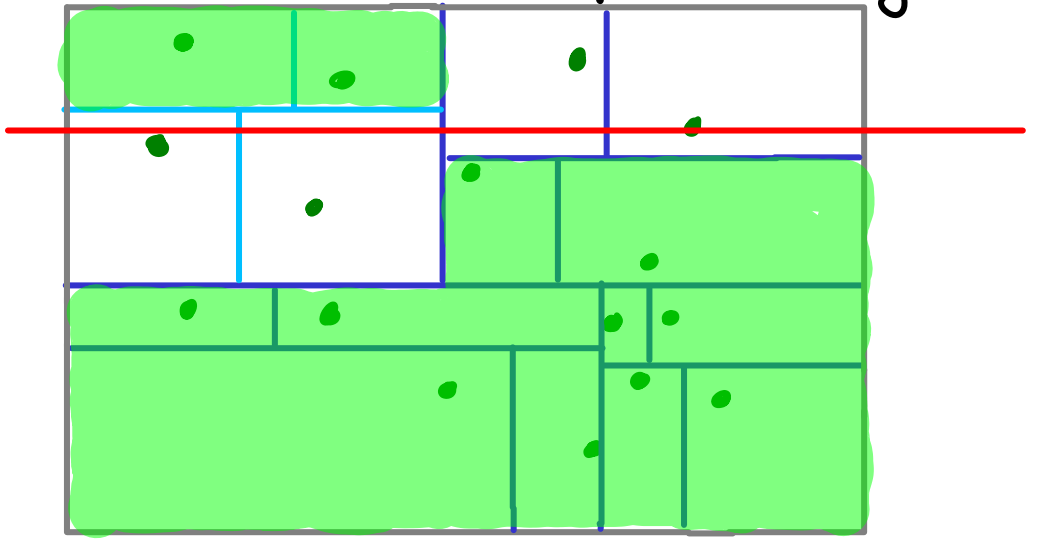
Other leaves are either

- in an ignored rectangle
- already counted (ancestor)

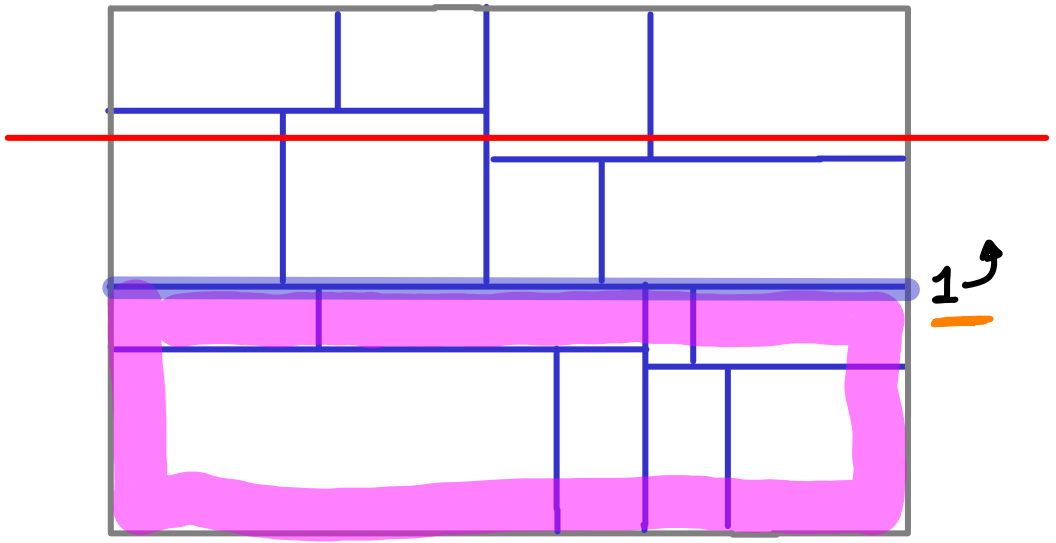
represented by nodes adjacent to paths to crossing leaves



CROSSINGS $\leq 4 \times$ (max # rectangles stabbed by a line)

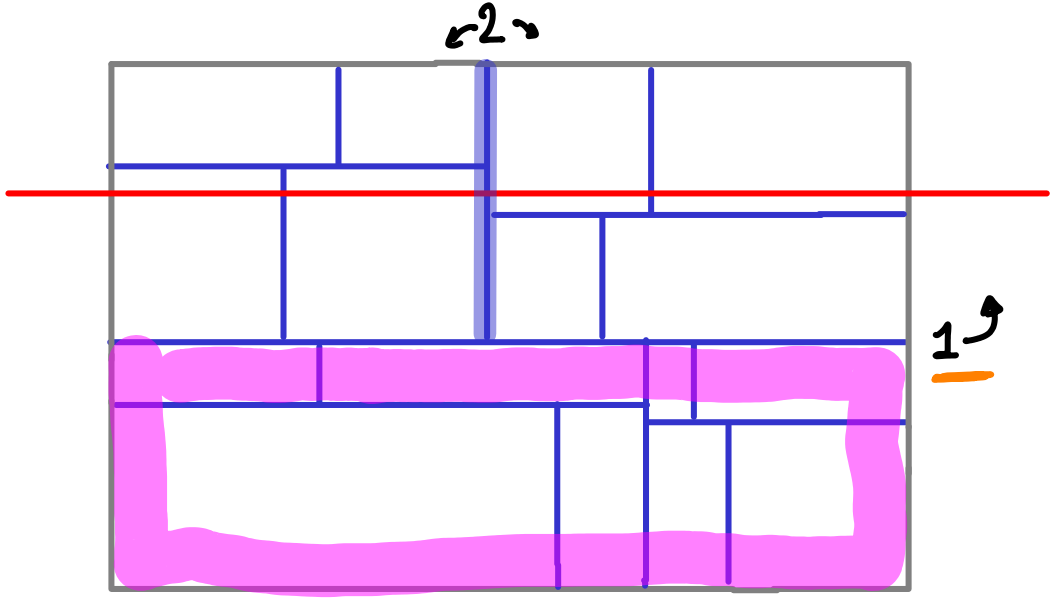


2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



Every horizontal node has one subtree untouched

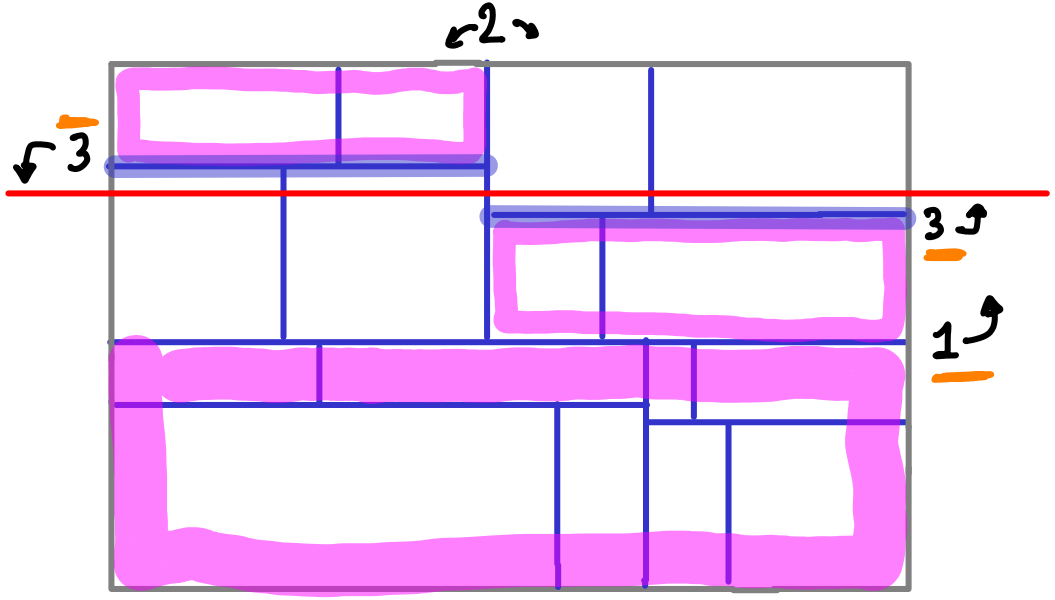
2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



Every horizontal node has one subtree untouched

Every vertical node:
must visit both children

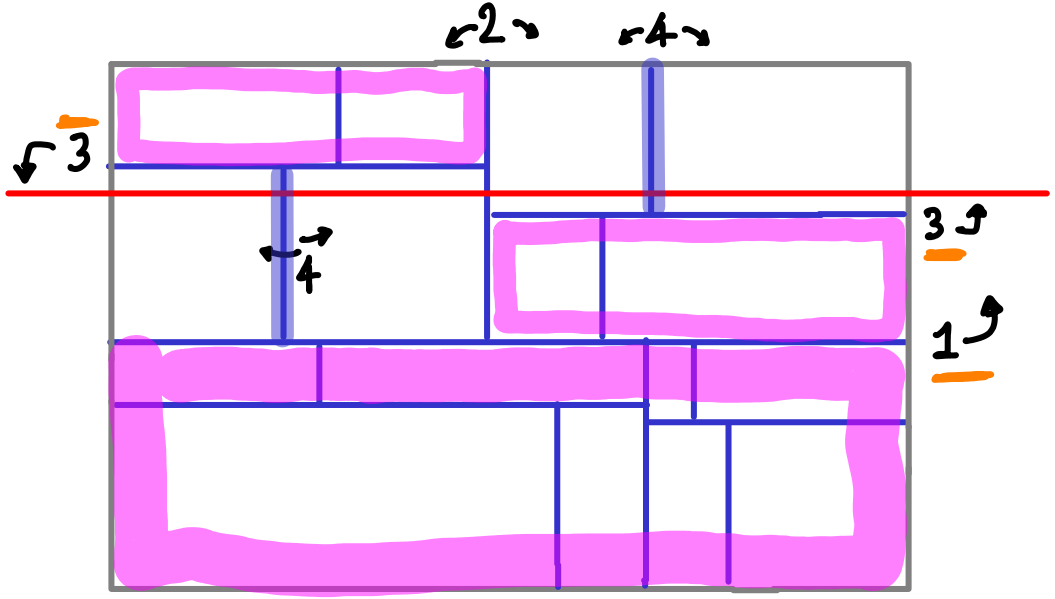
2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



Every horizontal node has one subtree untouched

Every vertical node:
must visit both children

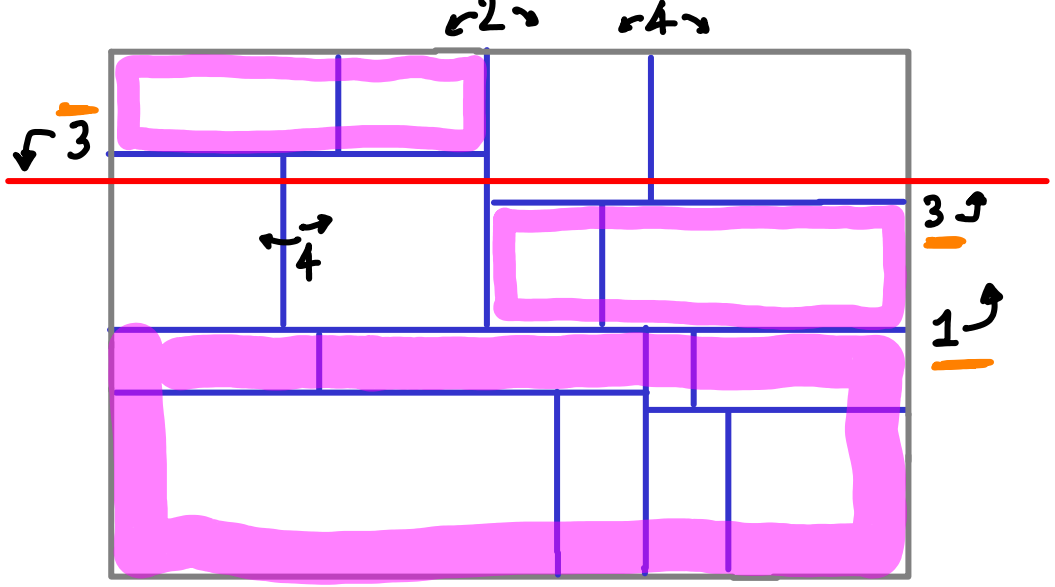
2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



Every horizontal node has one subtree untouched

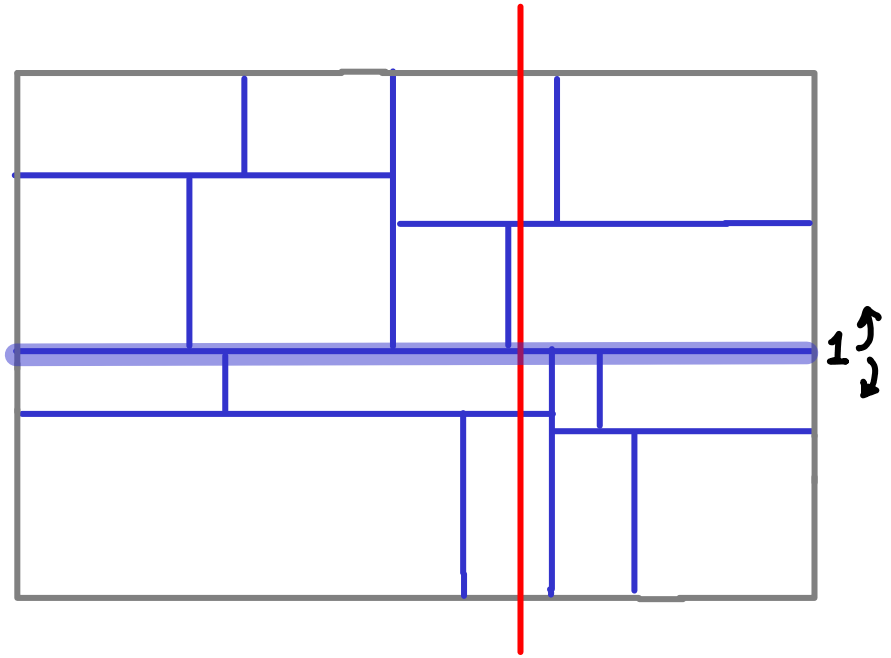
Every vertical node:
must visit both children

2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



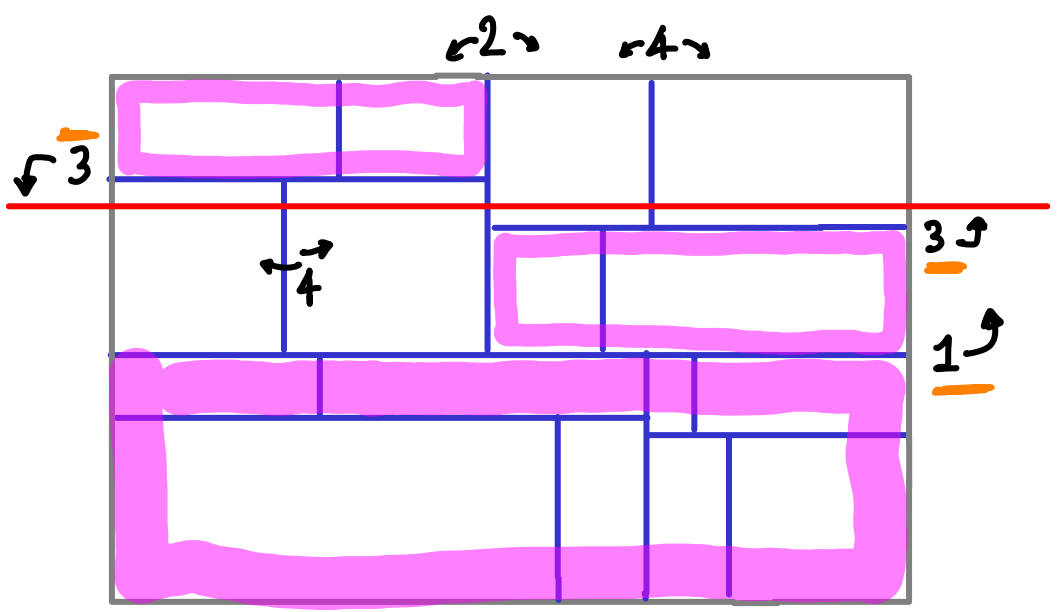
Every horizontal node has one subtree untouched

Every vertical node:
must visit both children



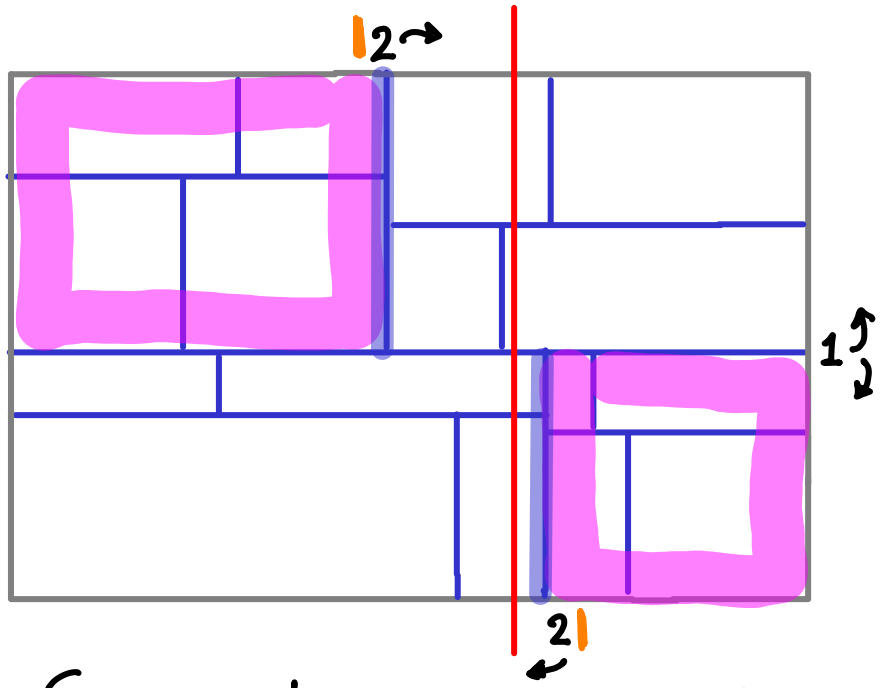
for vertical stabbing line,
opposite: h-nodes ↔ v-nodes

2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



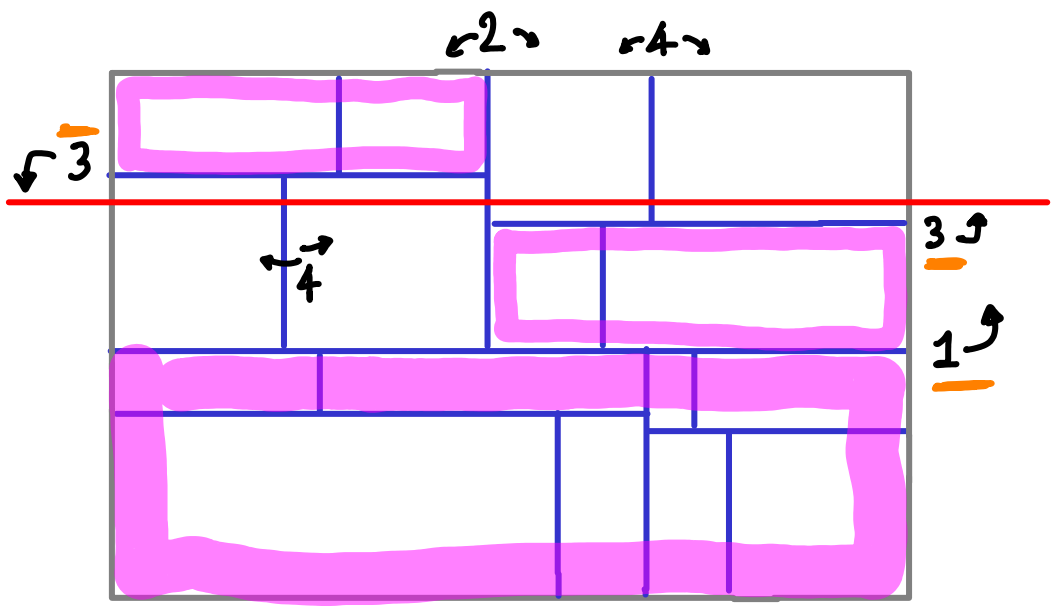
Every horizontal node has one subtree untouched

Every vertical node:
must visit both children



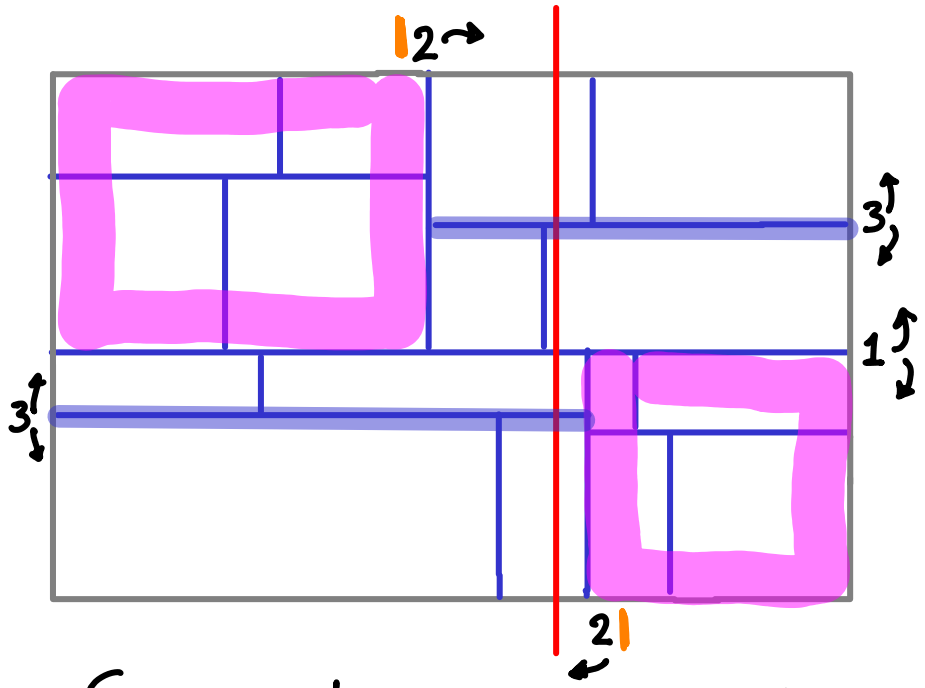
for vertical stabbing line,
opposite: h-nodes ↔ v-nodes

2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.



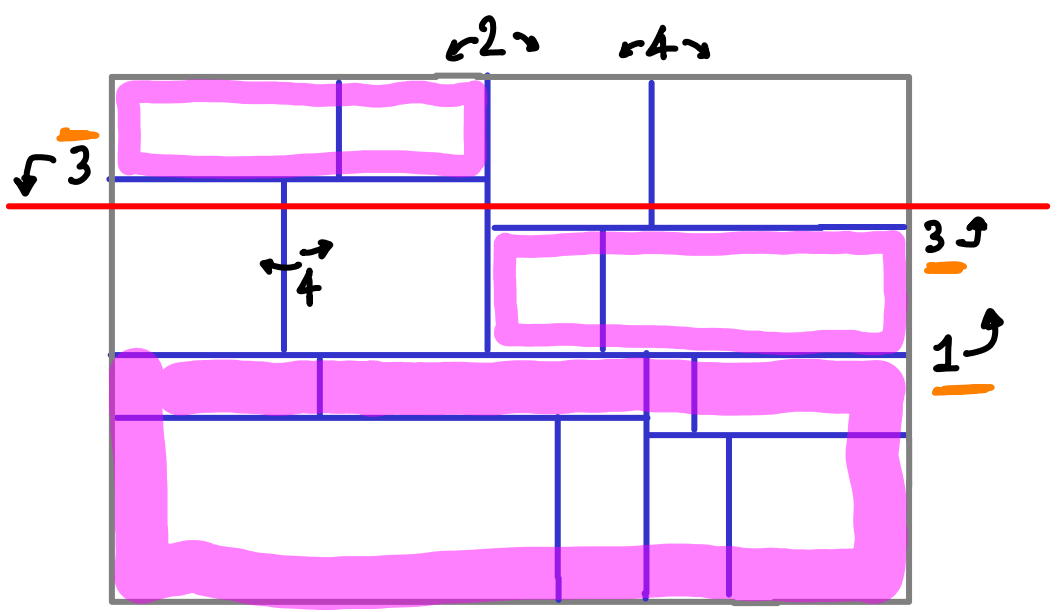
Every horizontal node has one subtree untouched

Every vertical node:
must visit both children



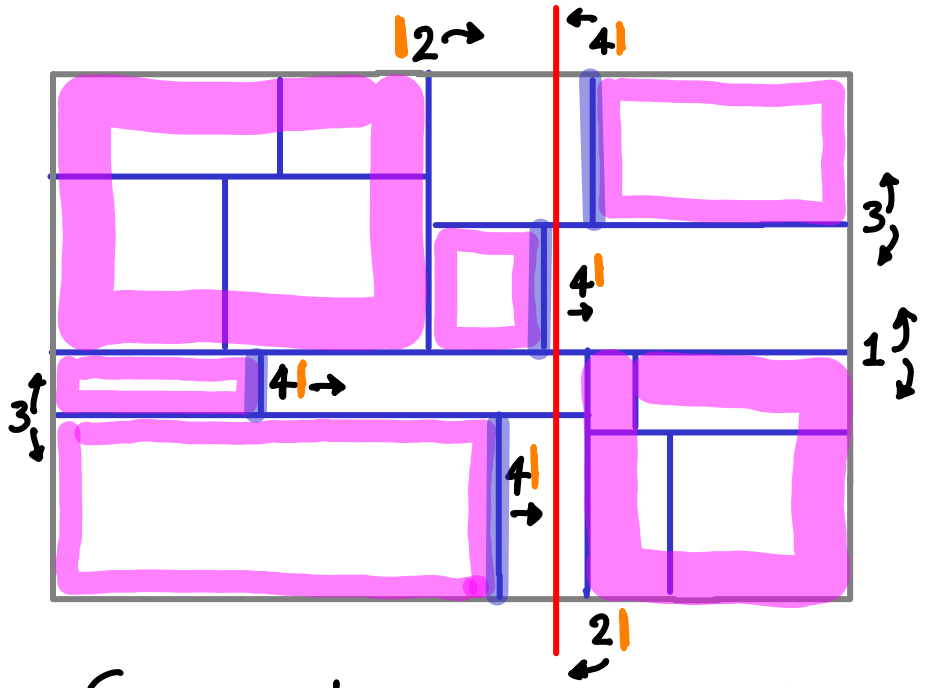
for vertical stabbing line,
opposite: h-nodes ↔ v-nodes

2 EXAMPLES of STABBING a 2-d TREE w/ a LINE.

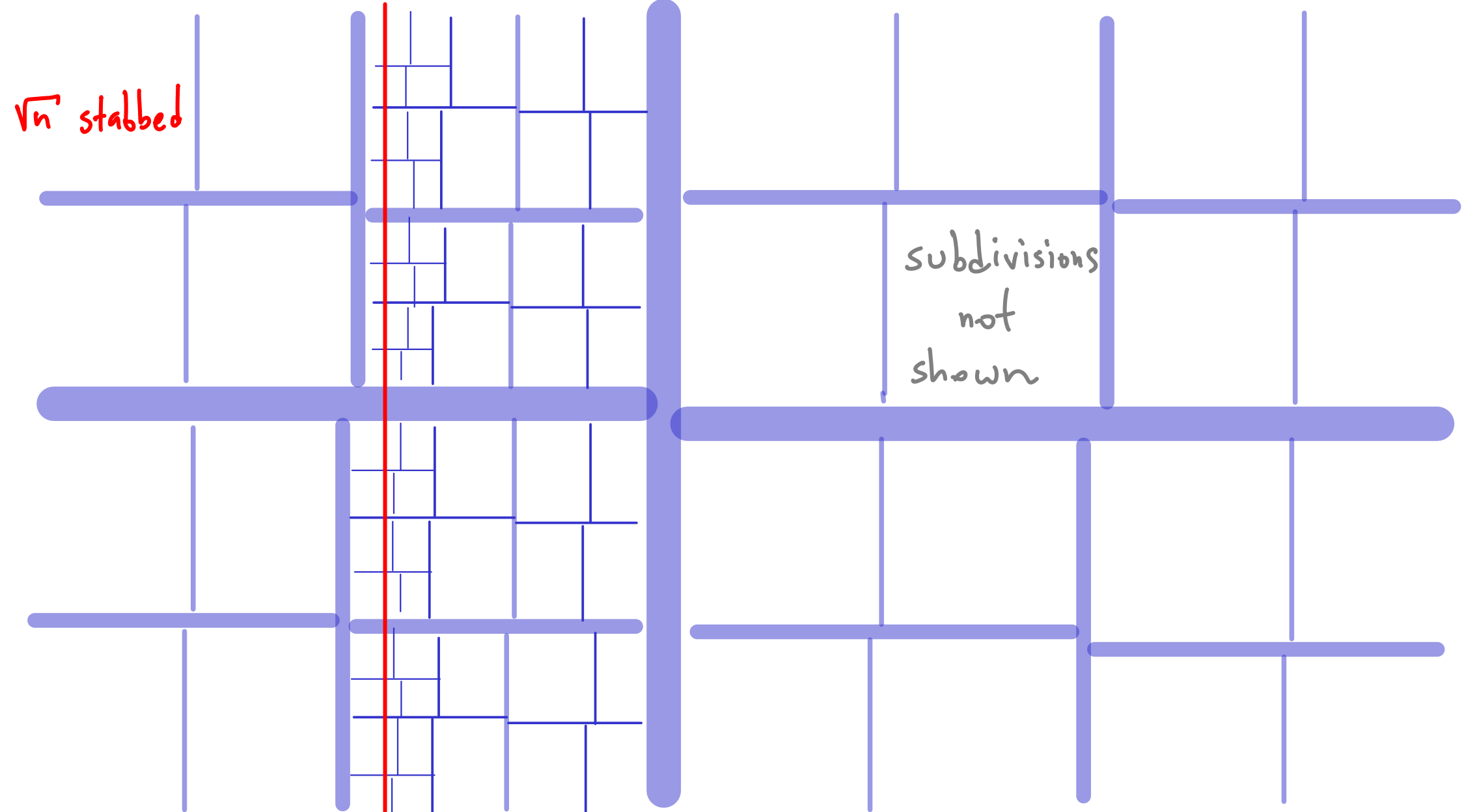


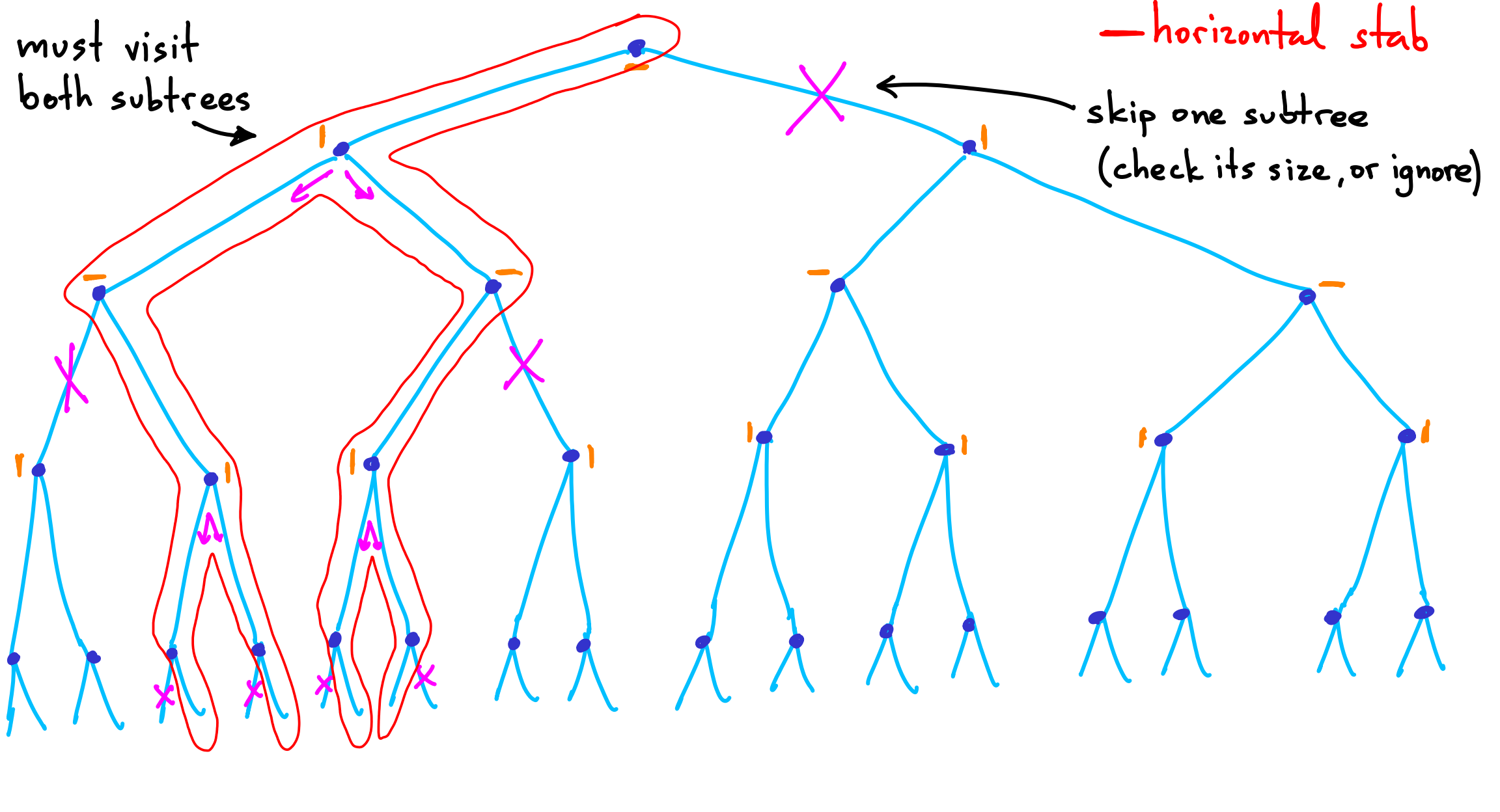
Every horizontal node has one subtree untouched

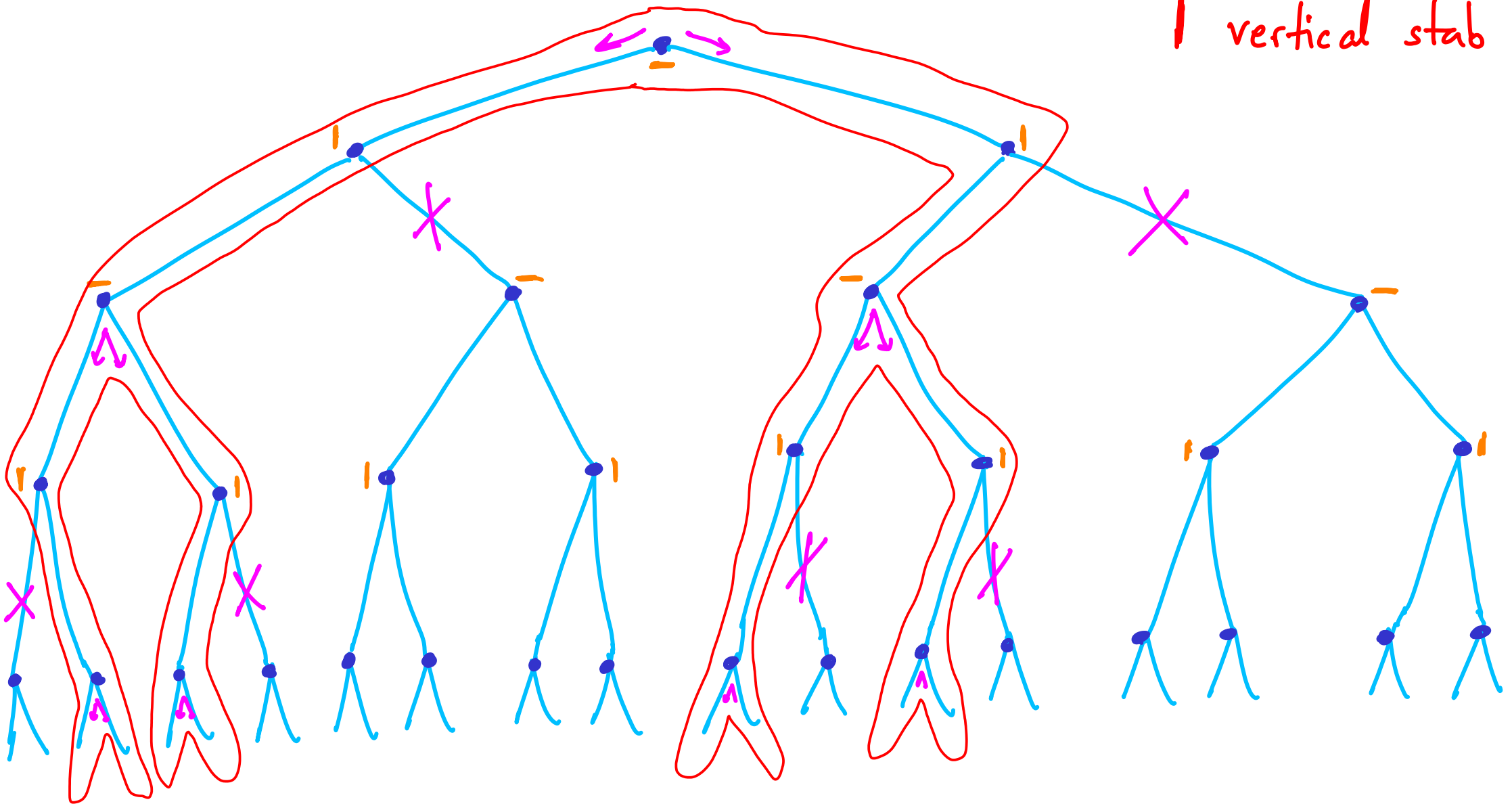
Every vertical node:
must visit both children



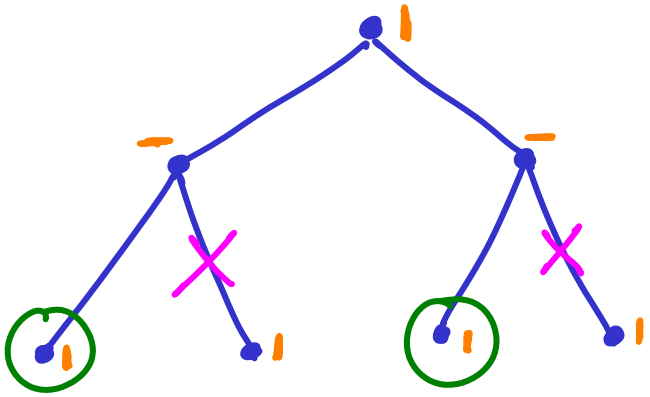
for vertical stabbing line,
opposite: h-nodes ↔ v-nodes

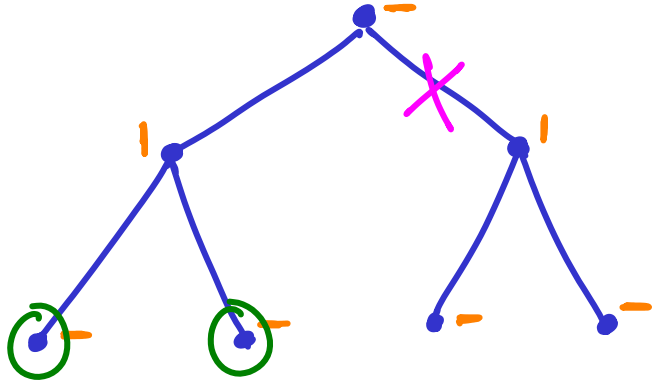

√n stabbed
subdivisions
not
shown


must visit
both subtrees
skip one subtree
(check its size, or ignore)
— horizontal stab


1 vertical stab

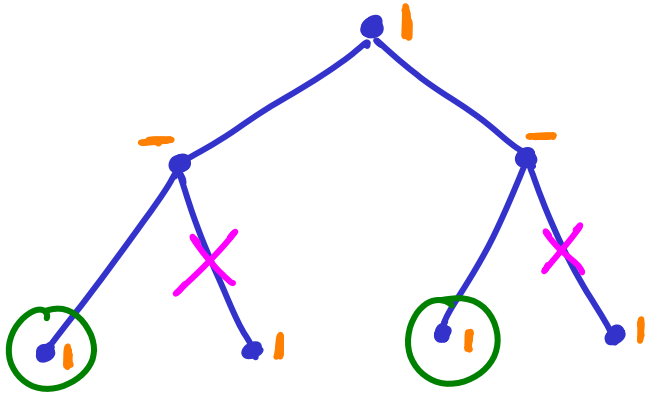
wlog

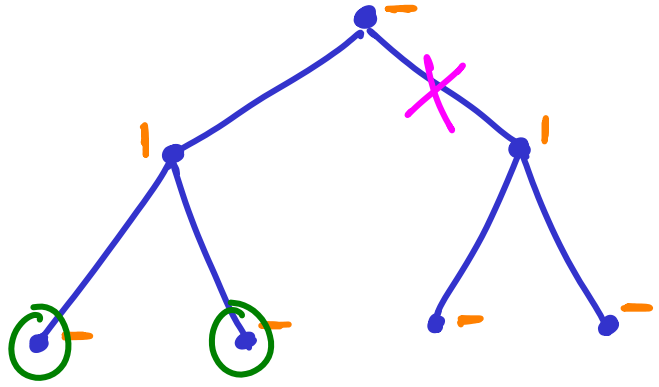




Never visit more than 2 grandchildren of a visited node.

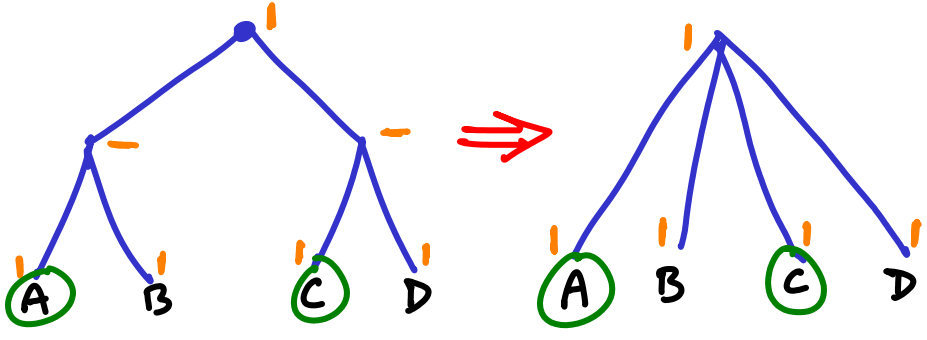
wlog

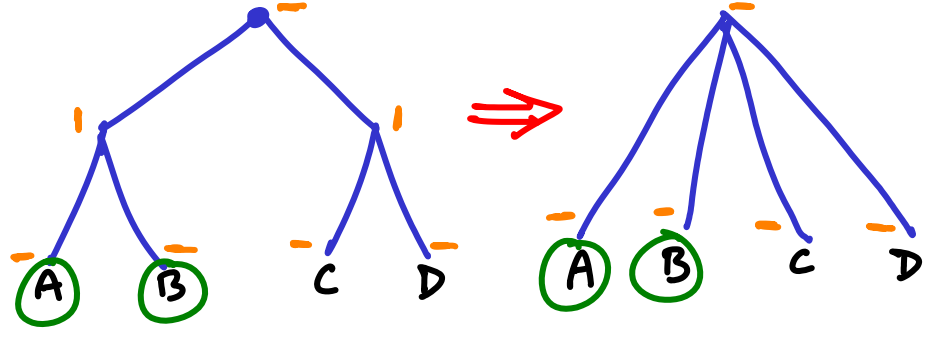


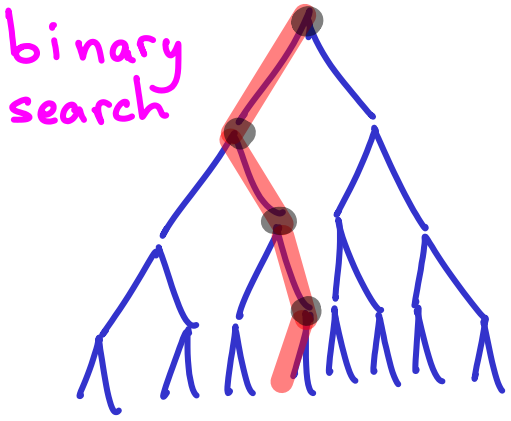


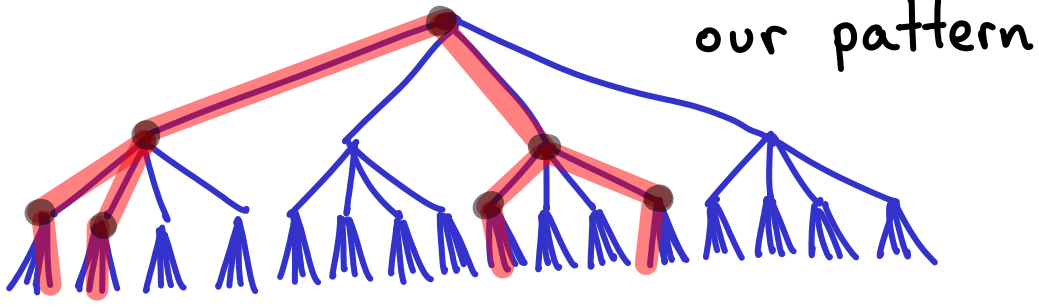
Never visit more than 2 grandchildren of a visited node.

Compress every other level in tree : explore 2 of 4 subtrees/children

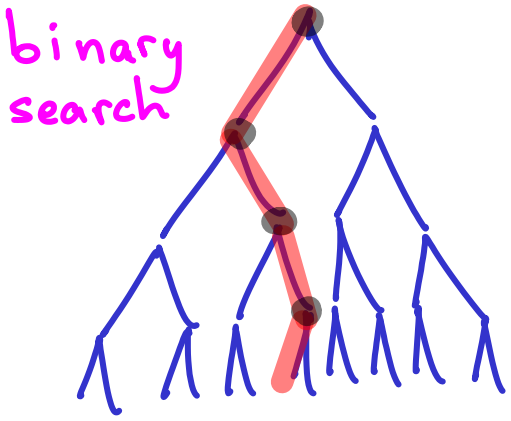


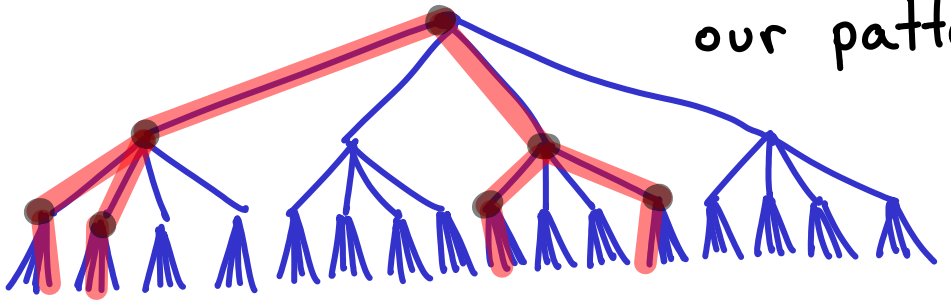






- Both trees : explore ½ of subtrees, O(1) work per node





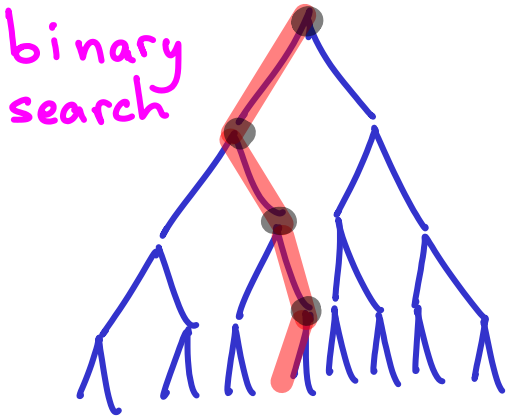
- Both trees : explore ½ of subtrees, $O(1)$ work per node

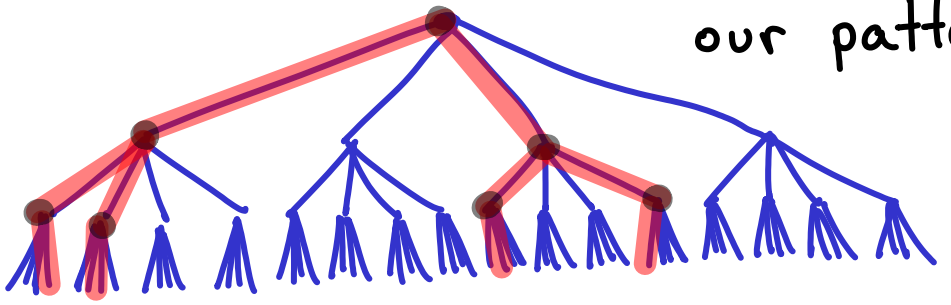
binary search

leaves visited $S(n) \leq 1 \cdot S(\frac{n}{2})$

total work $S(n) \leq 1 \cdot S(\frac{n}{2}) + O(1)$

$O(\log n)$

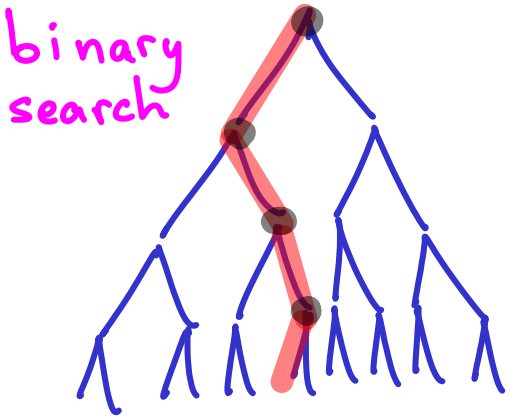


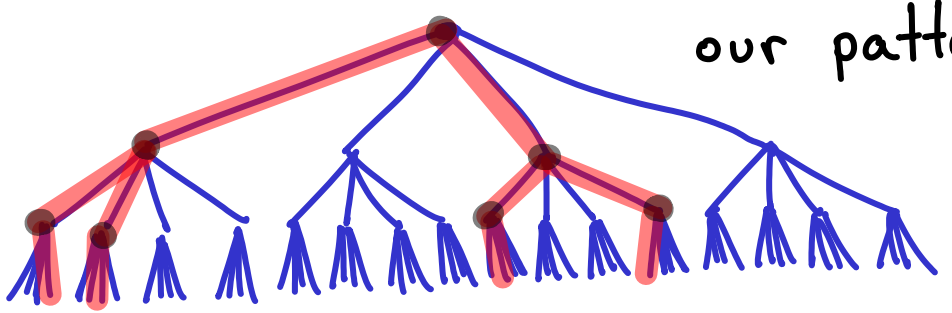


- Both trees : explore ½ of subtrees, $O(1)$ work per node

	binary search	our pattern
leaves visited	$S(n) \leq 1 \cdot S(\frac{n}{2})$	$S(n) \leq 2 \cdot S(\frac{n}{4})$
total work	$S(n) \leq 1 \cdot S(\frac{n}{2}) + O(1)$	$S(n) \leq 2 \cdot S(\frac{n}{4}) + O(1)$

$O(\log n)$





- Both trees : explore ½ of subtrees, $O(1)$ work per node

	binary search	our pattern
leaves visited	$S(n) \leq 1 \cdot S(\frac{n}{2})$	$S(n) \leq 2 \cdot S(\frac{n}{4})$
total work	$S(n) \leq 1 \cdot S(\frac{n}{2}) + O(1)$	$S(n) \leq 2 \cdot S(\frac{n}{4}) + O(1)$

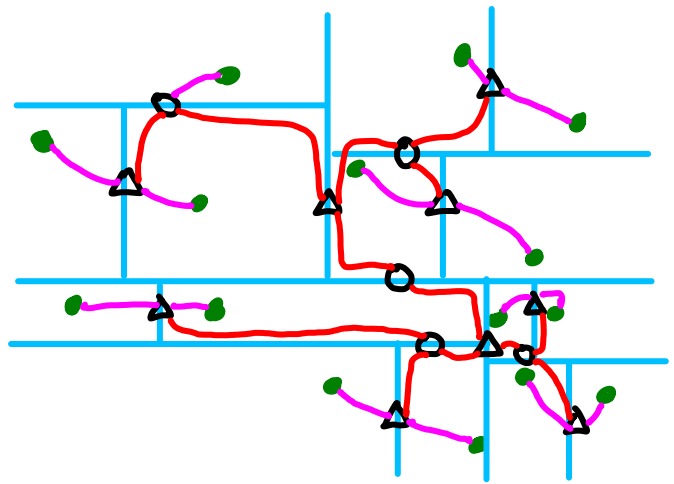
$O(\log n)$

Master method: $n^{\log_4 2} = \underline{\Theta(\sqrt{n})}$

leaf level dominates work

CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

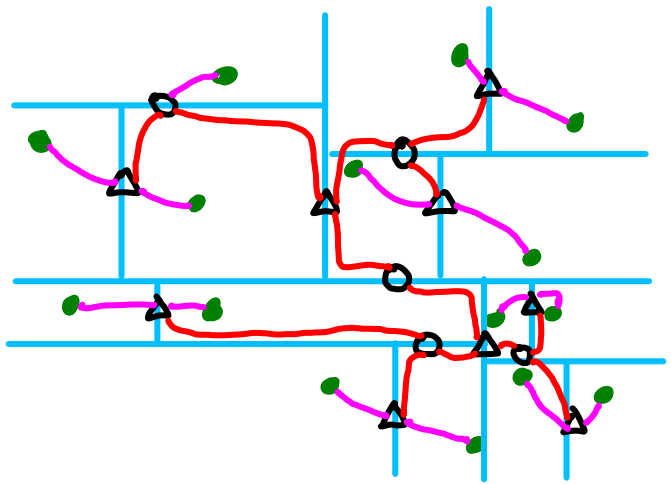


CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

---

kd: #leaves: $S(n) \leq 2^{k-1} \cdot S\left(\frac{n}{2^k}\right)$

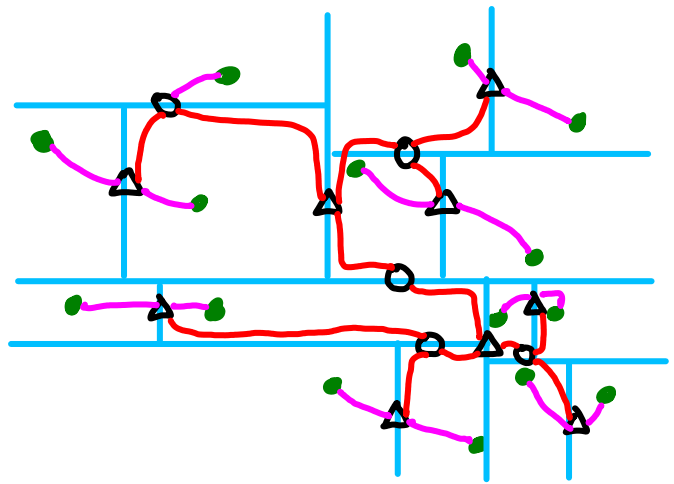


CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

---

kd: #leaves: $S(n) \leq 2^{k-1} \cdot S\left(\frac{n}{2^k}\right) = O\left(n^{\log_{2^k} 2^{k-1}}\right) = O\left(n^{\frac{k-1}{k}}\right) = O\left(n^{1-\frac{1}{k}}\right)$



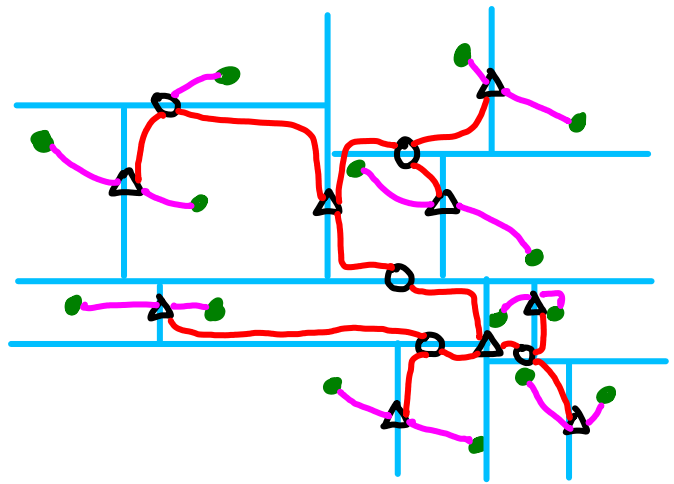
CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

---

kd: #leaves: $S(n) \leq 2^{k-1} \cdot S\left(\frac{n}{2^k}\right) = O\left(n^{\log_{2^k} 2^{k-1}}\right) = O\left(n^{\frac{k-1}{k}}\right) = O\left(n^{1-\frac{1}{k}}\right)$

but each node in contracted tree hides k levels $= \Theta(k)$ work



CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

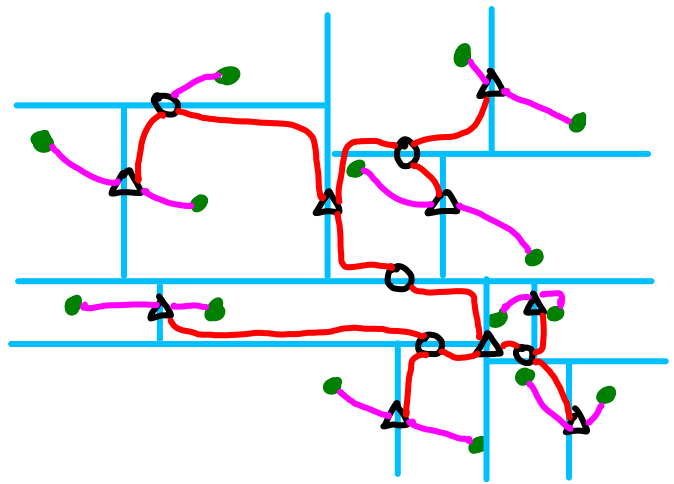
- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

---

kd: #leaves: $S(n) \leq 2^{k-1} \cdot S\left(\frac{n}{2^k}\right) = O(n^{\log_{2^k} 2^{k-1}}) = O\left(n^{\frac{k-1}{k}}\right) = O(n^{1-\frac{1}{k}})$

but each node in contracted tree hides k levels $= \Theta(k)$ work

Query $= O(k \cdot n^{1-1/k})$



CONCLUSION: any query intersects $O(\sqrt{n})$ edges of a 2D kd tree

- 2D range counting : $O(\sqrt{n})$ time
- Reporting: $O(R + \sqrt{n})$

---

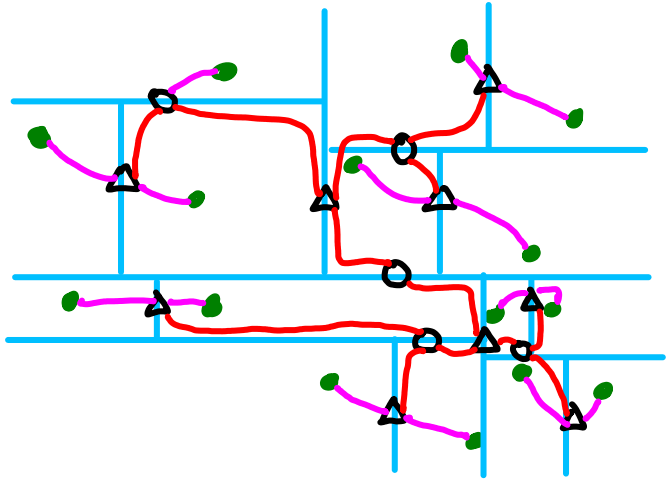
kd: #leaves: $S(n) \leq 2^{k-1} \cdot S\left(\frac{n}{2^k}\right) = O\left(n^{\log_{2^k} 2^{k-1}}\right) = O\left(n^{\frac{k-1}{k}}\right) = O(n^{1-\frac{1}{k}})$

but each node in contracted tree hides k levels $= \Theta(k)$ work

Query $= O(k \cdot n^{1-1/k})$

DYNAMIC:

- adding a point can affect many levels (medians)
- "complicated" to rebalance, but possible



RANGE COUNTING

	size of structure	query time
$k=2$ k-d tree	$\Theta(n)$	$O(\sqrt{n})$
Tree of trees (see other notes)	$\Theta(n \log n)$	$O(\log n)$

Dimension $k \geq 3$

k-d tree	$\Theta(kn)$	$O(k \cdot n^{1-\frac{1}{k}})$
(tree of)$^{k-1}$ trees	$o(n \log^{k-1} n)$	$o(\log^{k-1} n)$

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