CSE 566 Spring 2023

kd-trees

Instructor: Mingfu Shao

Optimizing the Size

- **Theorem**: let r be the false-positive rate; then the minimized size of the array B is $\underline{n \ln(1/r)/(\ln 2)}^2$.
- **Proof.** $\ln r = t \cdot \ln(1 - e^{-tn/m})$

using $t = \frac{m}{n} \cdot \ln 2 \Rightarrow r = n \cdot \ln(1/r)(\ln 2)^2$.

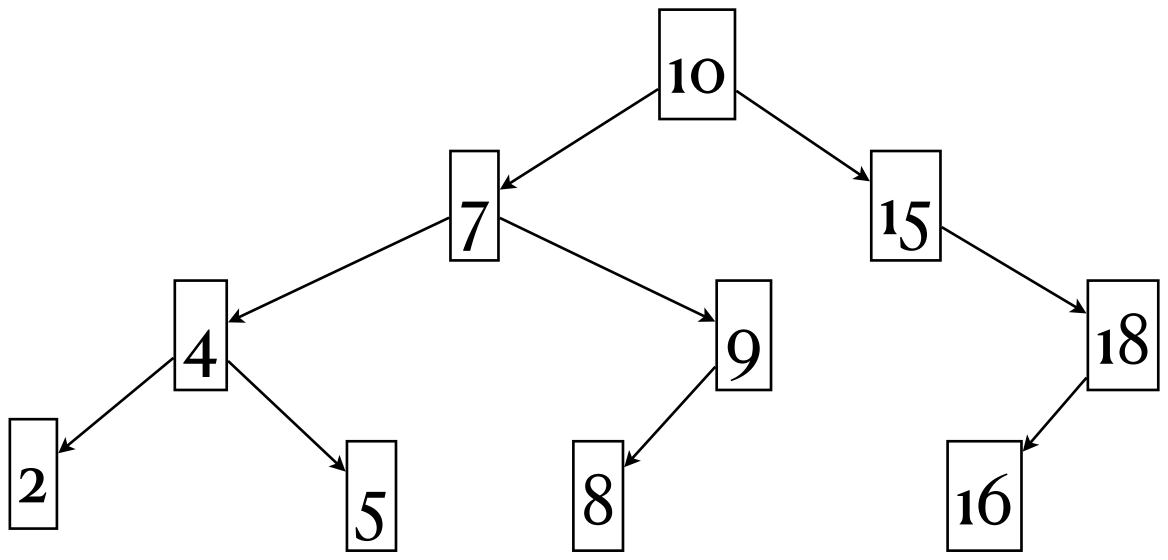
- If $r = 0.01$, then the optimal size $m = 9.6n$
- If $r = 0.05$, then the optimal size $m = 6.2n$

Applications of BF in Bioinformatics

- Efficient counting of k-mers in DNA sequences using a bloom filter (2011)
- Fast Search of Thousands of Short-Read Sequencing Experiments (2016)
- Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees (2018)
- AllSome Sequence Bloom Trees (2018)
- Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index (2018)
- MetaProFi: an ultrafast chunked Bloom filter for storing and querying protein and nucleotide sequence data… (2023)

kd-trees

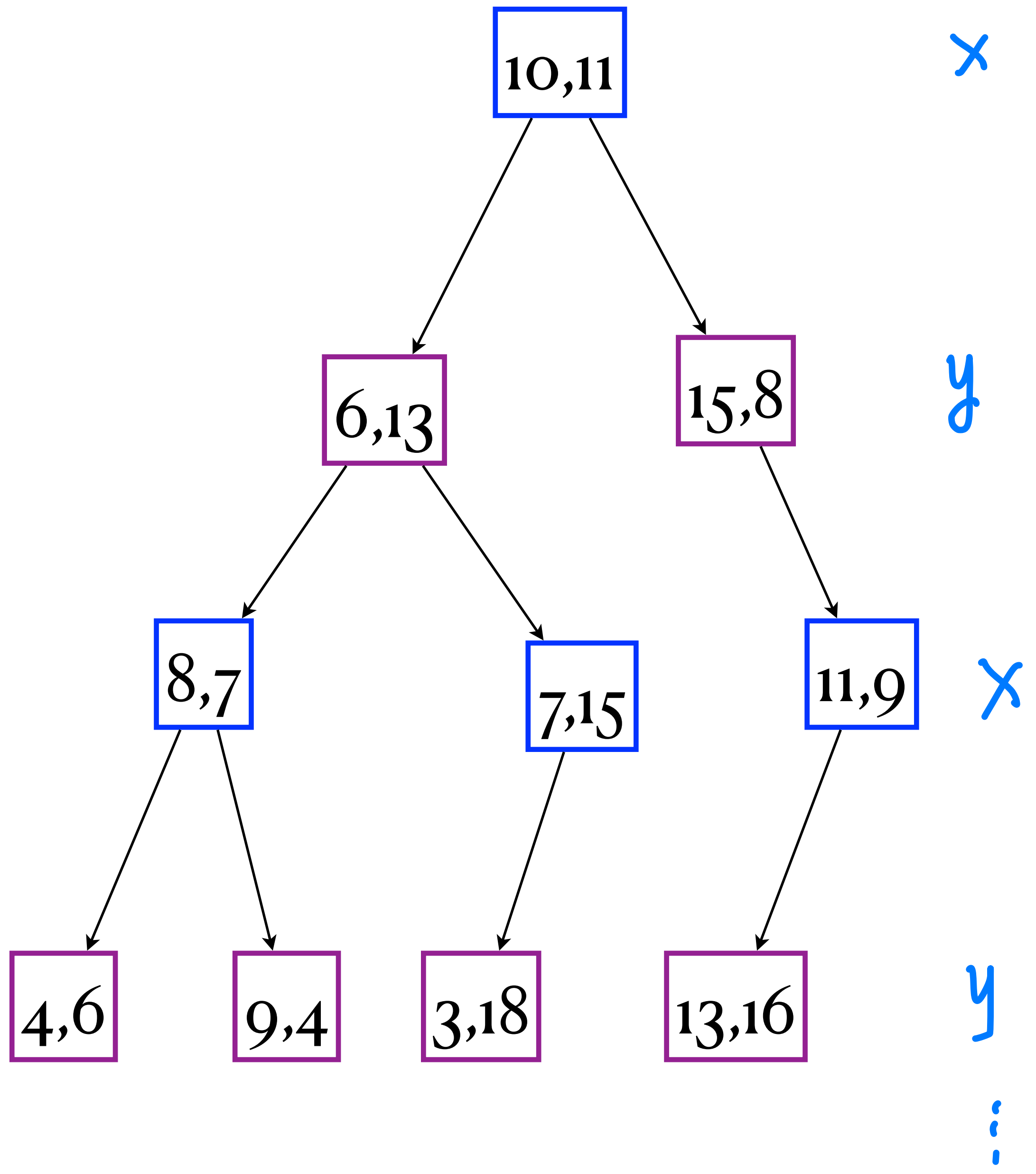
- To store/query high-dimensional data
- Name originally meant 2d-trees, 3d-trees, etc, where k is the number of dimensions.
- Generalize binary search tree.



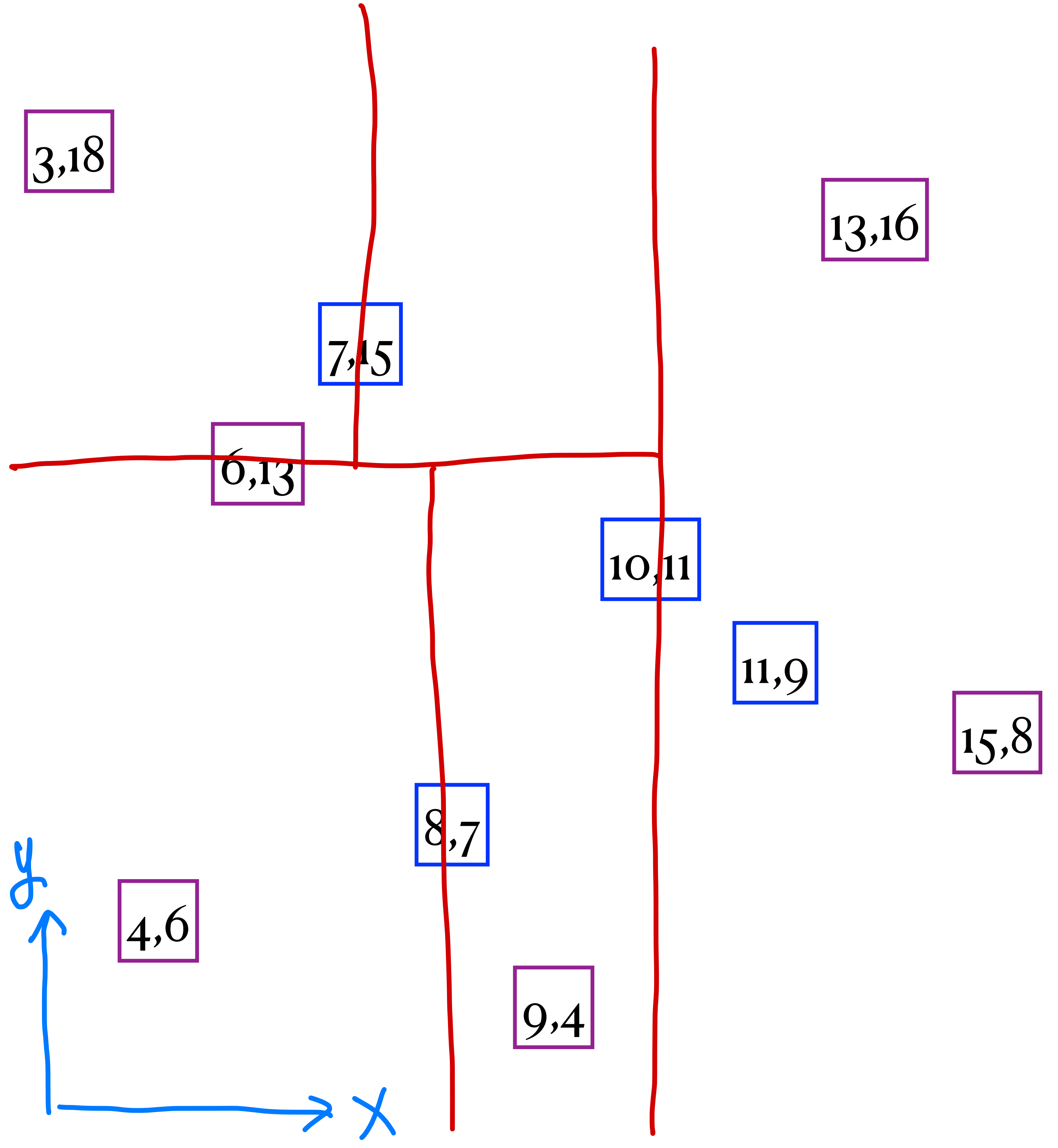
Definition of kd-trees

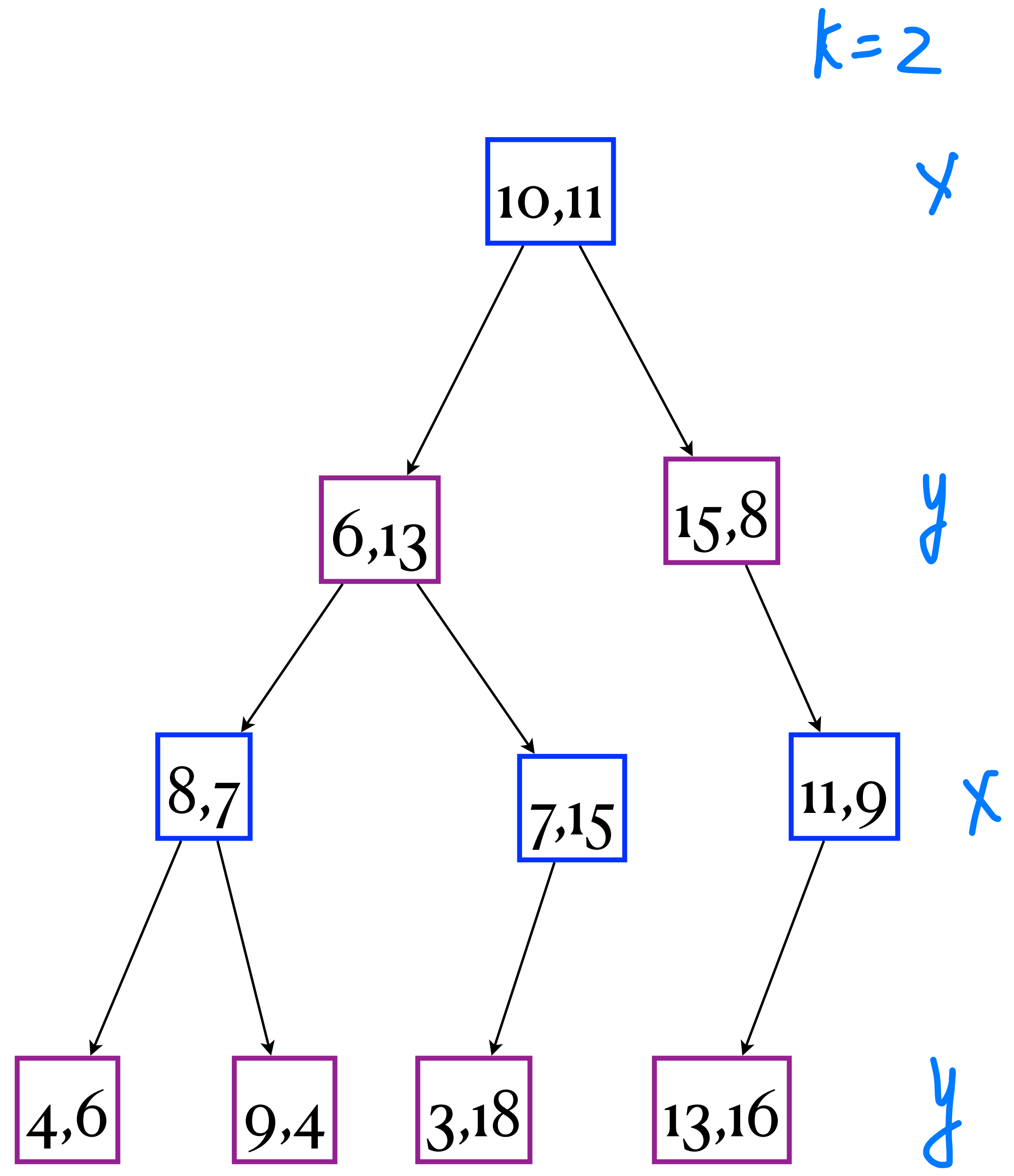
K=2

- Each node stores a data point.
- Each node has up to 2 children.
- Each level has a "cutting dimension (cd)", rotating from 1 to k down the tree.
- For each node v with cd(v) = x
    - All nodes in its left-subtree have its x-coordinate < v.x.
    - All nodes in its right-subtree have its x-coordinate > v.x.

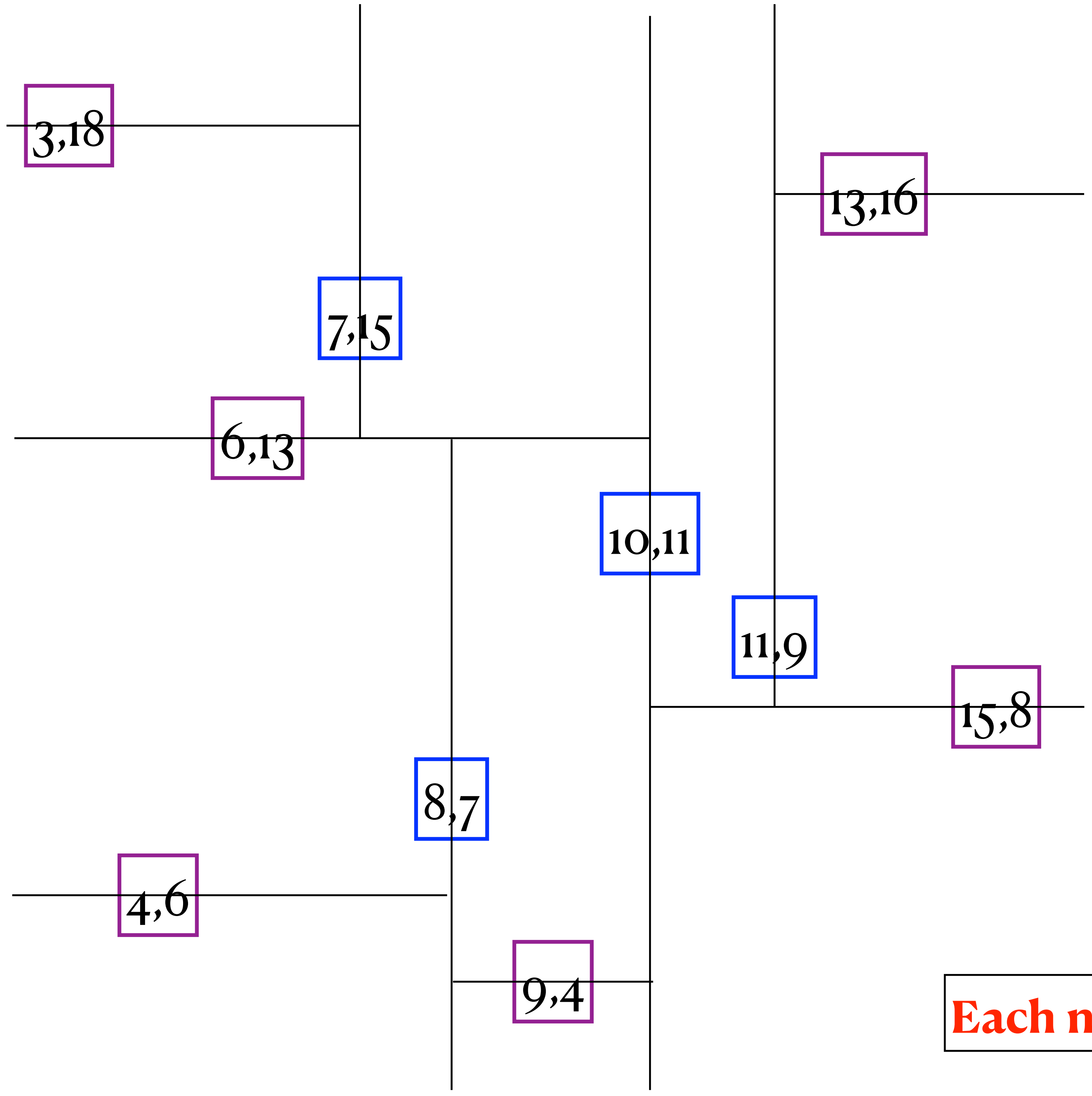


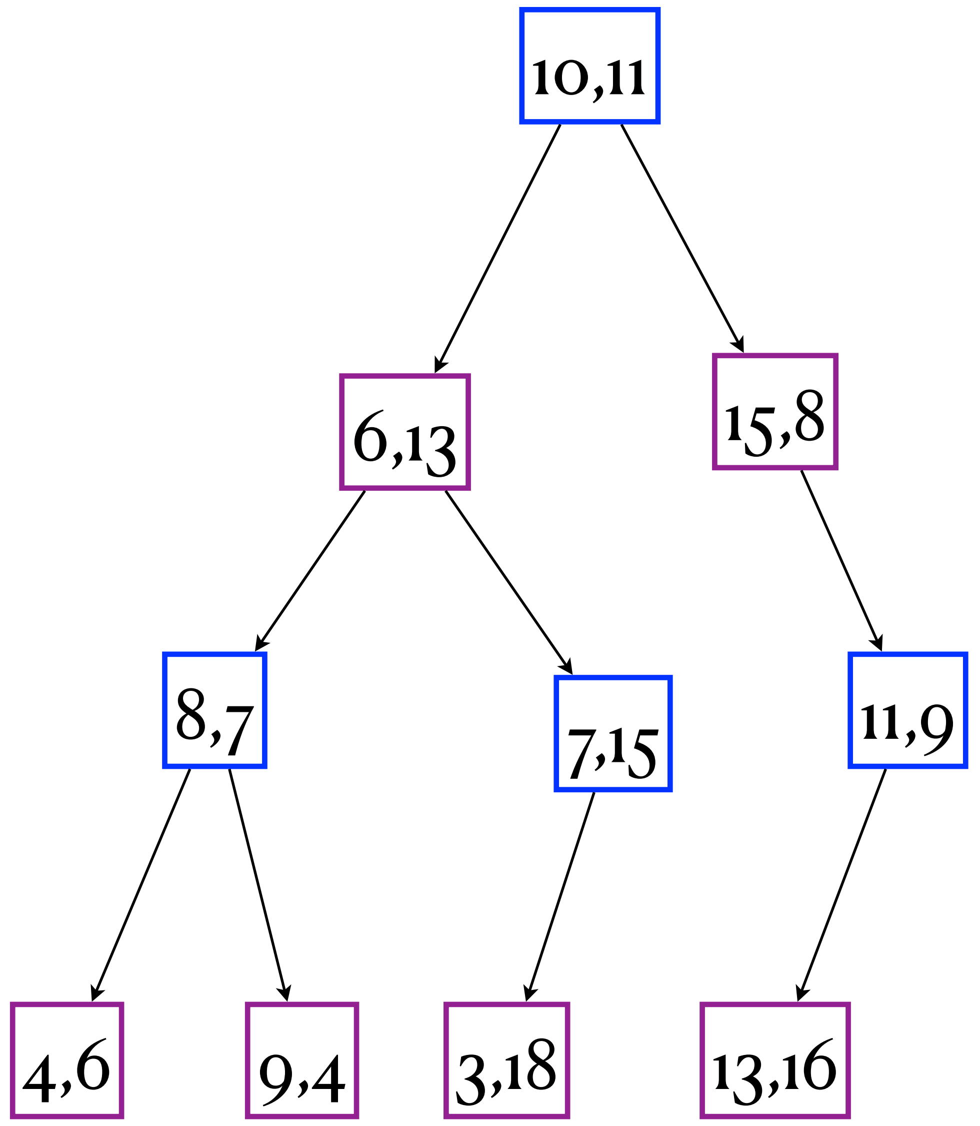
Partition of the k-dimension Space





Partition of the k-dimension Space





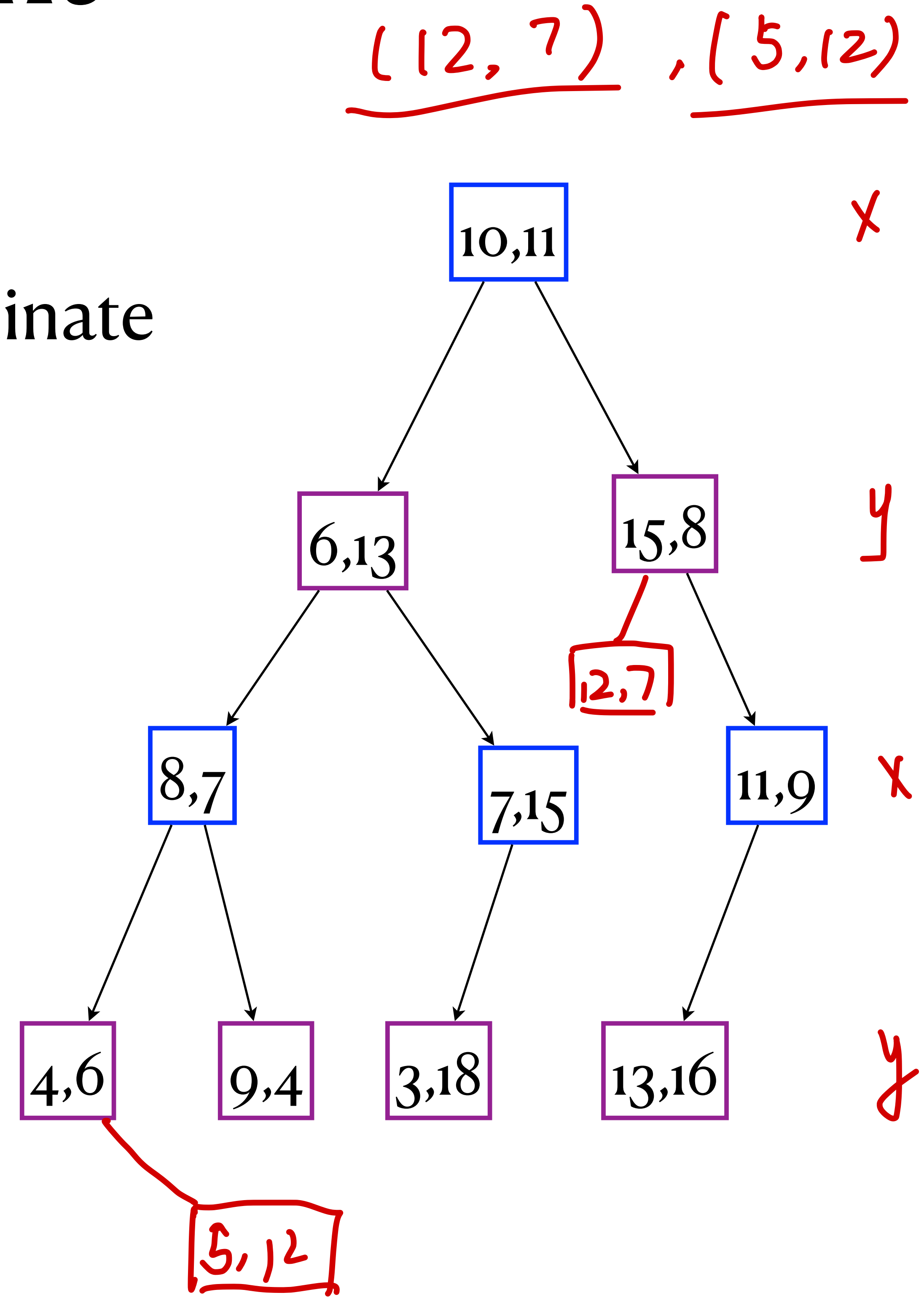
Each node is associated with a region!

Add Point

- Add point p=(p.x, p.y) to the tree
- Assume no duplicated x- or y-coordinate

```
function add(node, cd, p)
  if(node==null):
    node <- new Node(p);
  else if(p.cd < node.cd):
    add(node.left, (cd+1)%k, p)
  else
    add(node.right, (cd+1)%k, p)
  end if
end function
```

Running time: O(height)



Constructing Balanced kd-tree

- Construct kd-tree for a given set of points P = (p1, p2, ..., pn)

```
sort P according to x -> PX && sort P according to y -> PY
function construct(PX, PY, cd)
  if(cd == x): m = PX[|PX|/2]
  if(cd == y): m = PY[|PY|/2]
  partition PX into (PX1, m, PX2), where PX1.cd < m.cd < PX2.cd
  partition PY into (PY1, m, PY2), where PY1.cd < m.cd < PY2.cd
  L <- construct(PX1, PY1, (cd+1)%k)
  R <- construct(PX2, PY2, (cd+1)%k)
  create a new node for point median
  node.left = L && node.right = R
  return node
end function
```

$\Theta(n \log n)$

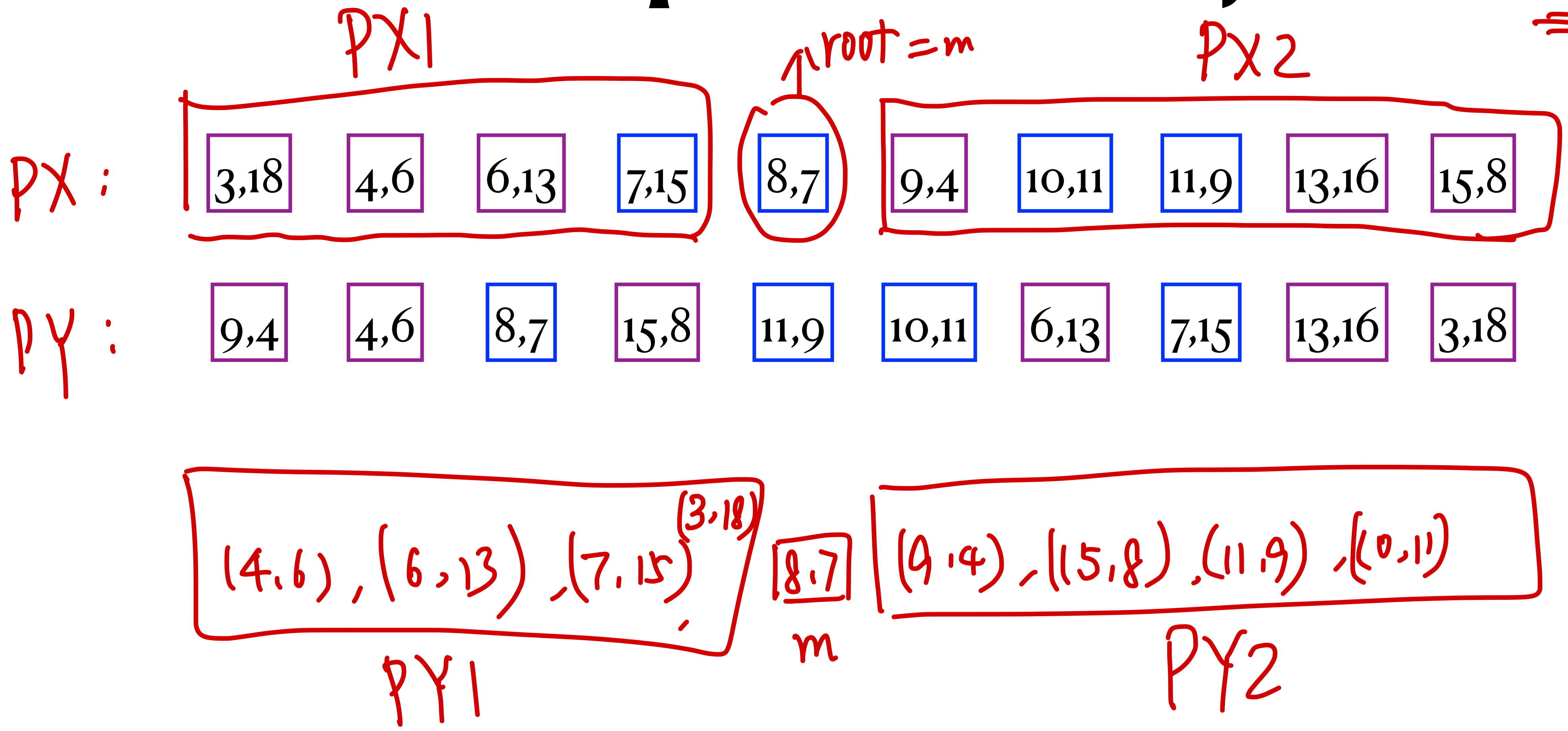
$T(n) = \Theta(n) + 2T(n/2)$

$\Rightarrow T(n) = \underline{\Theta(n \log n)}$

$O(n)$

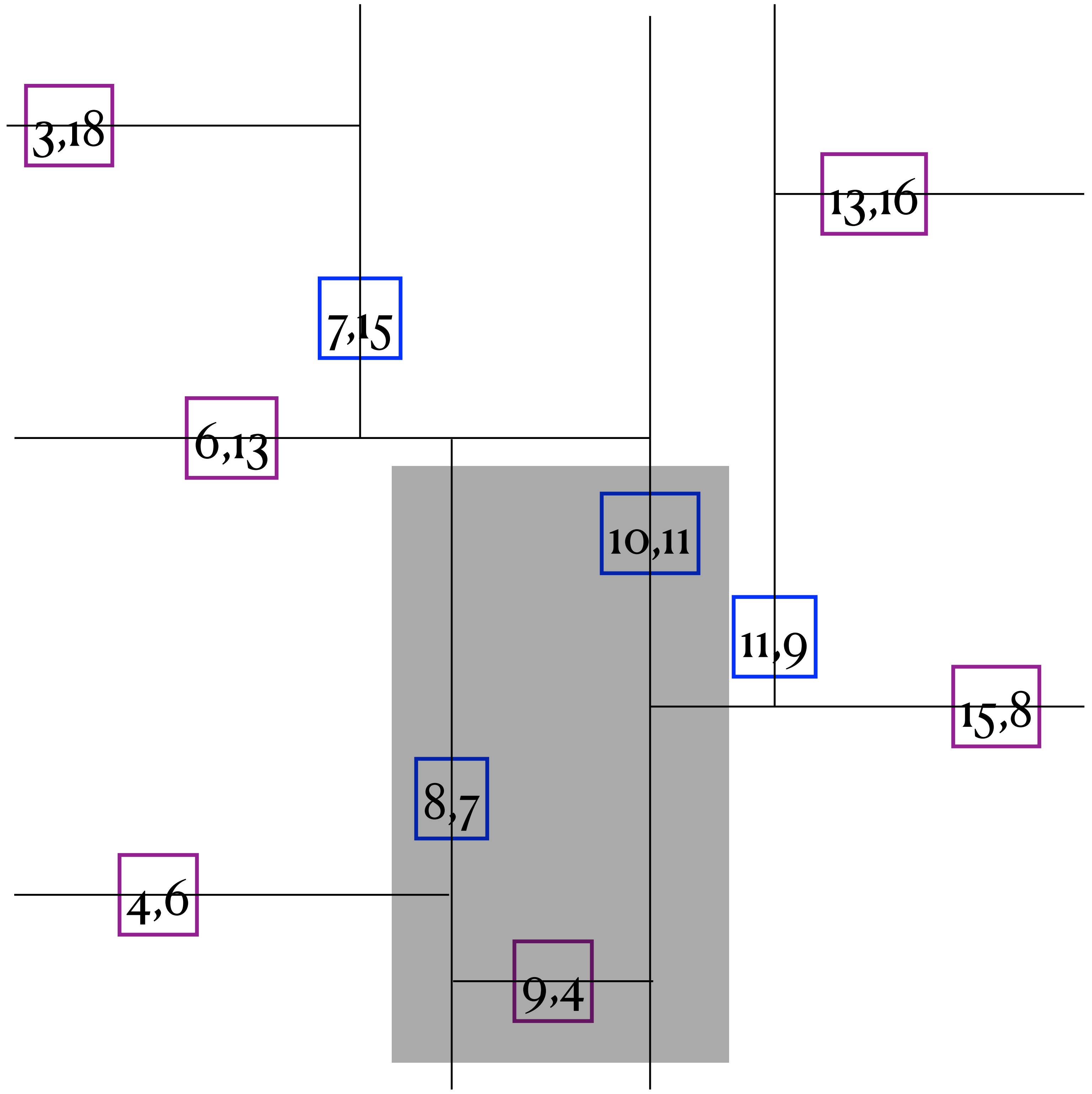
$O(\sqrt{n})$

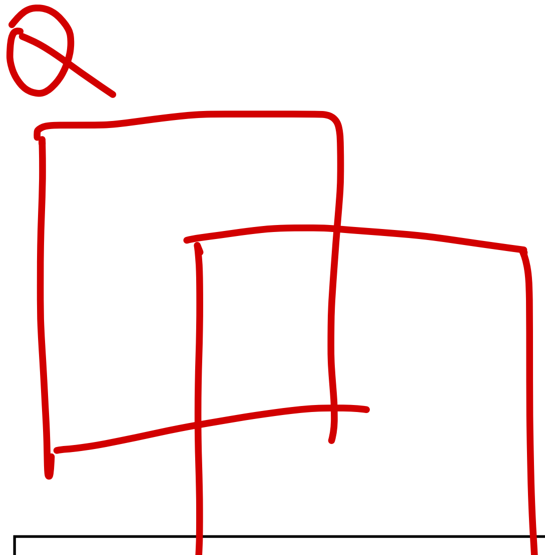
Examples and Analysis



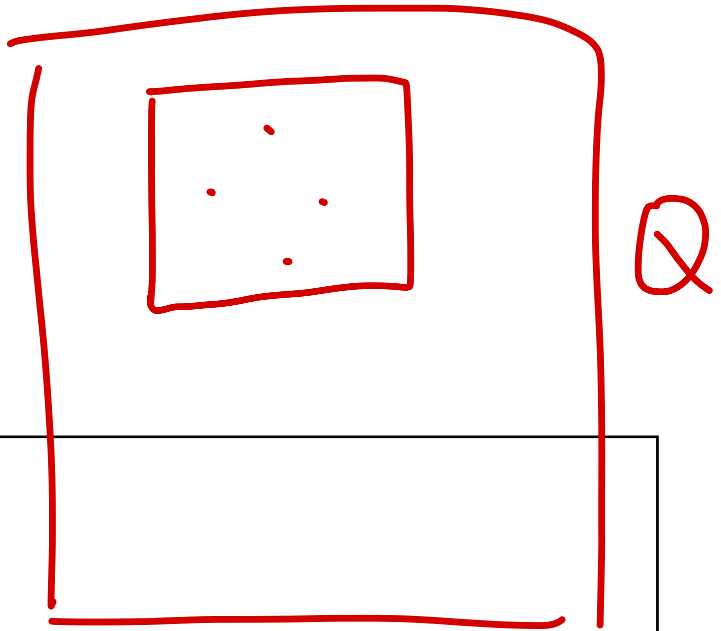
Range Query

- Input: query Q, a rectangle parallel to the axis
- Output: points in Q
- Define query(v, Q) returns the set of points in Q that are in the subtree rooted at v.





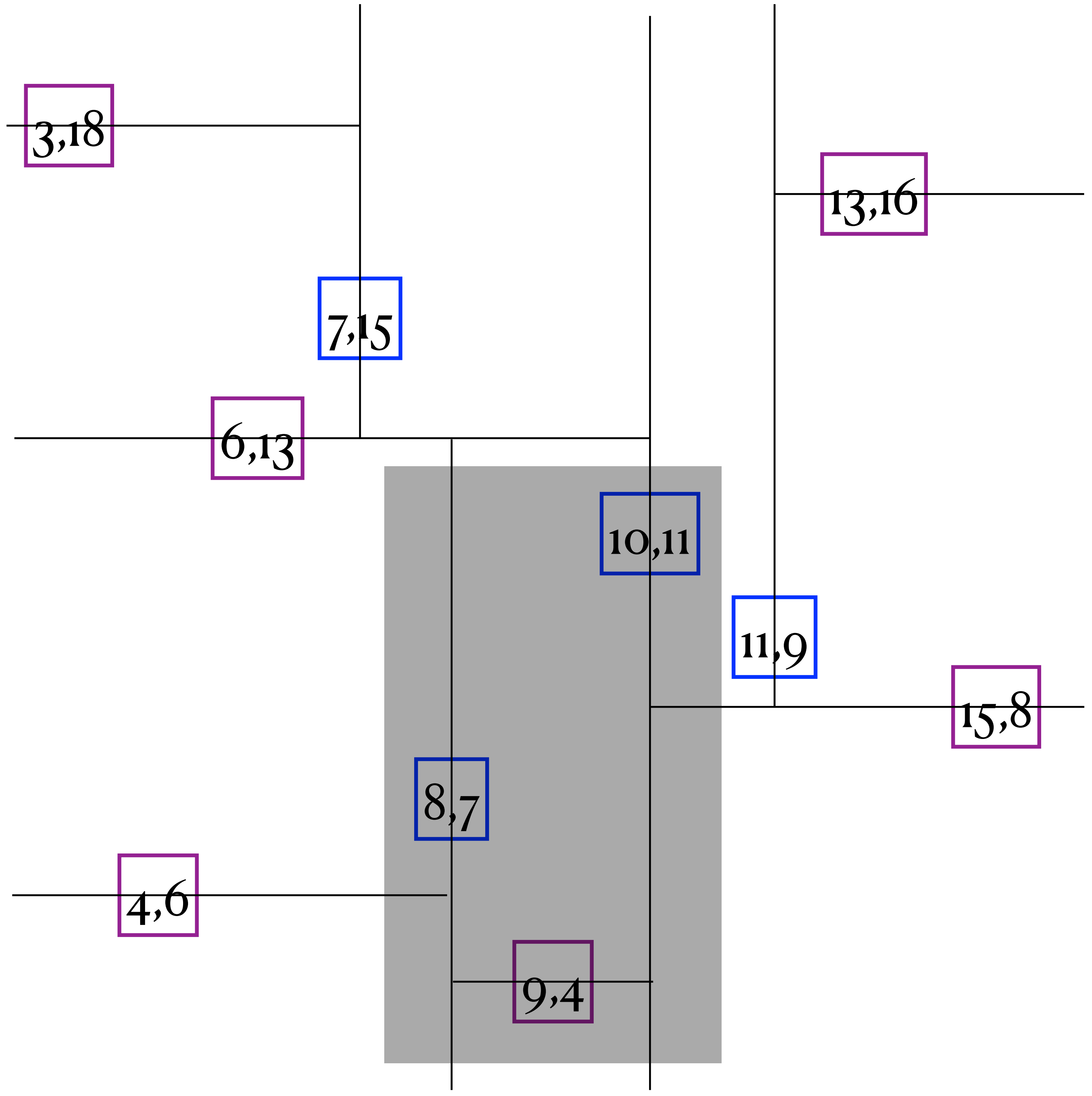
Range Query



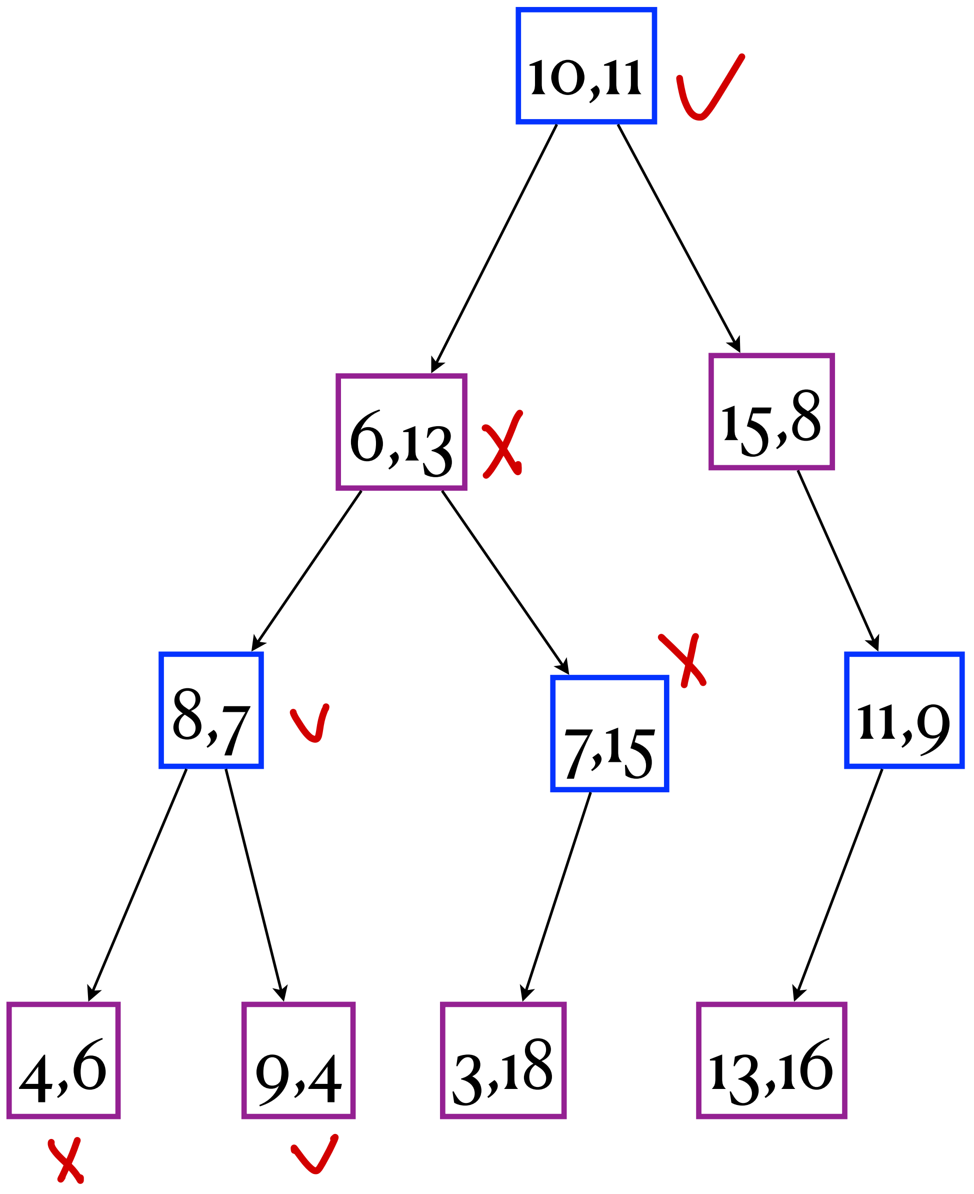
```
function query(v, Q)
  if(v is contained in Q): collect v
  if(v is a leaf): return
  if(region(v.left) is fully contained in Q): collect-all(v.left)
  else if(region(v.left) partially overlaps Q): query(v.left, Q)
  if(region(v.right) is fully contained in Q): collect-all(v.right)
  else if(region(v.right) partially overlaps Q): query(v.right, Q)
end function
```

- Whether a node is contained in Q can be detected in O(1) time.
- Whether a region is contained in Q can be detected in O(1) time.
- Whether two rectangles overlap can be detected in O(1) time.

An Example



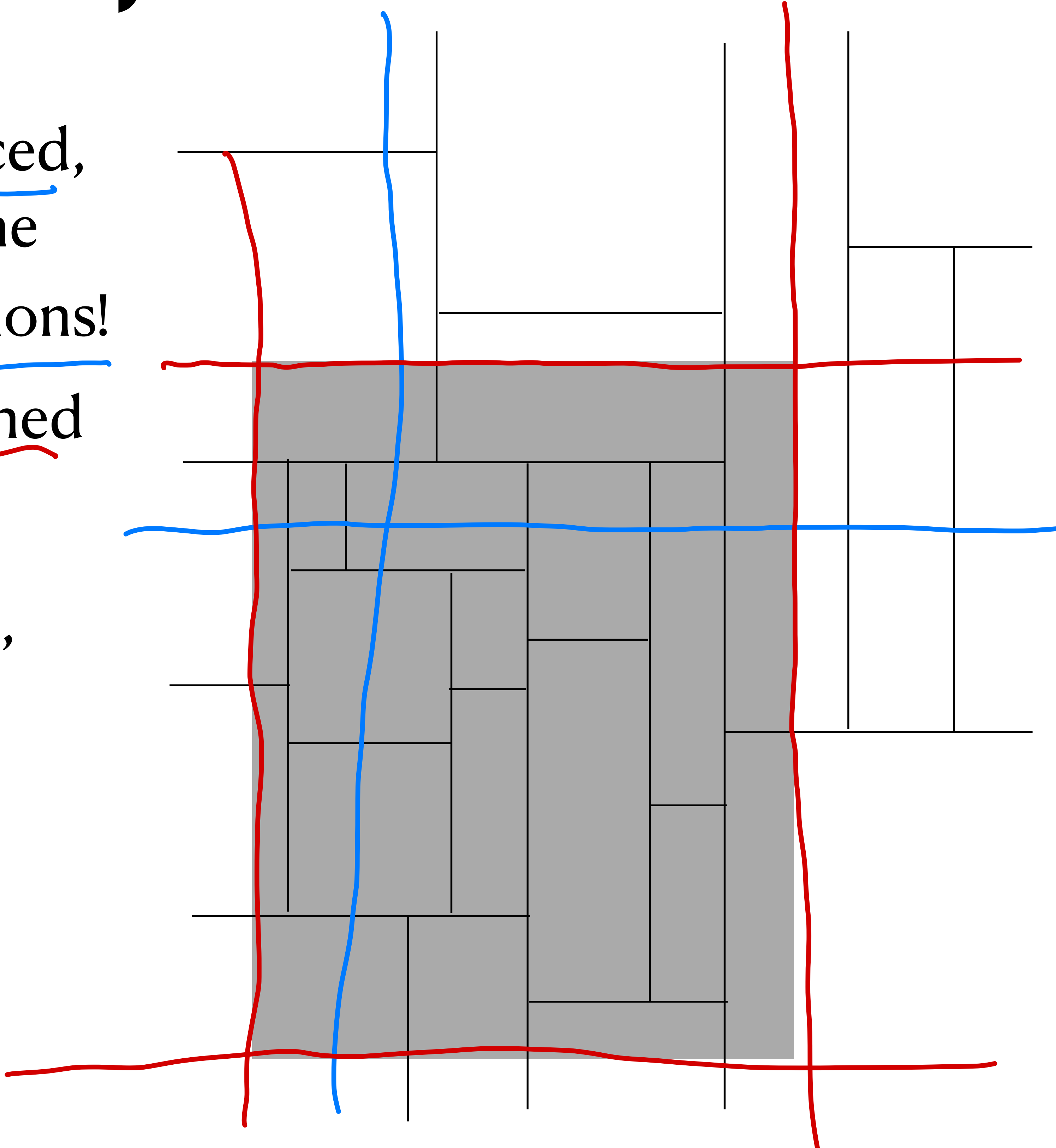
3,18
13,16
7,15
6,13
10,11
11,9
15,8
8,7
4,6
9,4

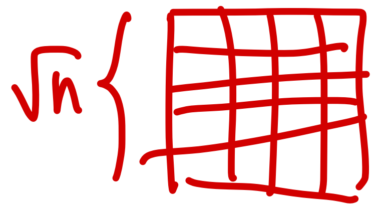


10,11 ✓
6,13 ✗
15,8
8,7 ✓
7,15 ✗
11,9
4,6 ✗
9,4 ✓
3,18
13,16

Analysis

- When the kd-tree is balanced, each vertical/horizontal line intersects with $O(\sqrt{n})$ regions!
- The total number of searched regions/nodes: $O(\sqrt{n})$
- Running time: $O(\sqrt{n} + m)$, where m is the number of points that are in Q.





n