CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Spatial Data Structures (pt 1)

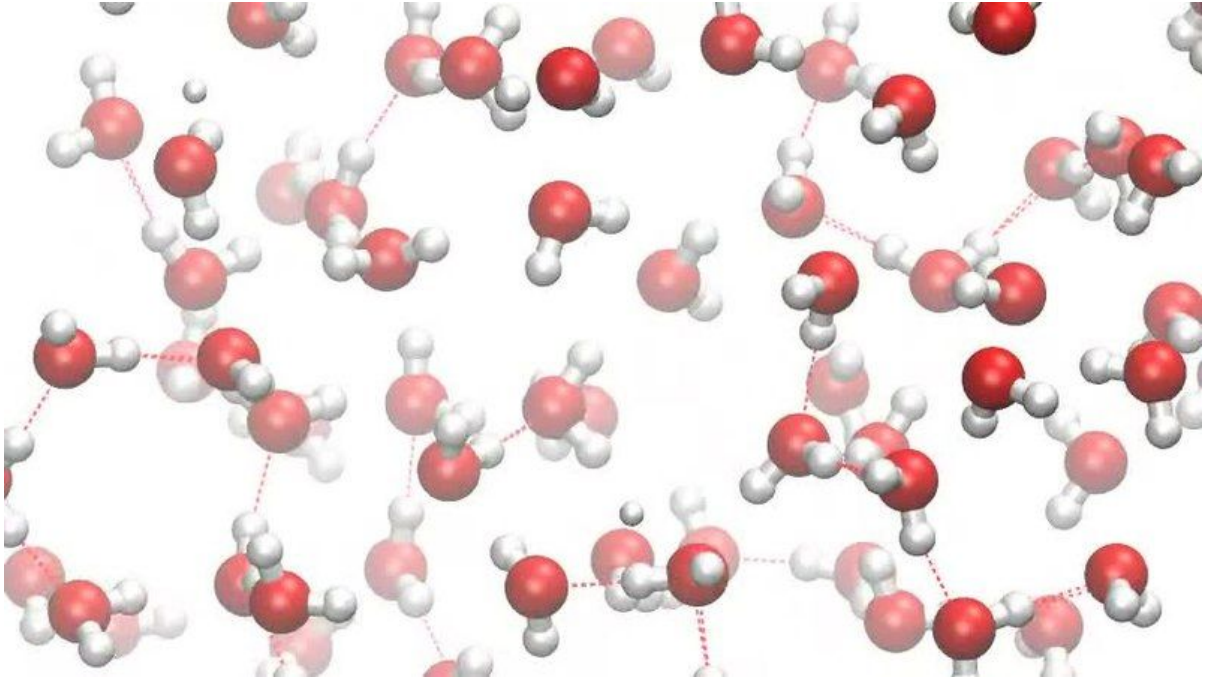
Announcements

- PA3 due Sunday
- Course Evaluations are Open!!
  - If enough people do the evaluation, we will release an exam question early
  - See Piazza for details, void where prohibited

Some Problems are REALLY Big

ESA/Hubble and NASA: http://www.spacetelescope.org/images/potw1006a/

Some Problems are REALLY Small



Molecular Dynamics Simulation of Liquid Water
https://commons.wikimedia.org/wiki/File:A_Molecular_Dynamics_Simulation_of_Liquid_Water_at_298_K.webm

Some Problems are REALLY Detailed

This is **NOT** a photo. It is a computer generated image.

https://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29#/media/File:Glasses_800_edit.png

What do these things have in common?

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What "bodies" (other planets, molecules, etc) are close to each other?

Which object(s) will a ray of light bounce/projectile hit?

What objects are closest to a given point?

Which objects fall within a given range?

What do these things have in common?

The have MANY elements (celestial bodies, molecules, mesh cells, etc) which are organized spatially

What "bodies" (other planets, molecules, etc) are close to each other?

Which object(s) will a ray of light bounce/projectile hit?

What objects are closest to a given point?

Which objects fall within a given range?

How can we organize these elements in a way that allows us to efficiently answer these questions?

Related Problems

Mapping

- What's within ½ mile of me?
- What's within 2 minutes of my route?

Games

- What objects are close enough that they might need to be rendered?

Science

- "Big Brain Project": Neuron A fired, so what other neurons are close enough to be stimulated?
- "Astronomy"/"MD": What forces are affecting a particular body, and what forces can we ignore/estimate?

Organizing/Storing Our Data

Can we use a HashTable to allow us to efficiently answer these questions?

Organizing/Storing Our Data

Can we use a HashTable to allow us to efficiently answer these questions?

No. HashTables help us find EXACT matches very quickly, but these types of questions are not looking for exact matches. HashTables do not keep our data "organized".

Organizing/Storing Our Data

What data structure have we seen already that lets us efficiently organize/store "sorted" data?

Organizing/Storing Our Data

What data structure have we seen already that lets us efficiently organize/store "sorted" data?

Idea: What if we organize our data in a BST

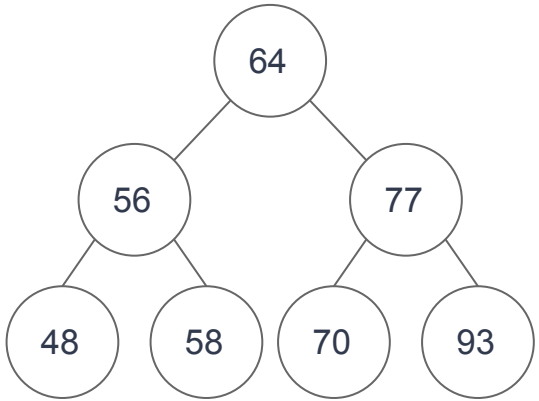
Binary Search Trees (for one dimension)

`Insert`

- Find the right spot: ***O*(*d*)**
- Create and insert the node: ***O*(1)**

`Find`

- Find the right node: ***O*(*d*)**
- Return the value if it is present: ***O*(1)**



If the tree is balanced, O(*d*) = O(log(*n*))

Multiple Dimensions

This worked for 1-dimensional data…How could we change it to work with 2-dimensional data, ie (x,y) coordinates?

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find points with a specific x coordinate
2. Find me points with a specific y coordinate
3. Find me points with a specific (x,y) coordinate

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find points with a specific x coordinate
2. Find me points with a specific y coordinate
3. Find me points with a specific (x,y) coordinate

Idea 1: BST over x coordinates

- 2 is $O(n)$
- 3 is $O(\log(n) + |\text{points with same x}|)$

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find points with a specific x coordinate
2. Find me points with a specific y coordinate
3. Find me points with a specific (x,y) coordinate

Idea 1: BST over x coordinates

- 2 is $O(n)$
- 3 is $O(\log(n) + |\text{points with same x}|)$

Idea 2: BST over y coordinates

- 1 is $O(n)$
- 3 is $O(\log(n) + |\text{points with same y}|)$

Multiple Dimensions

Goal: Create a data structure that can answer:

1. Find points with a specific x coordinate
2. Find me points with a specific y coordinate
3. Find me points with a specific (x,y) coordinate

Idea 1: BST over x coordinates
- 2 is ***O*(*n*)**
- 3 is ***O*(log(*n*) + |points with same x|)**

Idea 2: BST over y coordinates
- 1 is ***O*(*n*)**
- 3 is ***O*(log(*n*) + |points with same y|)**

Idea 3: BST over x, then y (lexical order)
- 2 is <u>still</u> ***O*(*n*)**

Why did it fail?

Ideas 1 & 2

BST works by grouping “nearby” values together in the same subtree….

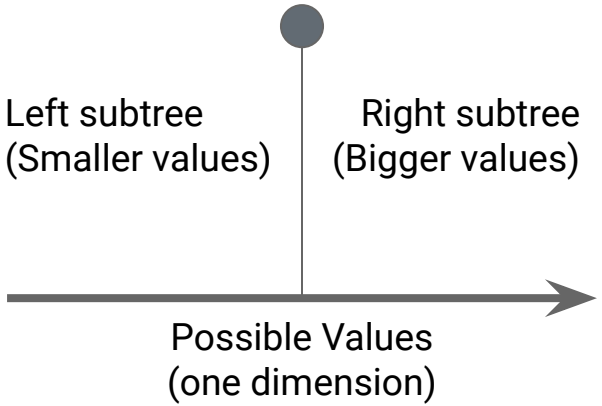
… but “near” in one dimension says nothing about the other!

Idea 3

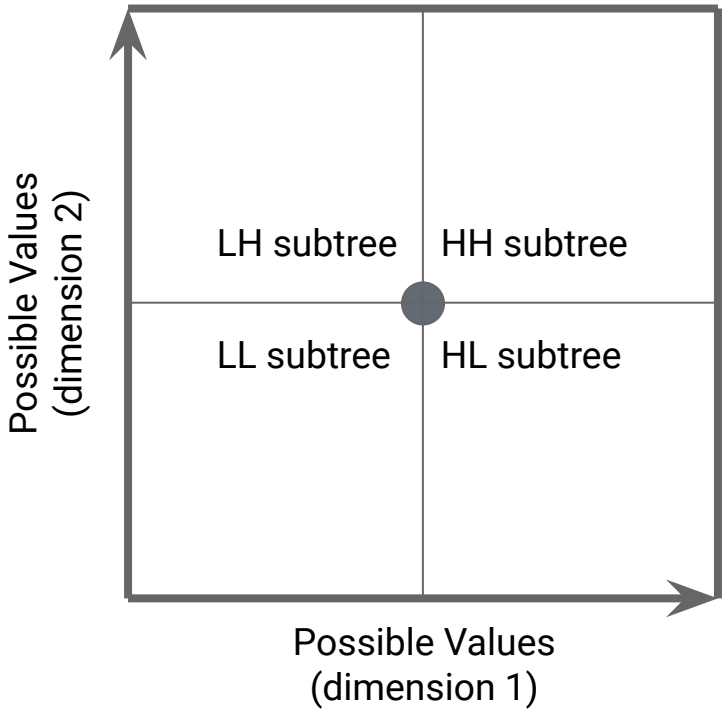
BST works by partitioning the data…

… but lexical order partitions fully on one dimension before partitioning on the other.

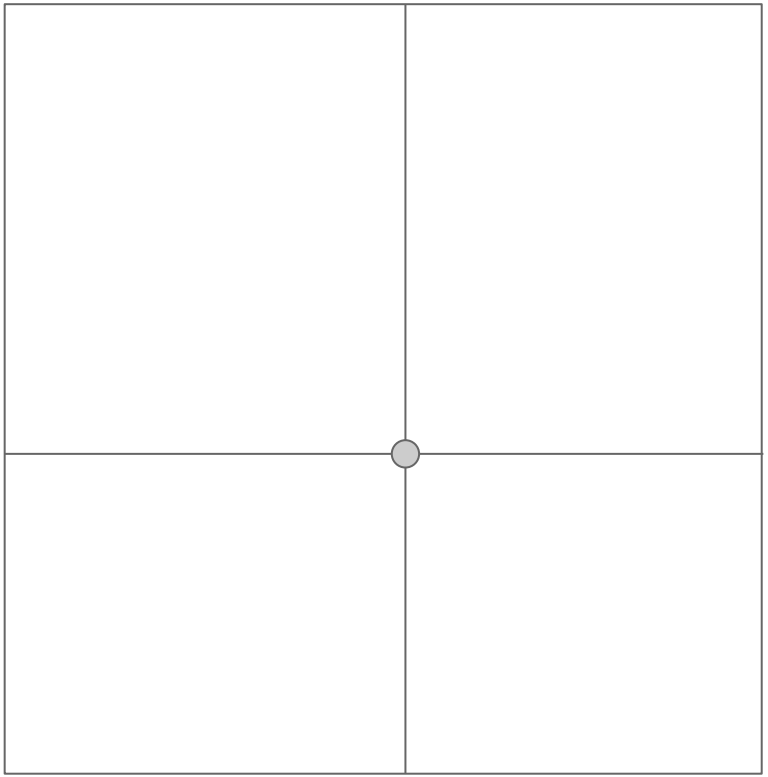
Instead of Partitioning on One Dimension…



Attempt 1 - Partition on BOTH dimensions

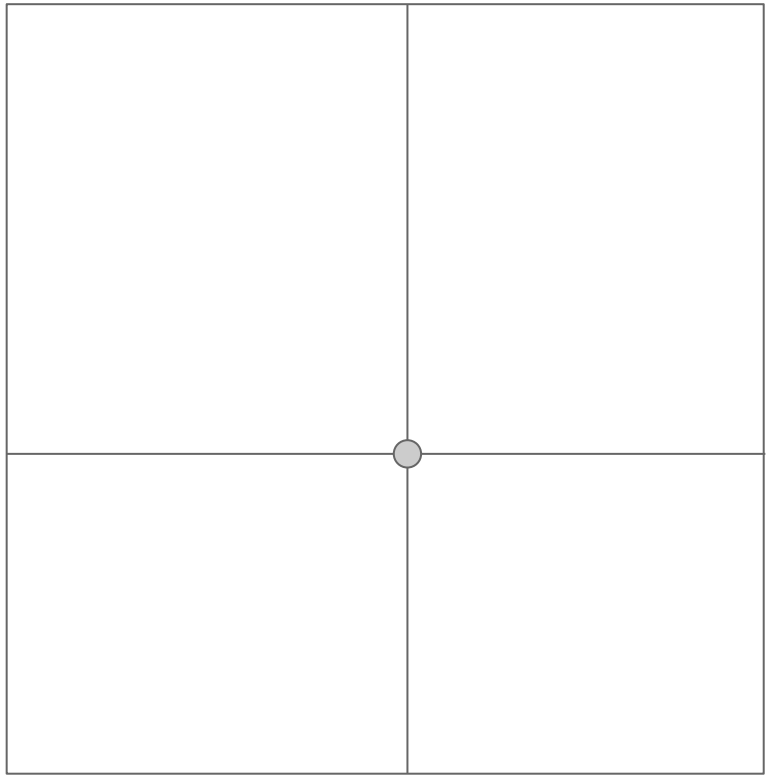


Attempt 1 - Partition on BOTH dimensions



5,4

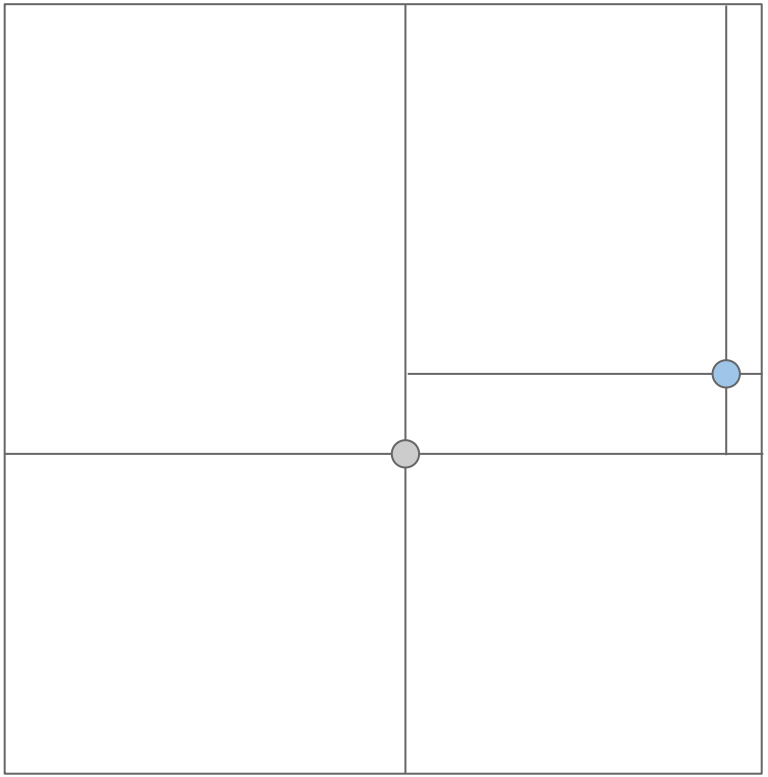
Attempt 1 - Partition on BOTH dimensions

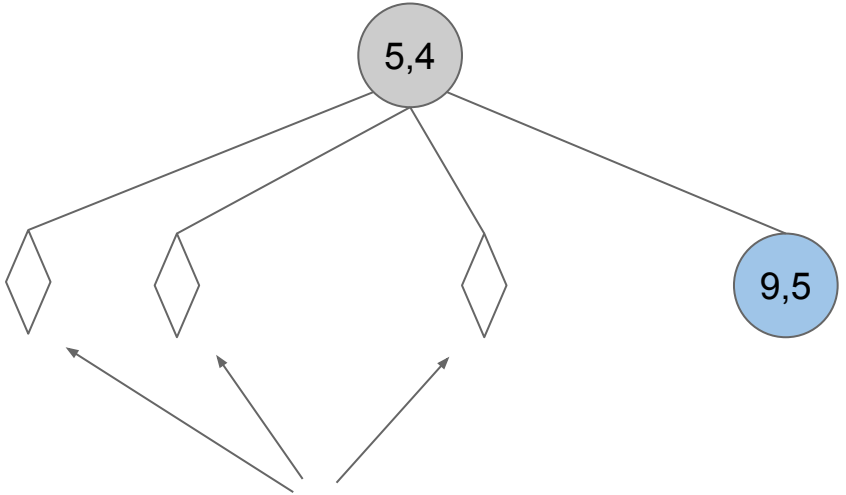


5,4

`insert((9,5))?`

Attempt 1 - Partition on BOTH dimensions

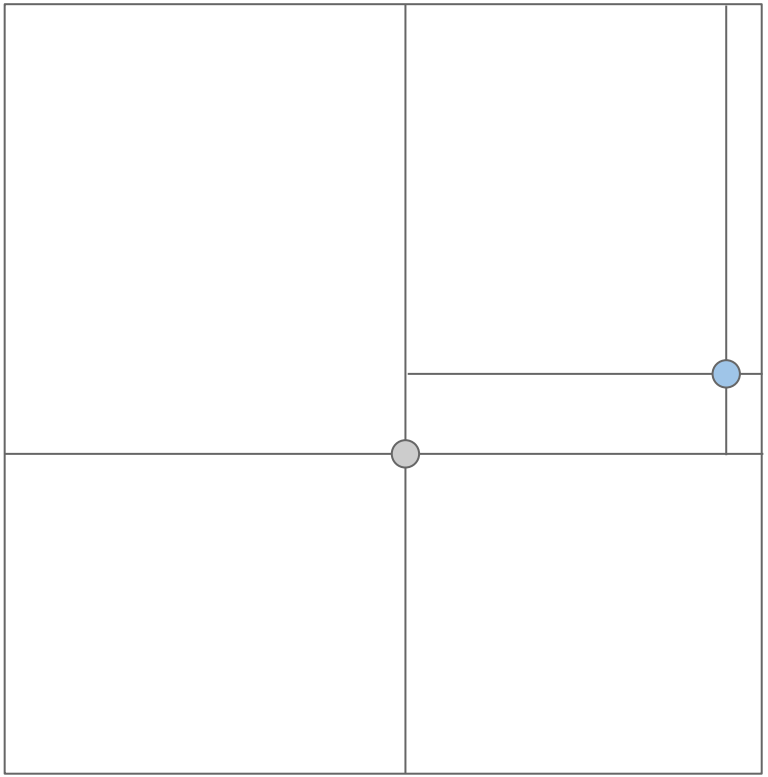


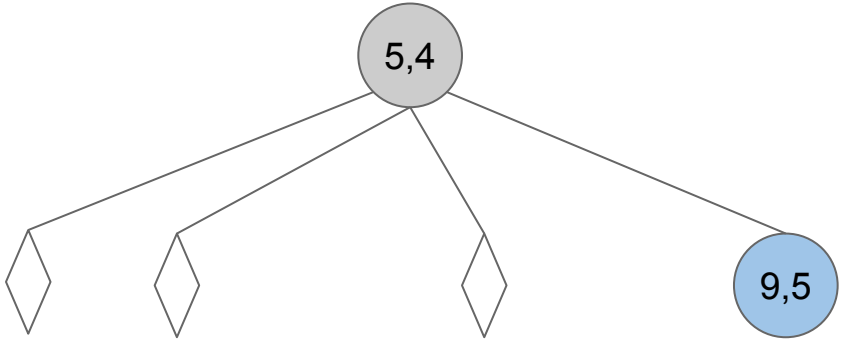


empty trees shown to emphasize which child we inserted to

```
insert((9,5))
```

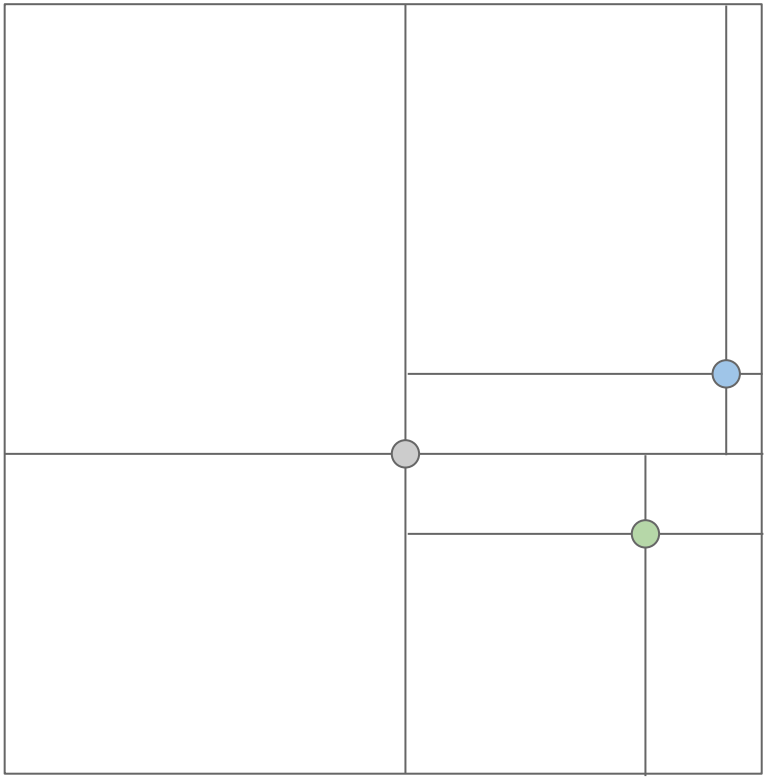
Attempt 1 - Partition on BOTH dimensions

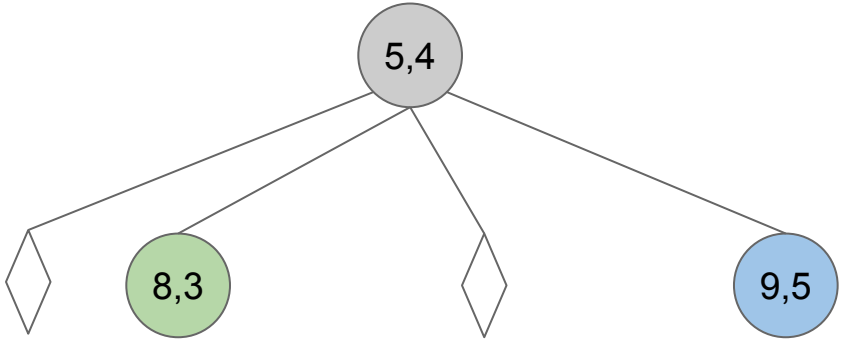




`insert((8,3))?`

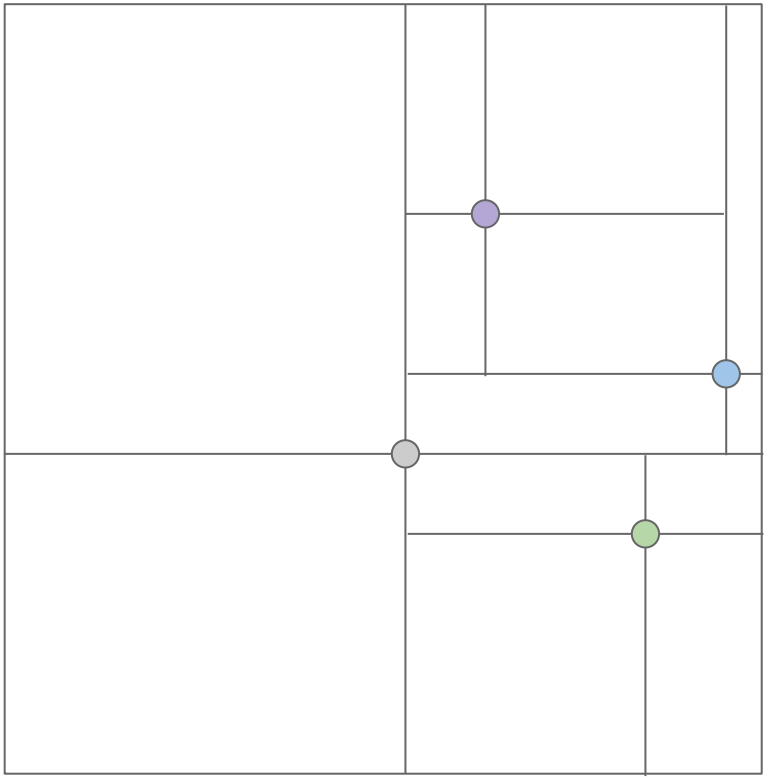
Attempt 1 - Partition on BOTH dimensions

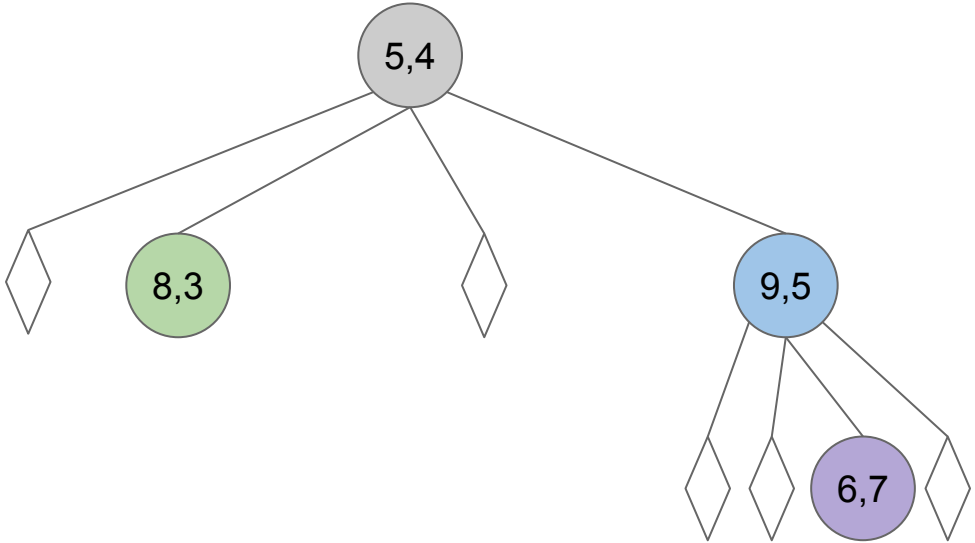





```
insert((8,3))
```

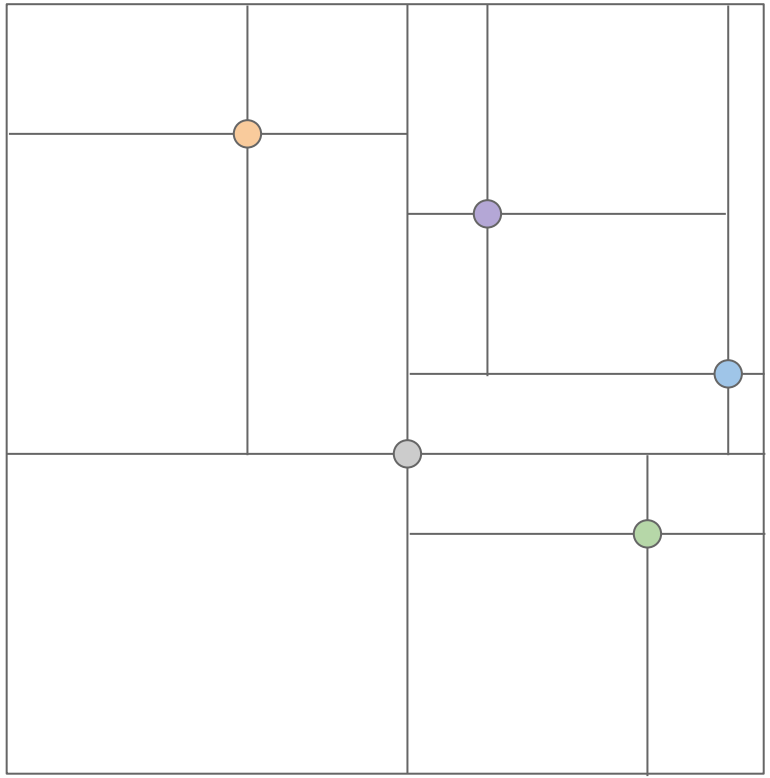
Attempt 1 - Partition on BOTH dimensions

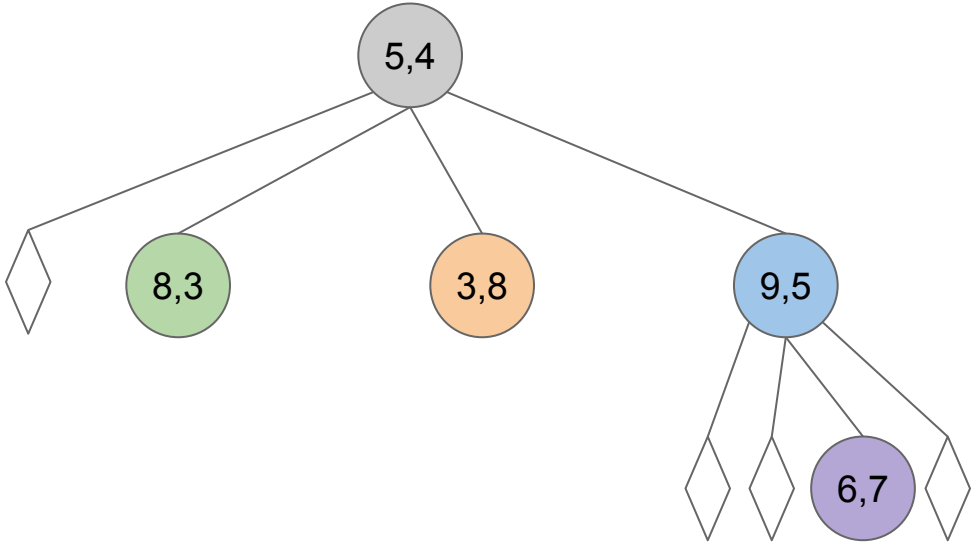





```
insert((6,7))
```

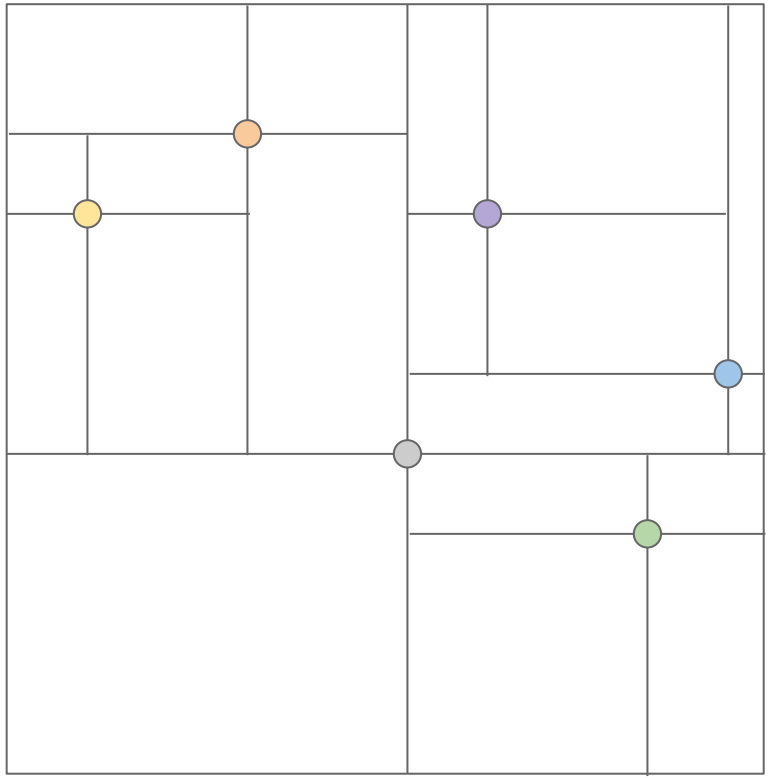
Attempt 1 - Partition on BOTH dimensions

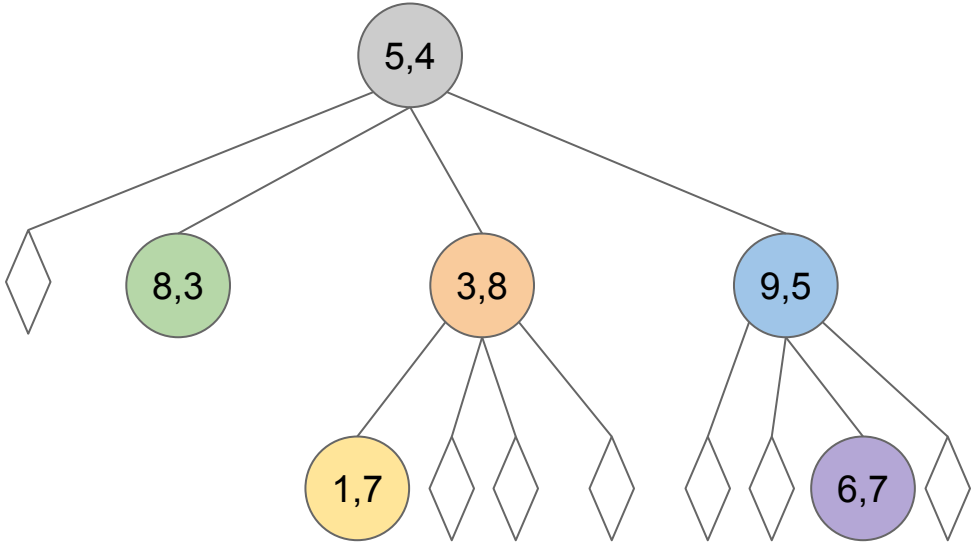





```
insert((3,8))
```

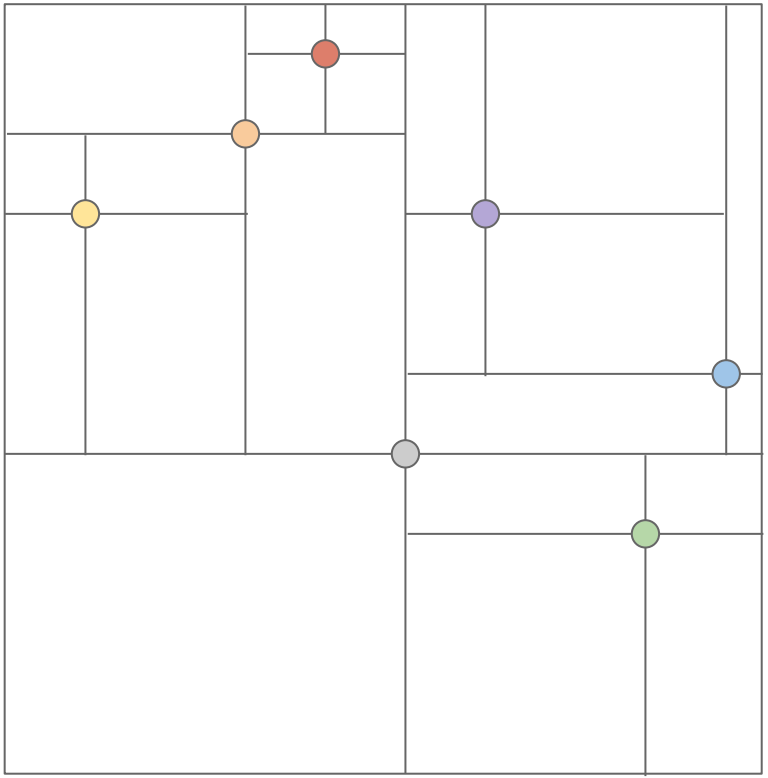
Attempt 1 - Partition on BOTH dimensions

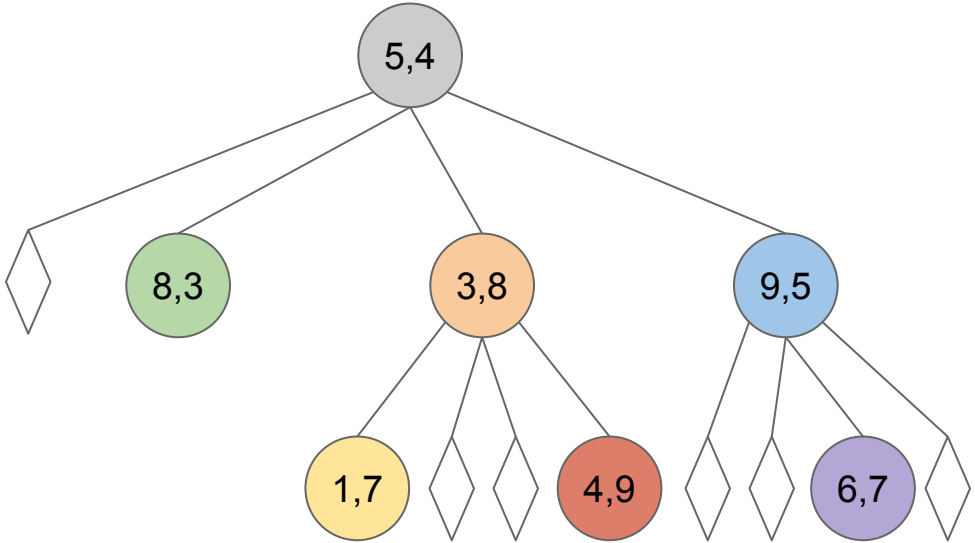





```
insert((1,7))
```

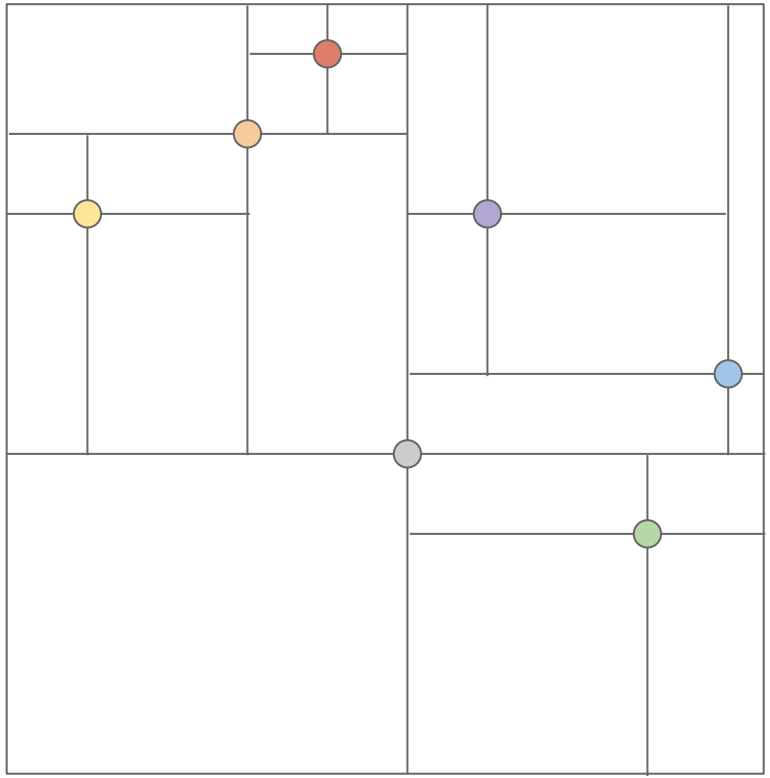
Attempt 1 - Partition on BOTH dimensions

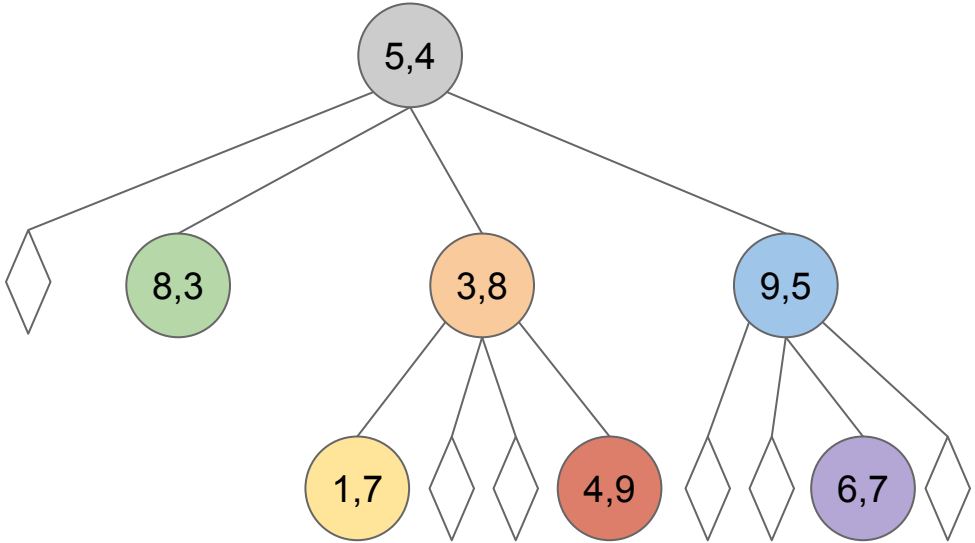





```
insert((4,9))
```

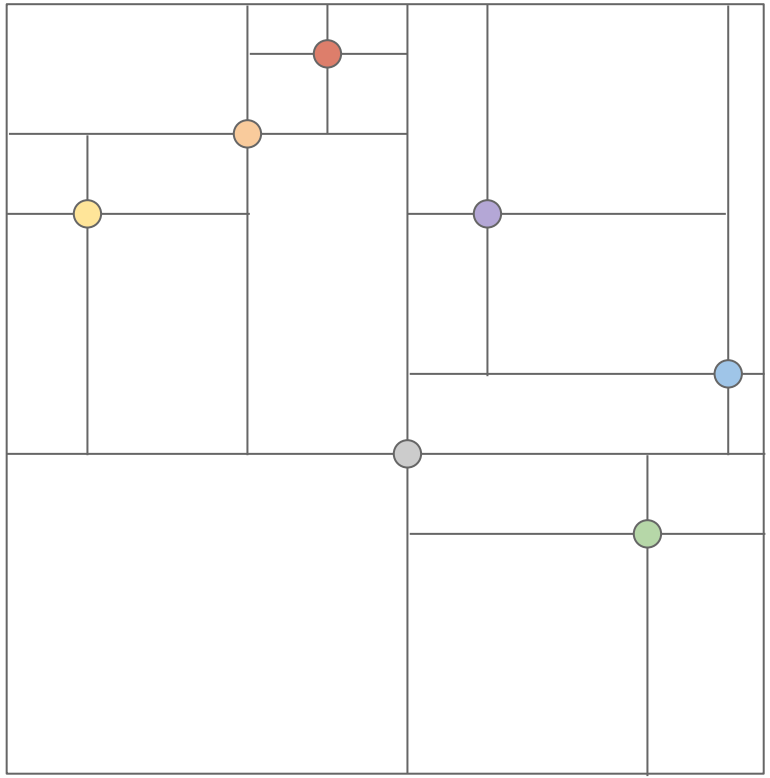
Attempt 1 - Partition on BOTH dimensions

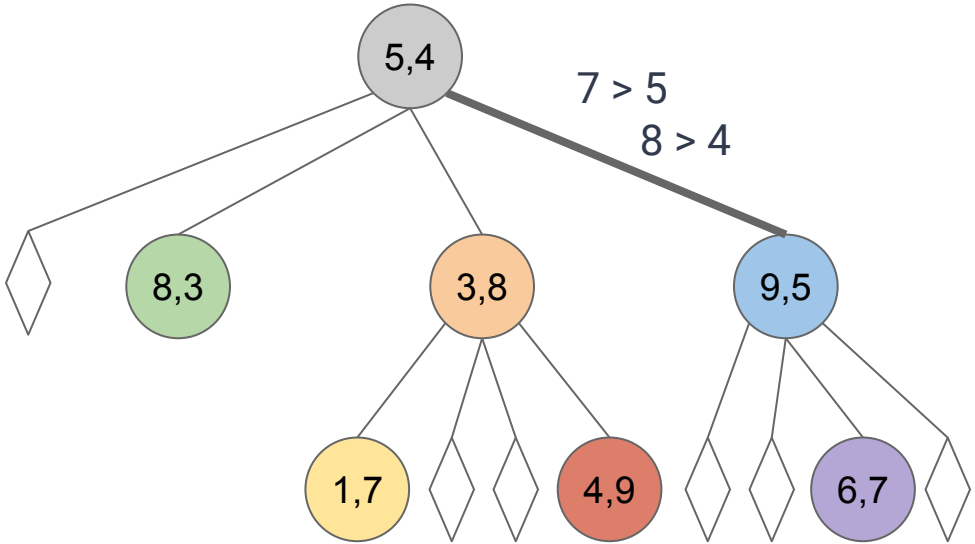




Where would (7,8) go?

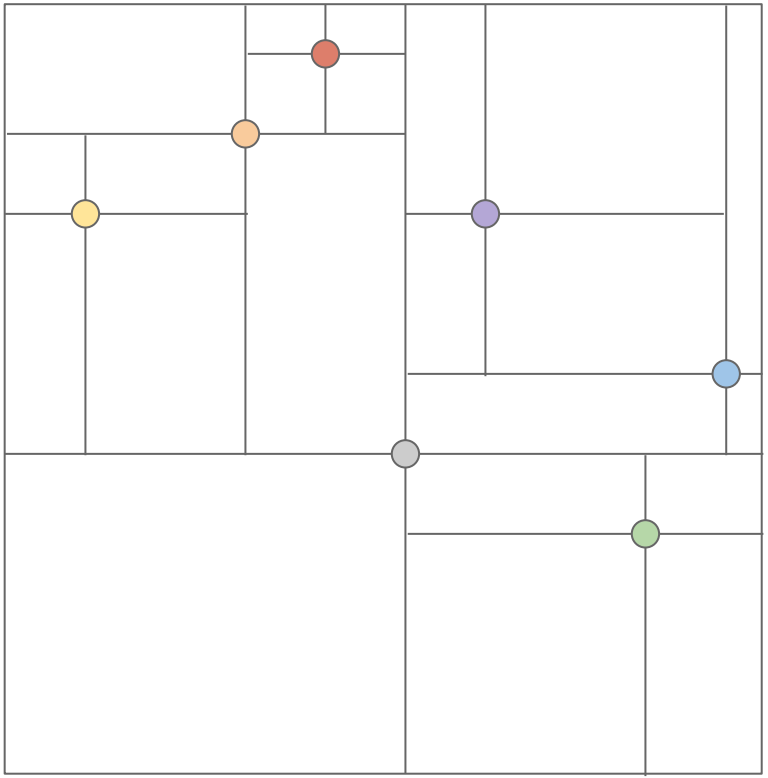
Attempt 1 - Partition on BOTH dimensions

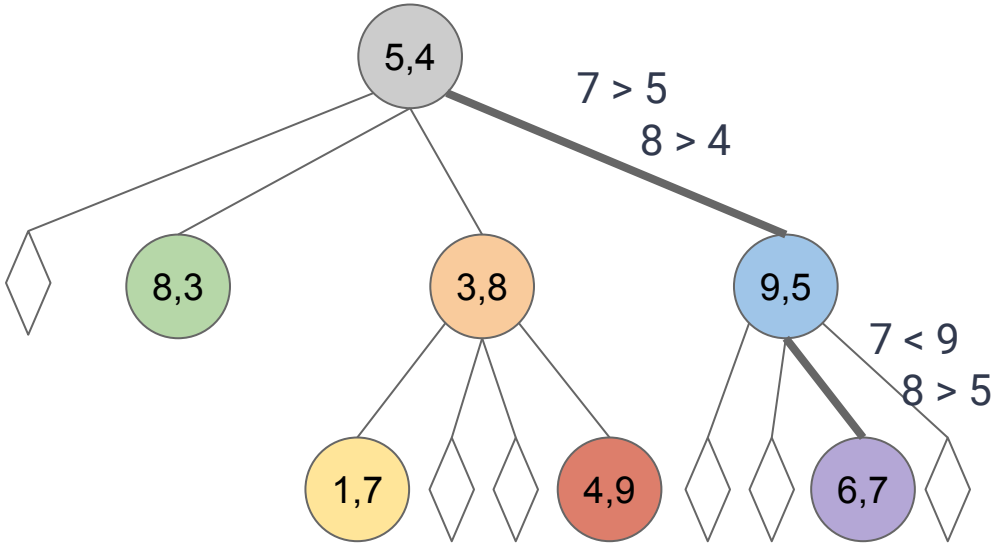




Where would (7,8) go?

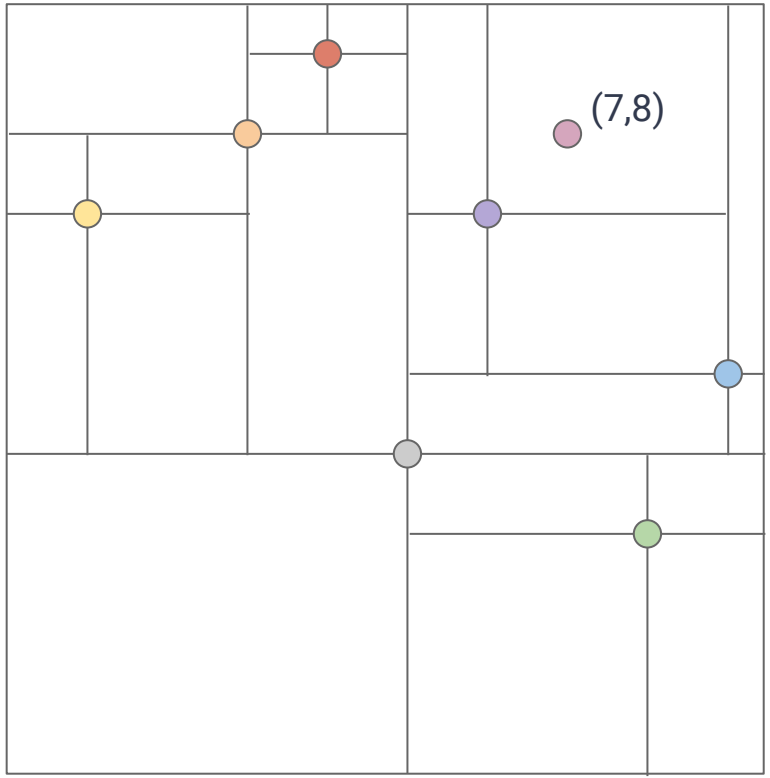
Attempt 1 - Partition on BOTH dimensions

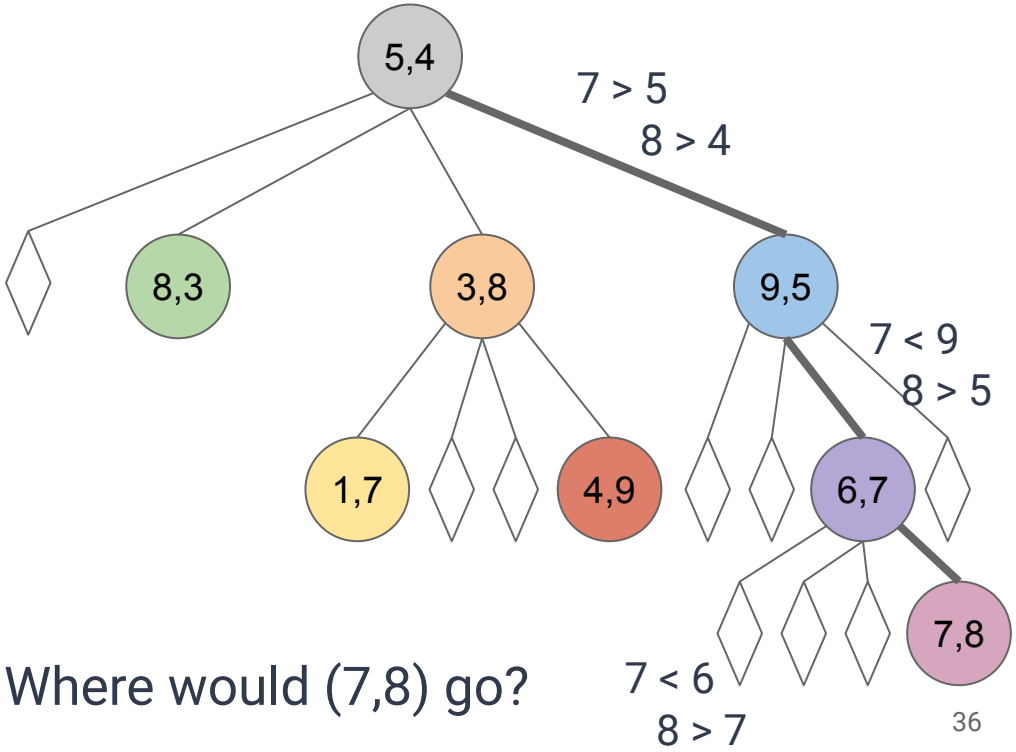




Where would (7,8) go?

Attempt 1 - Partition on BOTH dimensions





Where would (7,8) go?

Quadary Trees

“Binary” Search Tree

- Bin - Prefix meaning “2”
- Each node has (at most) 2 children

“Quadary” Search Tree

- Quad - Prefix meaning 4
- Each node has (at most) 4 children
- Usually say: “Quad-Tree” instead

Quad Trees - Find Node

```
1  public QuadNode findNode(QuadNode root, Integer x, Integer y) {
2    if (root == null || root.x == x && root.y == y) { return root; }
3
4    if (x < root.x) {
5      if (y < root.y) return findNode(root.llChild, x, y);
6      else            return findNode(root.lhChild, x, y);
7    } else {
8      if (y < root.y) return findNode(root.hlChild, x, y);
9      else            return findNode(root.hhChild, x, y);
10   }
11 }
```

Quad Trees - Find Node

```
1  public QuadNode findNode(QuadNode root, Integer x, Integer y) {
2    if (root == null || root.x == x && root.y == y) { return root; }
3
4    if (x < root.x) {
5      if (y < root.y) return findNode(root.llChild, x, y);
6      else            return findNode(root.lhChild, x, y);
7    } else {
8      if (y < root.y) return findNode(root.hlChild, x, y);
9      else            return findNode(root.hhChild, x, y);
10   }
11 }
```

Complexity?

Quad Trees - Find Node

```
1  public QuadNode findNode(QuadNode root, Integer x, Integer y) {
2    if (root == null || root.x == x && root.y == y) { return root; }
3
4    if (x < root.x) {
5      if (y < root.y) return findNode(root.llChild, x, y);
6      else            return findNode(root.lhChild, x, y);
7    } else {
8      if (y < root.y) return findNode(root.hlChild, x, y);
9      else            return findNode(root.hhChild, x, y);
10   }
11 }
```

Complexity? $O(d)$

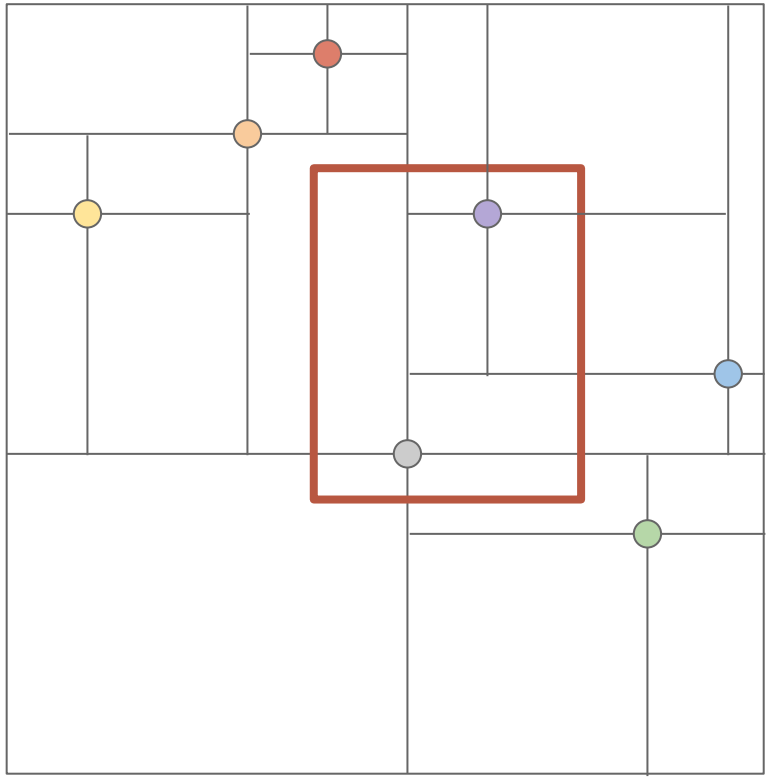
Quad Trees - Other Operations

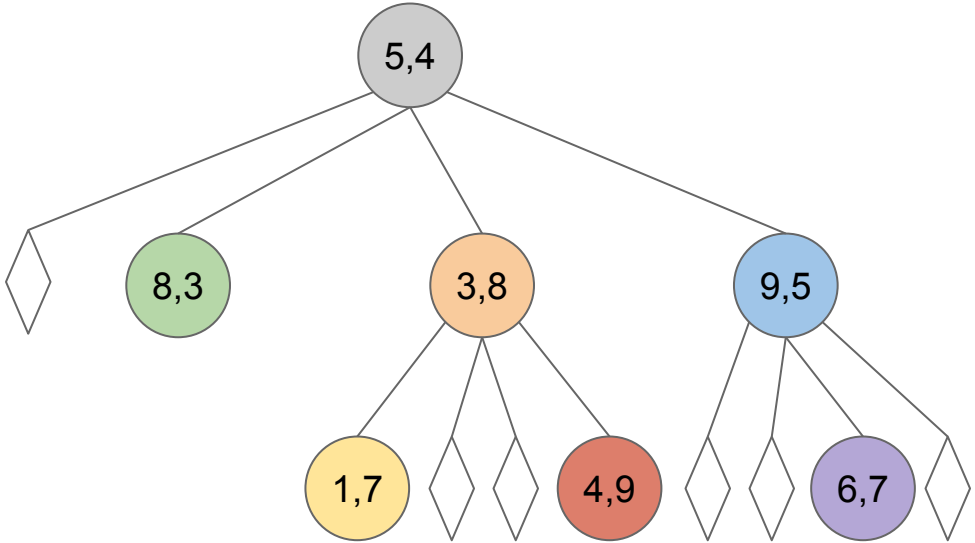
What if I want to find a range of points instead of just one point?

`range(xlow, xhigh, ylow, yhigh)`

- …?

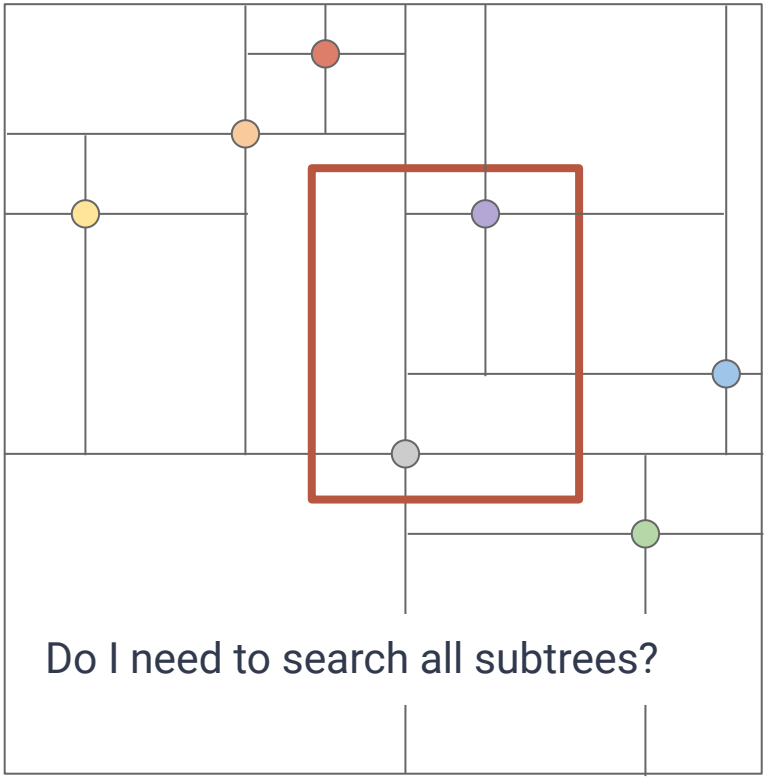
Attempt 1 - Partition on BOTH dimensions

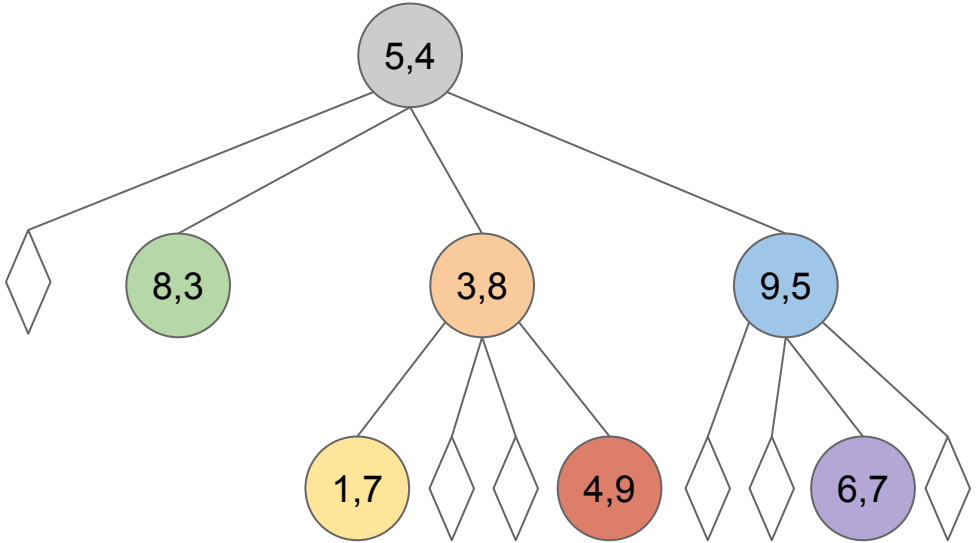




`range(4,7,4.5,7.5)?`

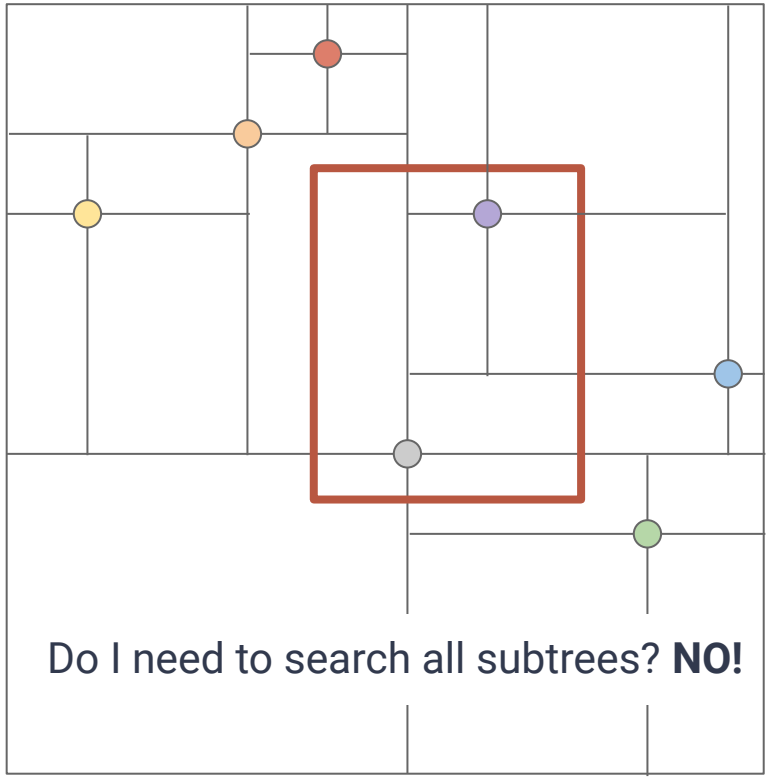
Attempt 1 - Partition on BOTH dimensions



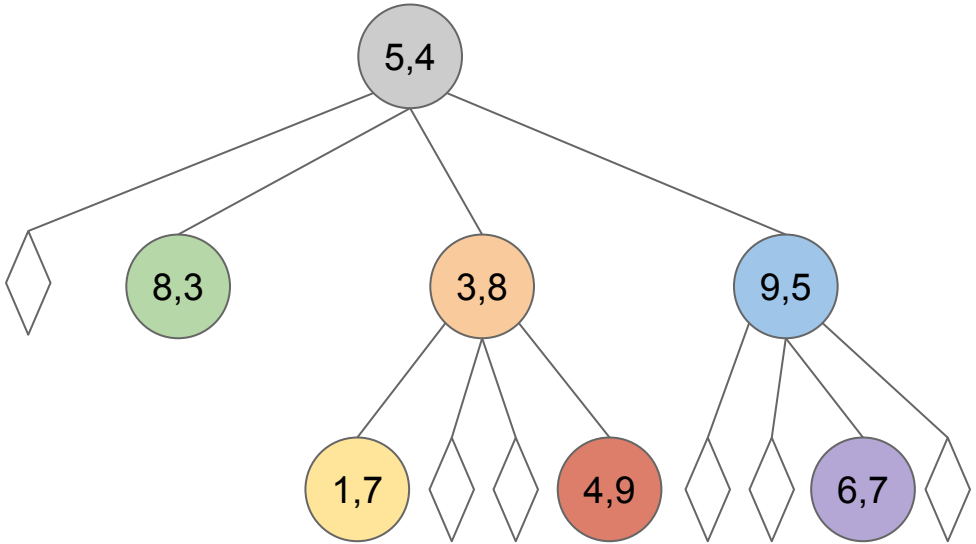


`range(4,7,4.5,7.5)`?

Attempt 1 - Partition on BOTH dimensions

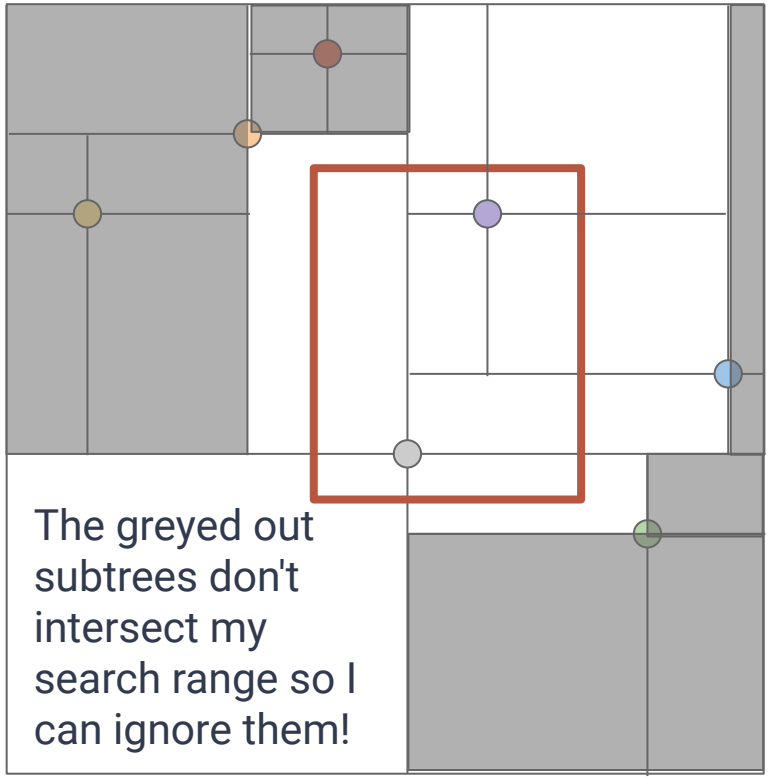


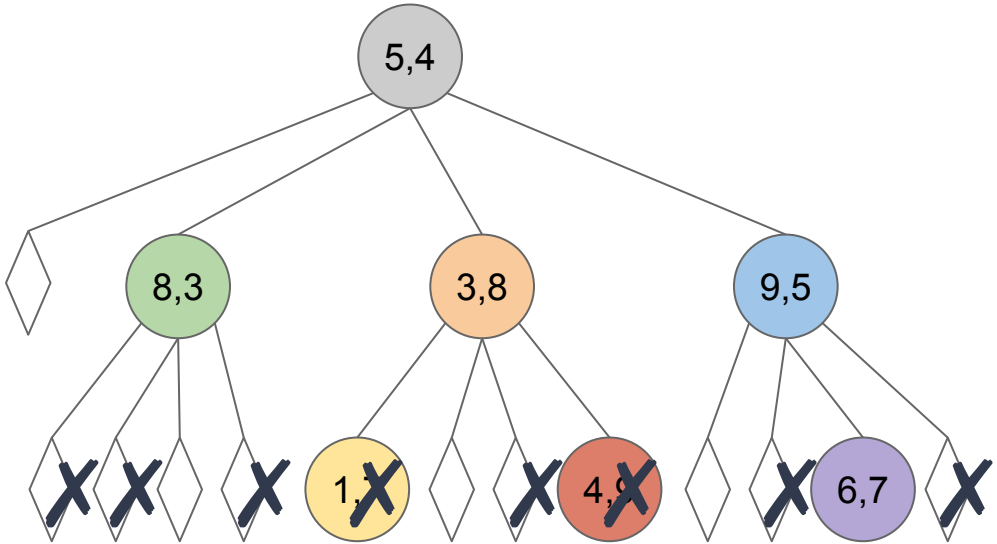
Do I need to search all subtrees? **NO!**



`range(4,7,4.5,7.5)`?

Attempt 1 - Partition on BOTH dimensions





`range(4,7,4.5,7.5)?`

Quad Trees - Range

```
1  public void range(QuadNode root, Rectangle target, List<QuadNode> list) {
2    if (root == null || !target.intersects(root.region)) { return; }
3    if (target.contains(root.x, root.y)) { list.add(root); }
4
5    range(root.llChild, target, list);
6    range(root.lhChild, target, list);
7    range(root.hlChild, target, list);
8    range(root.hhChild, target, list);
9  }
```

Quad Trees - Range

```
1  public void range(QuadNode root, Rectangle target, List<QuadNode> list) {
2    if (root == null || !target.intersects(root.region)) { return; }
3    if (target.contains(root.x, root.y)) { list.add(root); }
4
5    range(root.llChild, target, list);
6    range(root.lhChild, target, list);
7    range(root.hlChild, target, list);
8    range(root.hhChild, target, list);
9  }
```

If root does not exist, or the region it belongs to does not intersect our target area, we can ignore it!

The region a node belongs to can be set when the node is inserted into the tree

Quad Trees - Range

```
1  public void range(QuadNode root, Rectangle target, List<QuadNode> list) {
2    if (root == null || !target.intersects(root.region)) { return; }
3    if (target.contains(root.x, root.y)) { list.add(root); }
4
5    range(root.llChild, target, list);
6    range(root.lhChild, target, list);
7    range(root.hlChild, target, list);
8    range(root.hhChild, target, list);
9  }
```

Otherwise…if the root is in the target region, add it to the list of nodes and…

The region a node belongs to can be set when the node is inserted into the tree

Quad Trees - Range

```
1  public void range(QuadNode root, Rectangle target, List<QuadNode> list) {
2    if (root == null || !target.intersects(root.region)) { return; }
3    if (target.contains(root.x, root.y)) { list.add(root); }
4
5    range(root.llChild, target, list);
6    range(root.lhChild, target, list);
7    range(root.hlChild, target, list);
8    range(root.hhChild, target, list);
9  }
```

…recursively explore it's children

The region a node belongs to can be set when the node is inserted into the tree

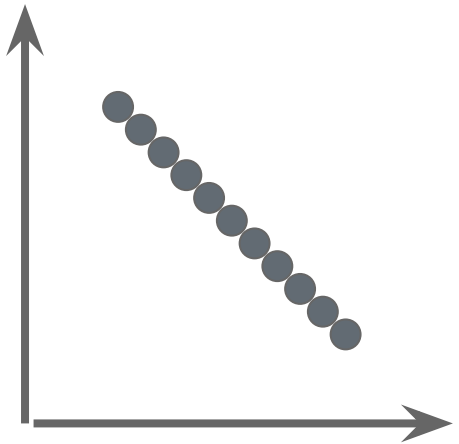
Quad Trees - Challenges

Quad Trees - Challenges

Creating a balanced Quad Tree is hard

- Impossible to always split elements evenly across all four subtrees
  (though depth = O(log(n)) still possible)

Worst Case:
No possible way to create nodes with >2 nonempty subtrees



Quad Trees - Challenges

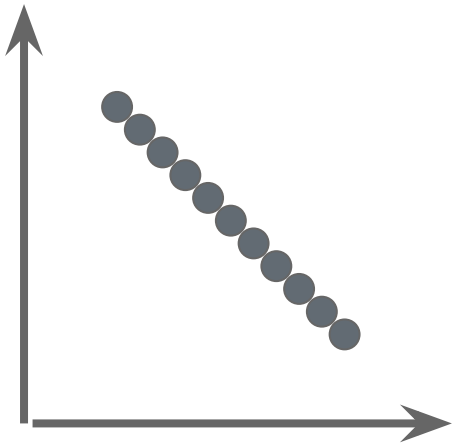
Creating a balanced Quad Tree is hard

- Impossible to always split elements evenly across all four subtrees
  (though depth = O(log(n)) still possible)

Keeping the quad tree balanced after updates is significantly harder

- No “simple” analog for rotate left/right.

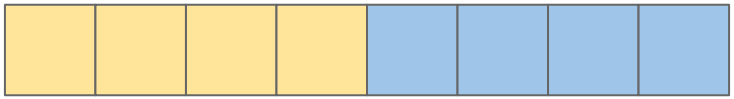
Worst Case:
No possible way to create nodes with >2 nonempty subtrees



Quad Trees - Challenges

Problem: Every node has 4 children!

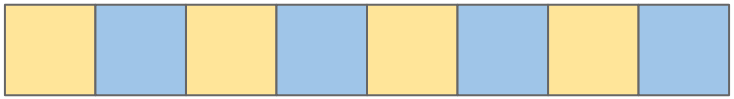
Revisiting Lexical Order



Problem: Searches on lexical order partitions all of one dimension first

(ie **fully** partitions based on the yellow dimension first, then blue)

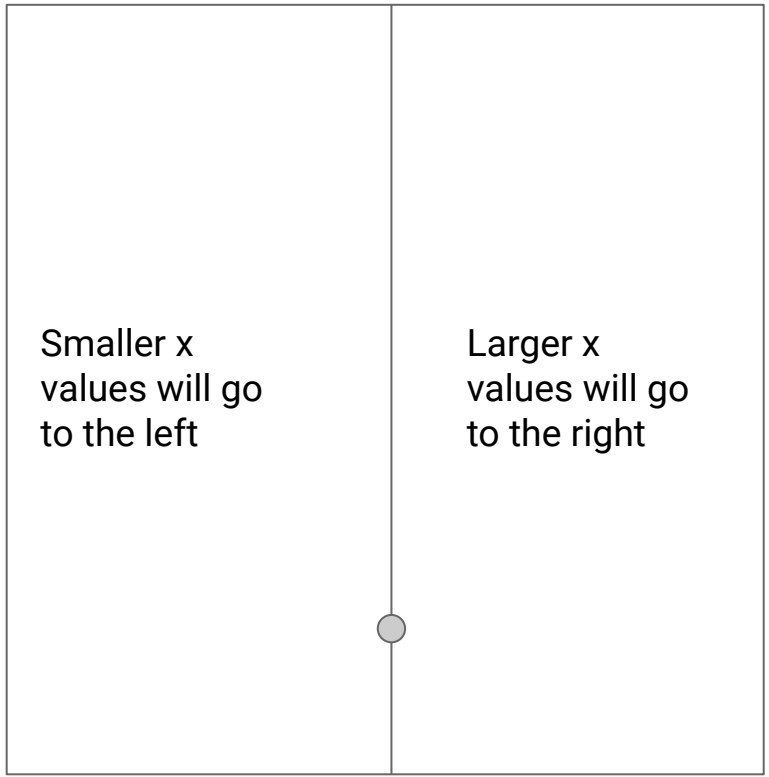
Revisiting Lexical Order



Idea: Alternate dimensions

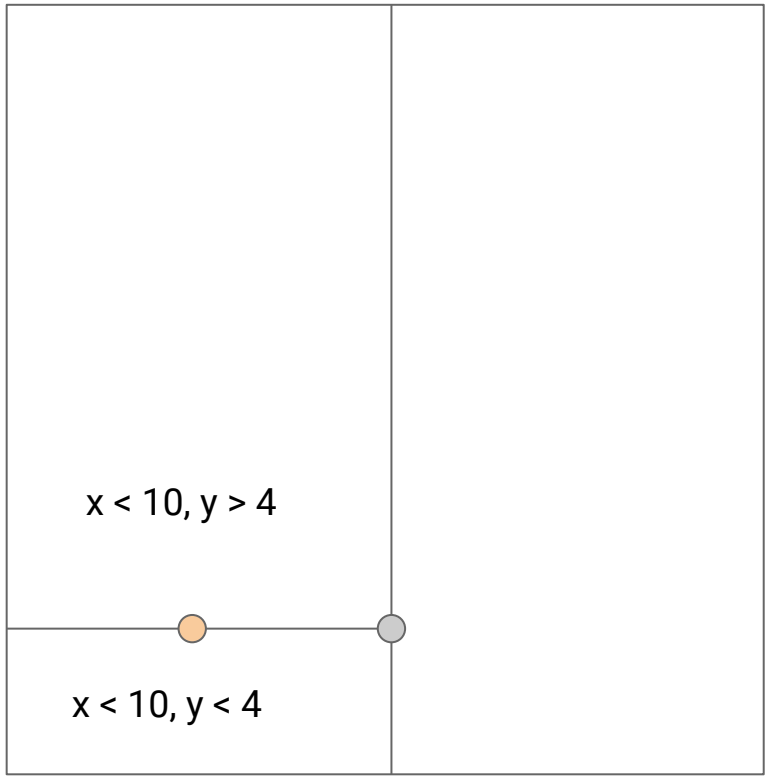
(ie partition a little bit on yellow dimension, then a little bit on blue, then a little on yellow, etc…)

k-D Tree Example

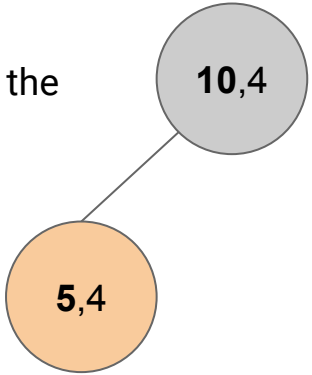


Nodes at level 1 will partition on the first dimension, x

k-D Tree Example - insert(5,4)

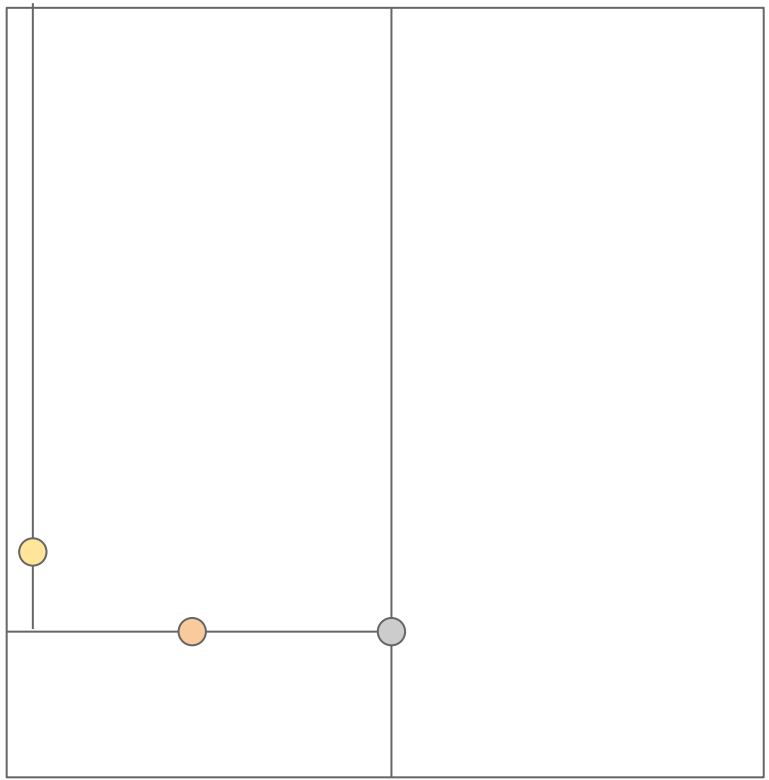


5 < 10, so insert into the left subtree

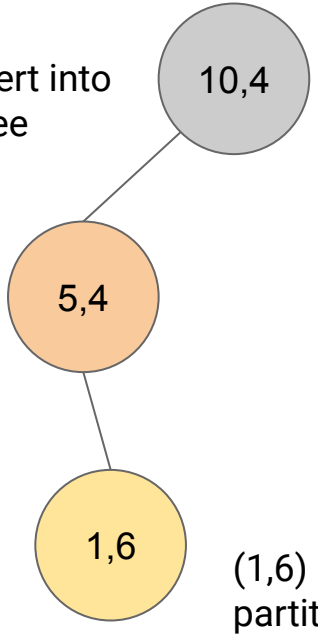


(5,4) is at level 2 so it will partition it's children on the second dimension, y

k-D Tree Example - insert(1,6)



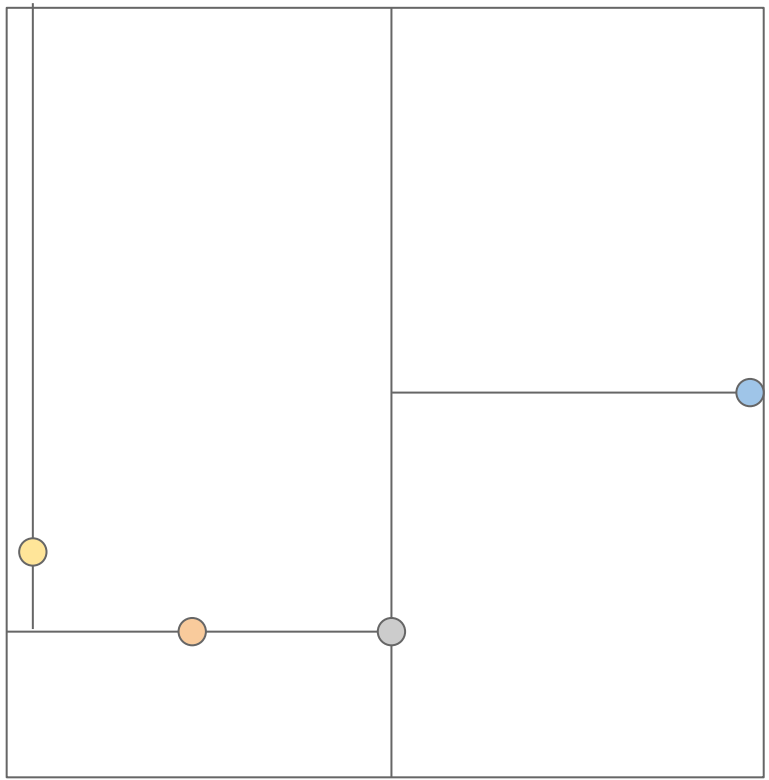
1 < 10, so insert into the left subtree

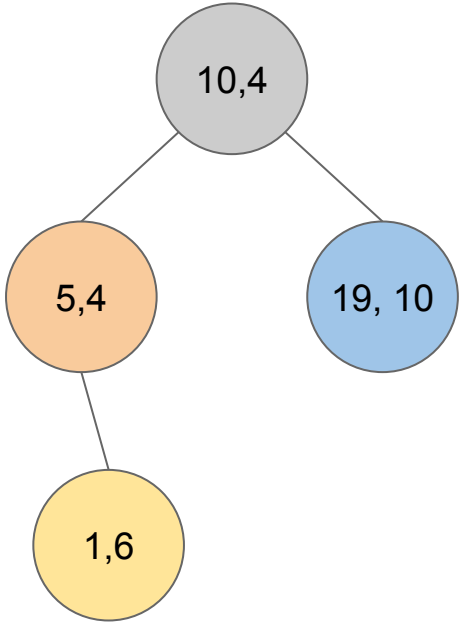


6 > 4 so insert into the right subtree

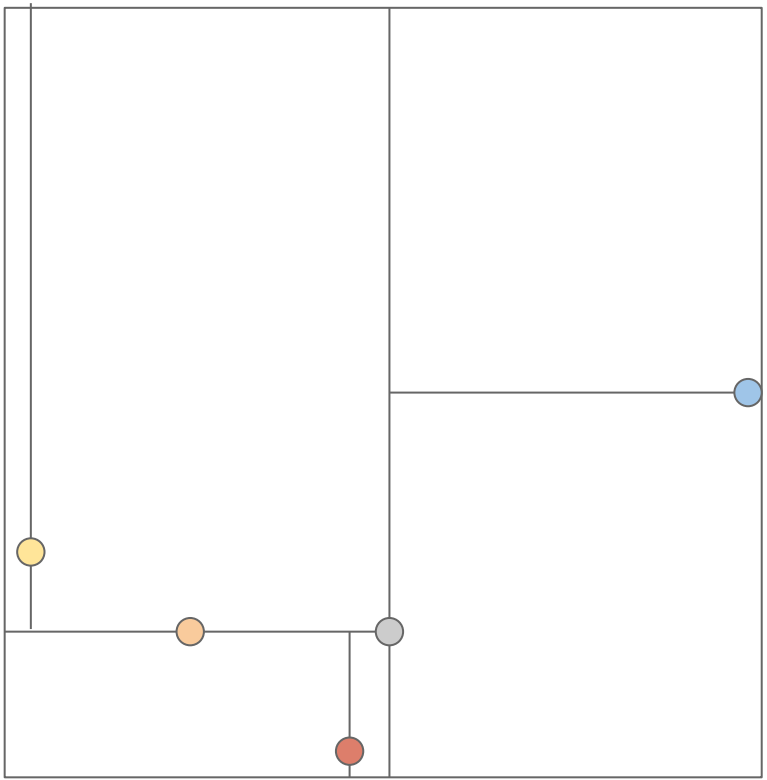
(1,6) is at level 3 so it will partition it's children on the first dimension, x

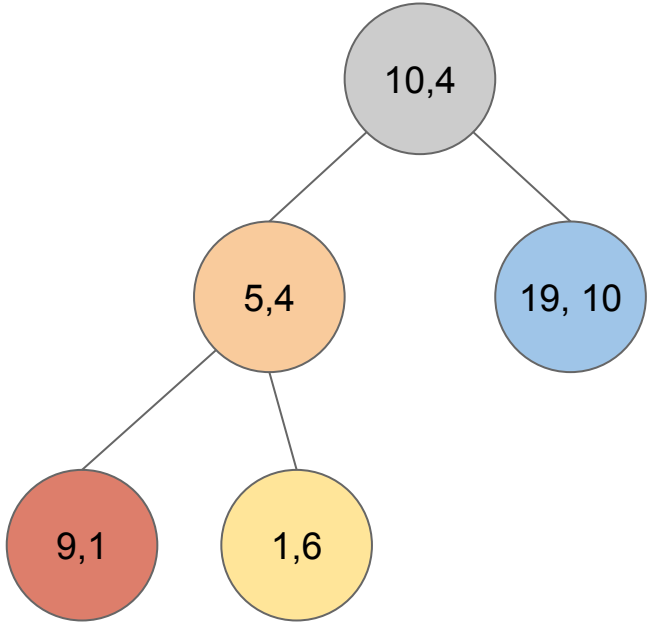
k-D Tree Example - insert(19,10)



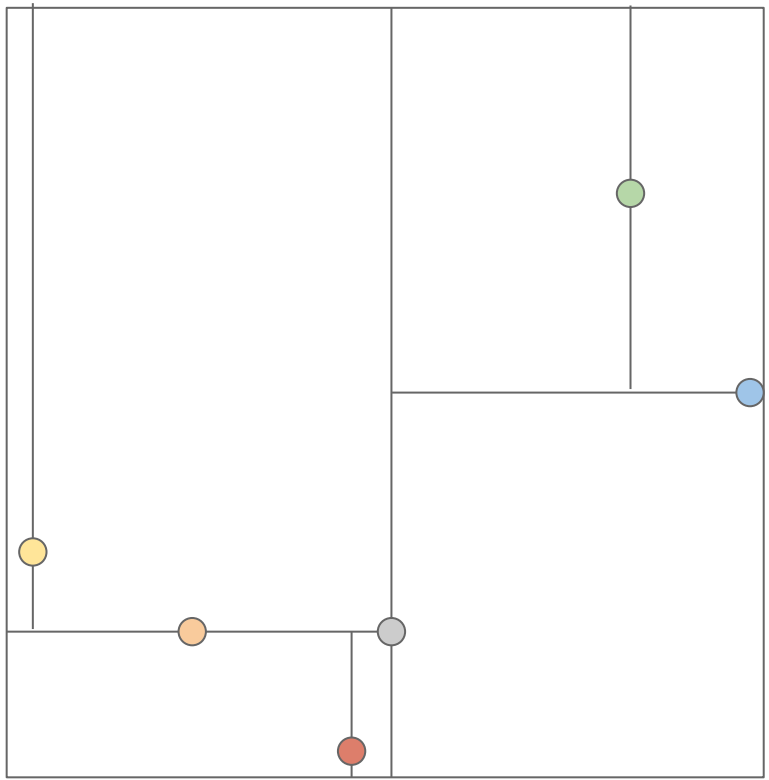


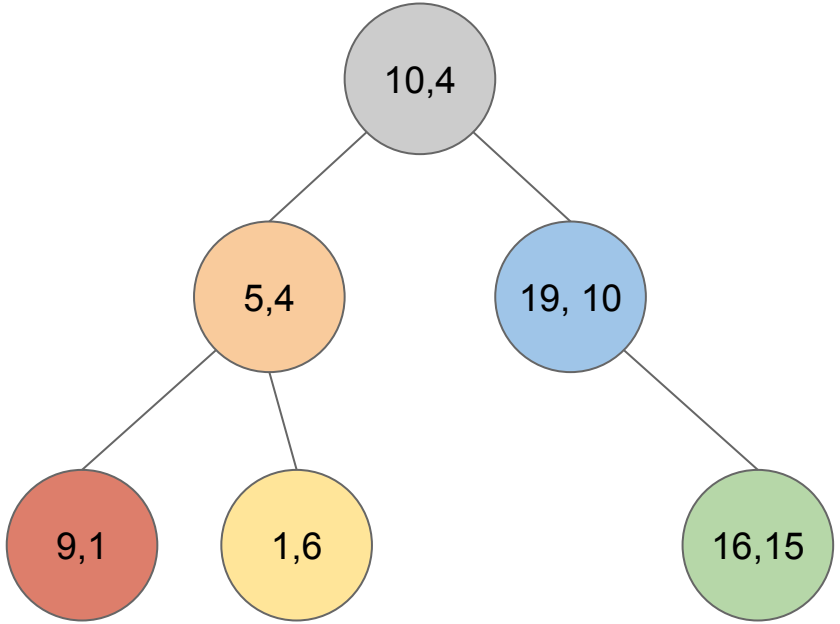
k-D Tree Example - insert(9,1)



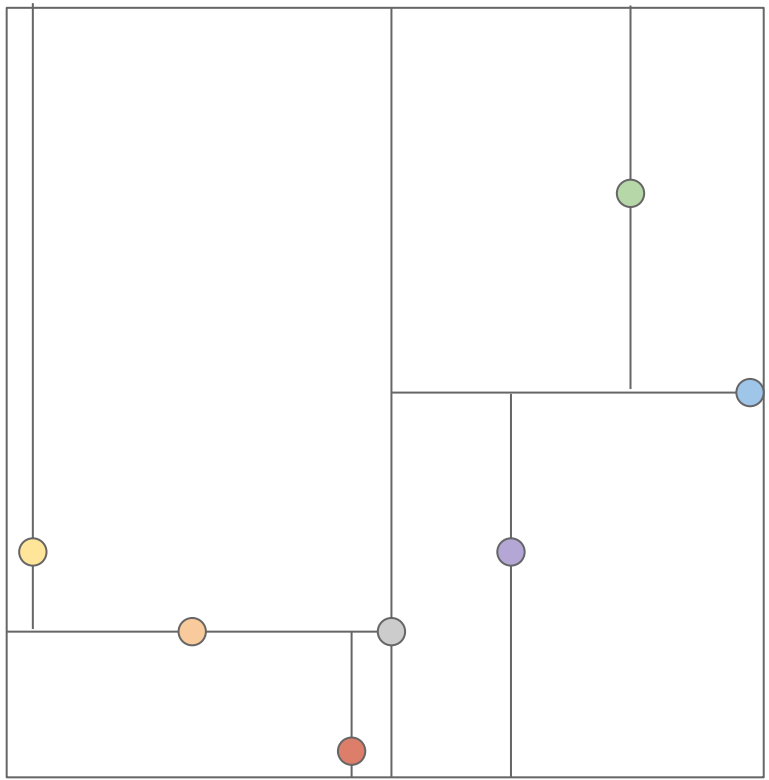


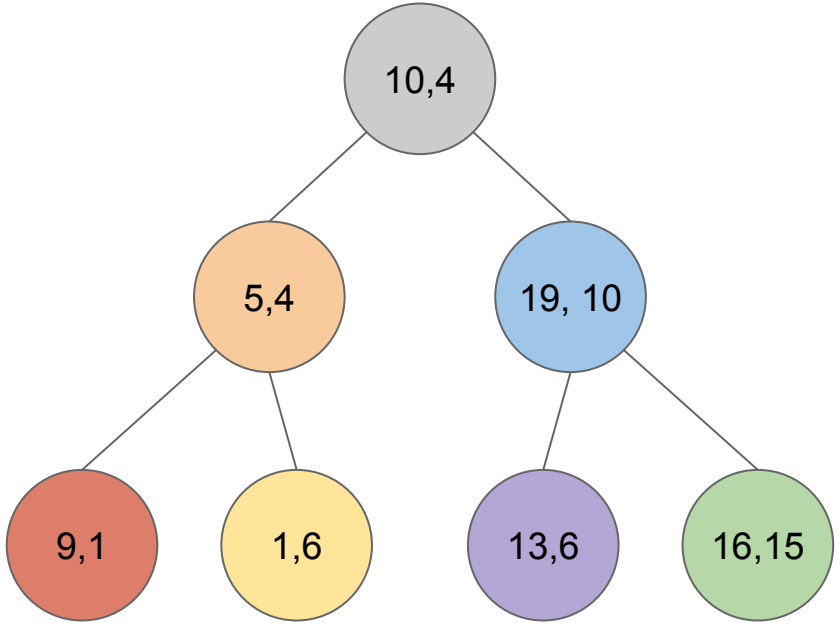
k-D Tree Example - insert(16,15)





k-D Tree Example - insert(13,6)





```
1  public KDNode findNode(KDNode root, Integer x, Integer y) {
2    KDNode current = root;
3    Integer depth = 0;
4    while (current != null && (current.x != x || current.y != y)) {
5      if(depth % 2 == 0) {
6        if(x < current.x) { current = current.left; }
7        else              { current = current.right; }
8      } else {
9        if(y < current.y) { current = current.left; }
10       else              { current = current.right; }
11     }
12     depth += 1;
13   }
14   return current
15 }
```

k-D Trees - Find Node

```
1  public KDNode findNode(KDNode root, Integer x, Integer y) {
2    KDNode current = root;
3    Integer depth = 0;
4    while (current != null && (current.x != x || current.y != y)) {
5      if(depth % 2 == 0) {
6        if(x < current.x) { current = current.left; }
7        else              { current = current.right; }
8      } else {
9        if(y < current.y) { current = current.left; }
10       else              { current = current.right; }
11     }
12     depth += 1;
13   }
14   return current
15 }
```

If depth is even, act like a BST that partitions on x

k-D Trees - Find Node

```
1  public KDNode findNode(KDNode root, Integer x, Integer y) {
2    KDNode current = root;
3    Integer depth = 0;
4    while (current != null && (current.x != x || current.y != y)) {
5      if(depth % 2 == 0) {
6        if(x < current.x) { current = current.left; }
7        else              { current = current.right; }
8      } else {
9        if(y < current.y) { current = current.left; }
10       else              { current = current.right; }
11     }
12     depth += 1;
13   }
14   return current
15 }
```

Complexity?

k-D Trees - Find Node

```
1  public KDNode findNode(KDNode root, Integer x, Integer y) {
2    KDNode current = root;
3    Integer depth = 0;
4    while (current != null && (current.x != x || current.y != y)) {
5      if(depth % 2 == 0) {
6        if(x < current.x) { current = current.left; }
7        else              { current = current.right; }
8      } else {
9        if(y < current.y) { current = current.left; }
10       else              { current = current.right; }
11     }
12     depth += 1;
13   }
14   return current
15 }
```

Complexity? ***O(d)***

k-D Trees - Find Node

Nearest Neighbor

What if we want to find the closest point to our target?

Nearest Neighbor

What if we want to find the closest point to our target?

Problem: Can't just do normal find; the target may not be in the tree at all

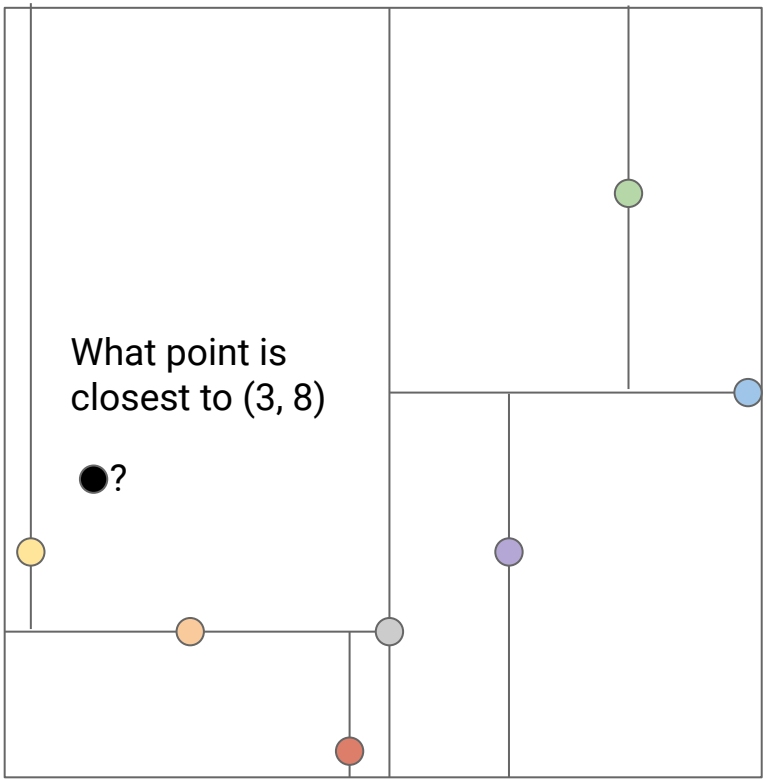
Nearest Neighbor

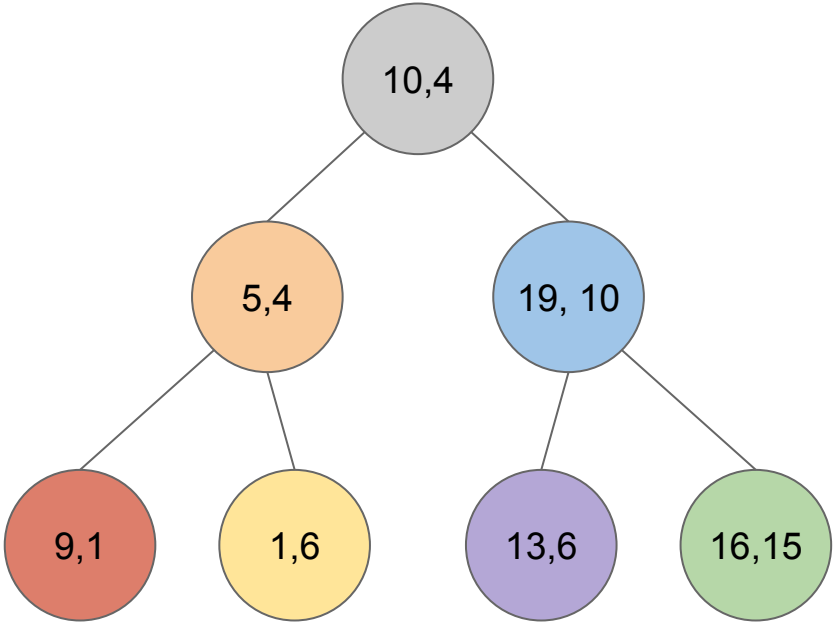
What if we want to find the closest point to our target?

Problem: Can't just do normal find; the target may not be in the tree at all

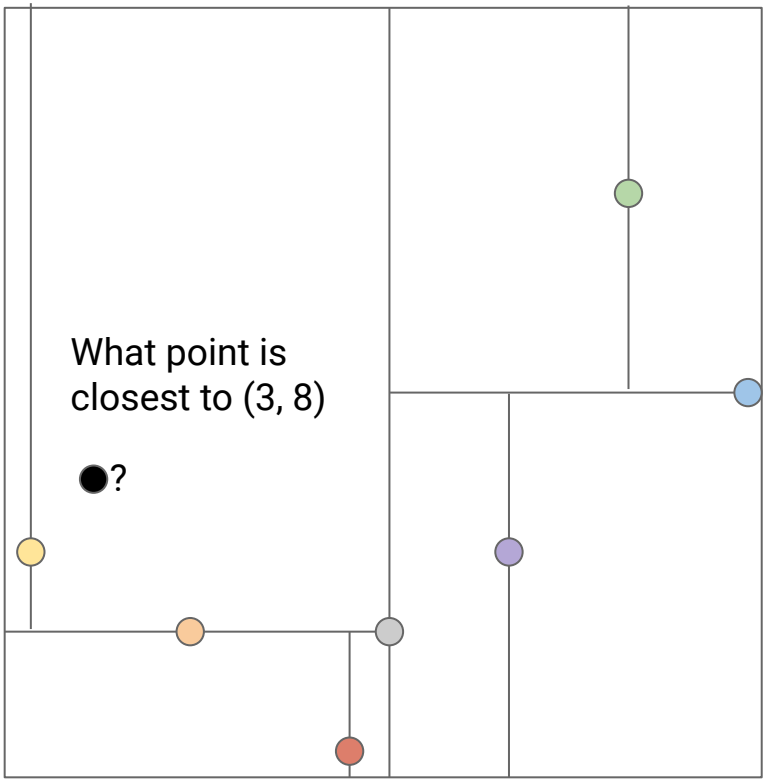
Idea: Search like normal until we hit a leaf, then go back up the tree and see if there's a possibility we missed something.

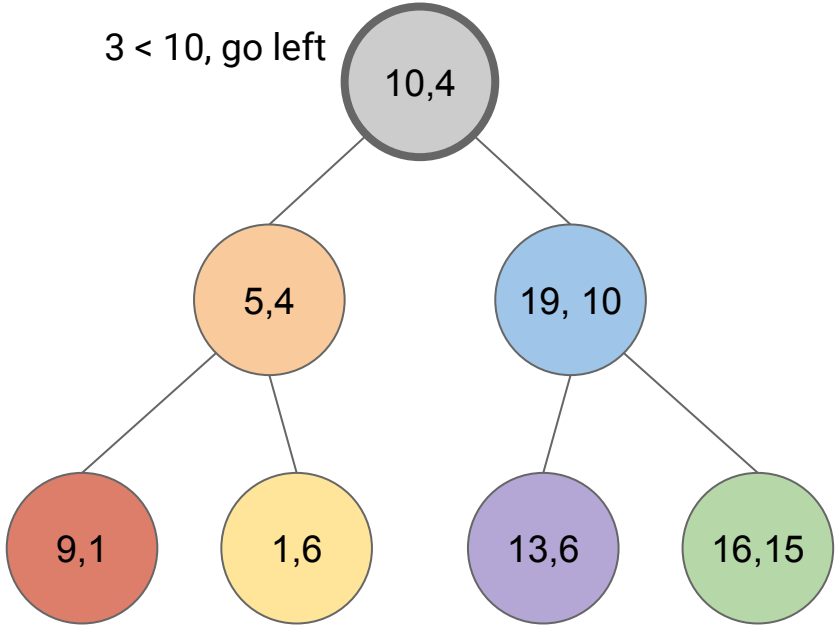
Nearest Neighbor - Example 1



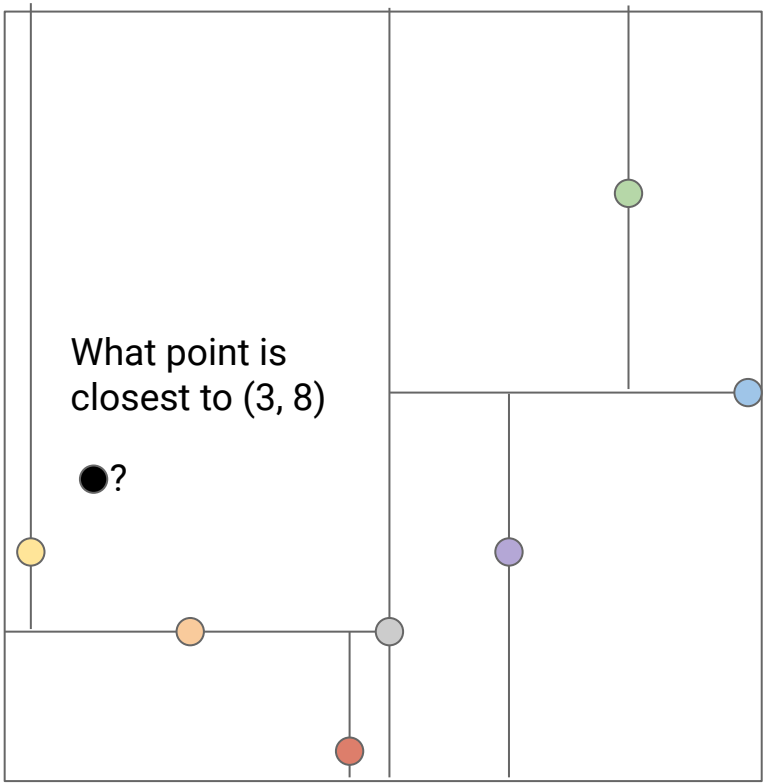


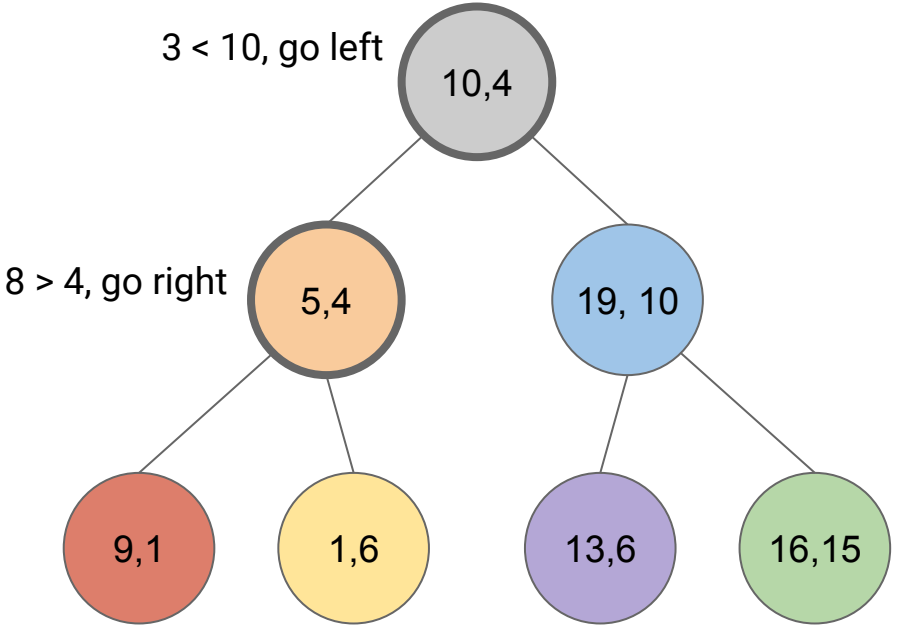
Nearest Neighbor - Example 1



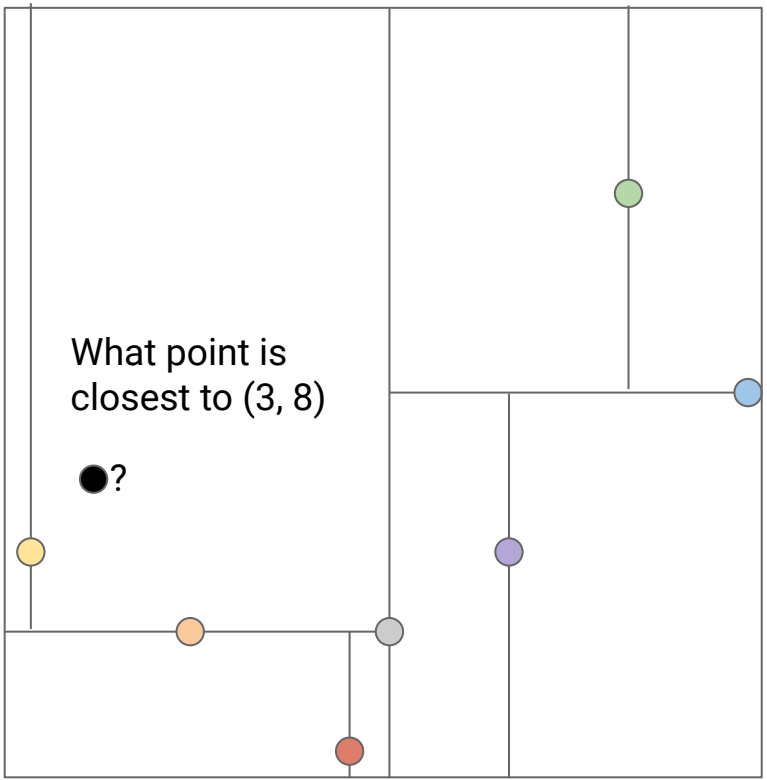


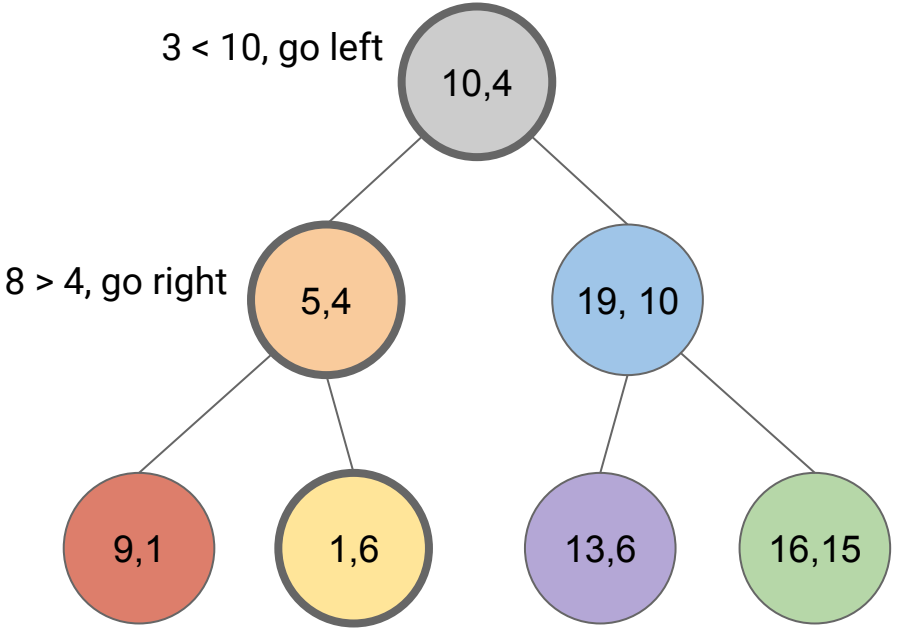
Nearest Neighbor - Example 1





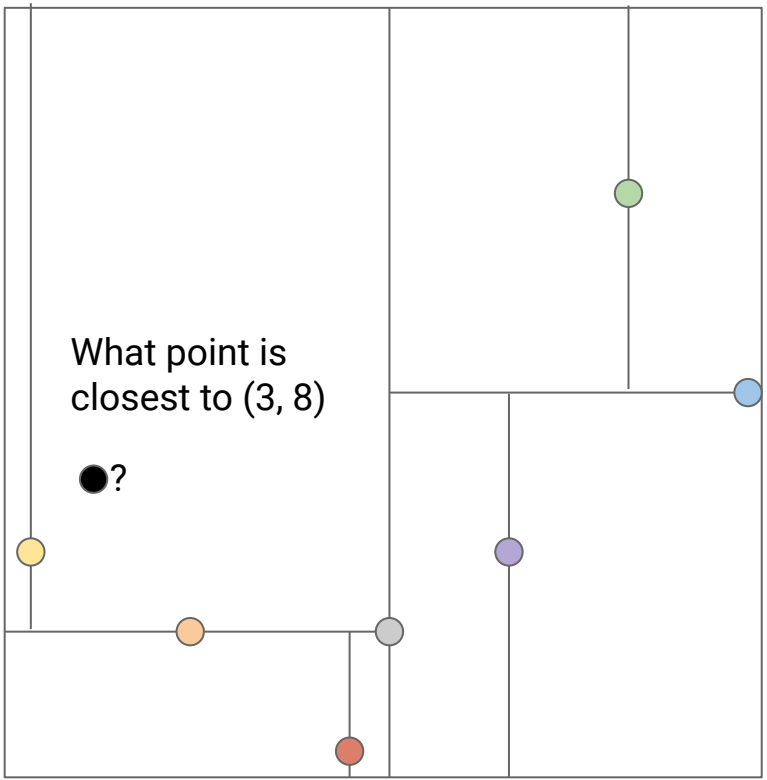
Nearest Neighbor - Example 1



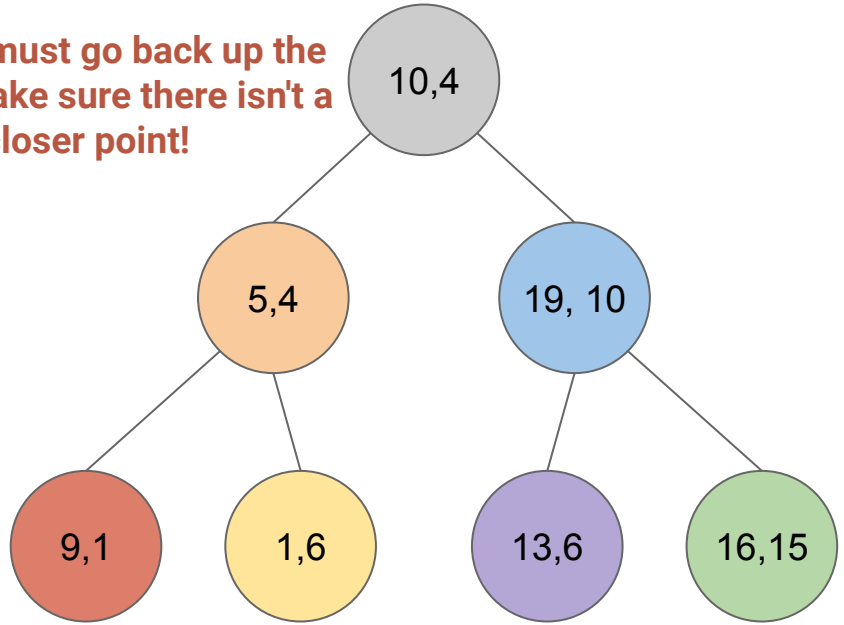


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

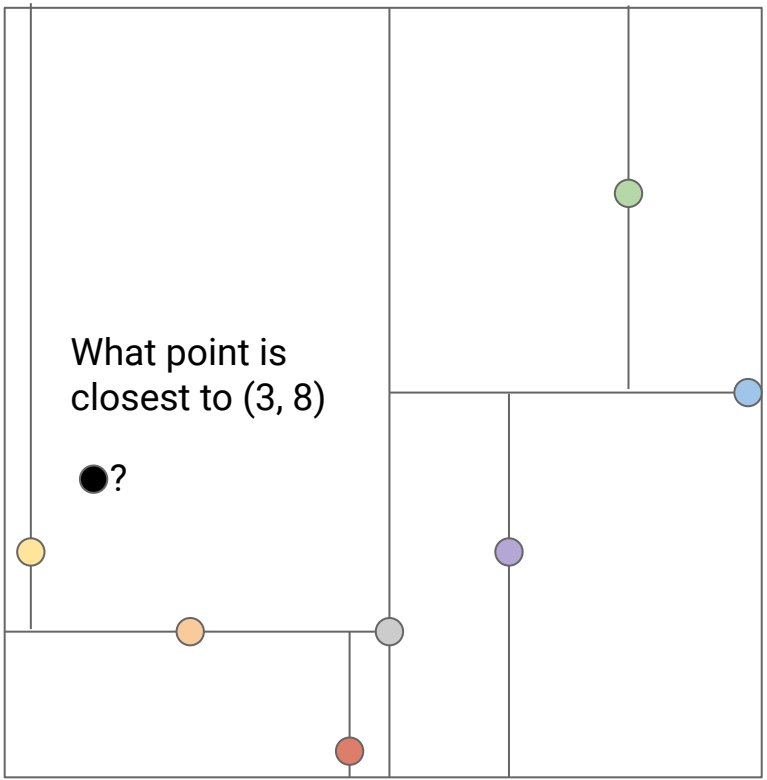


Now we must go back up the tree to make sure there isn't a closer point!

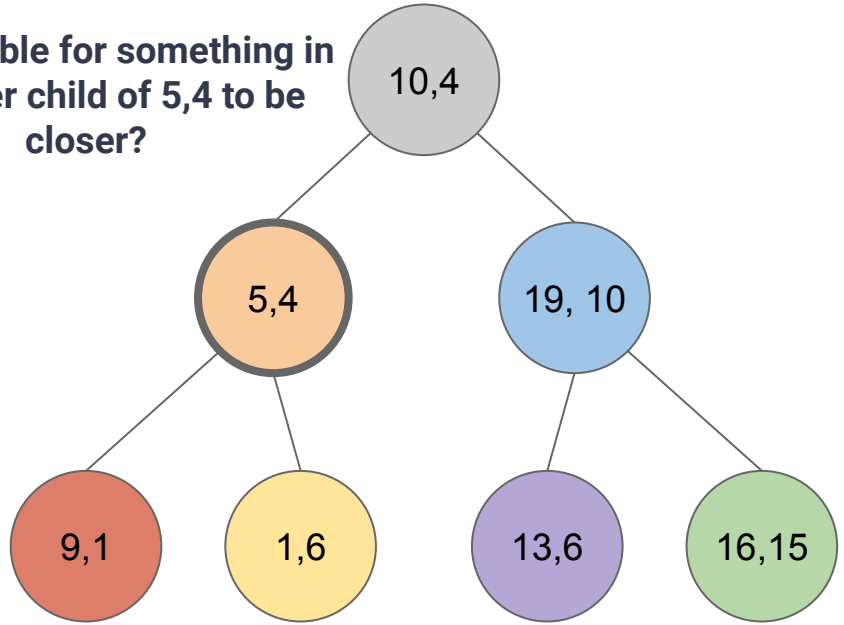


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

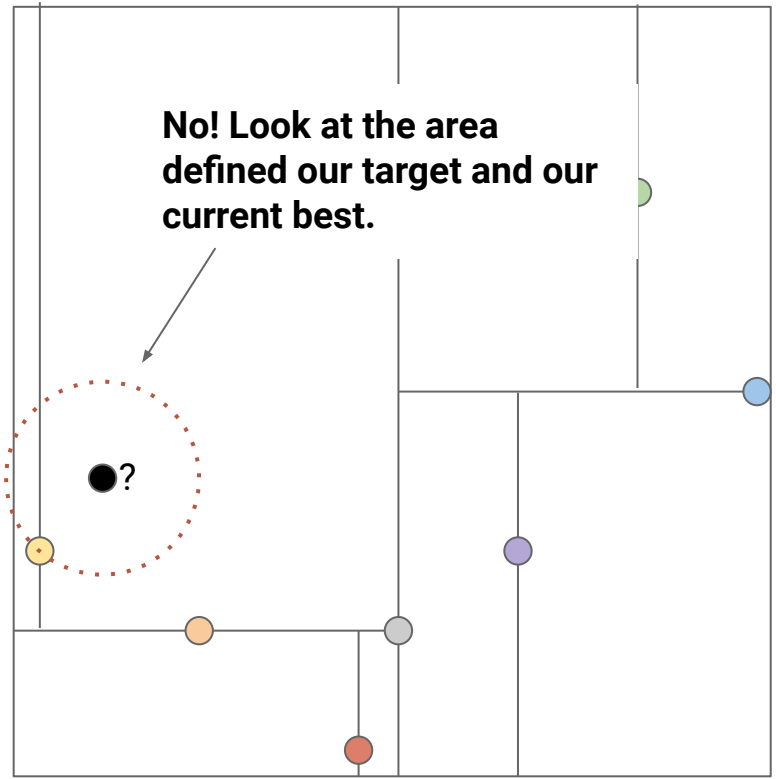


Is it possible for something in the other child of 5,4 to be closer?

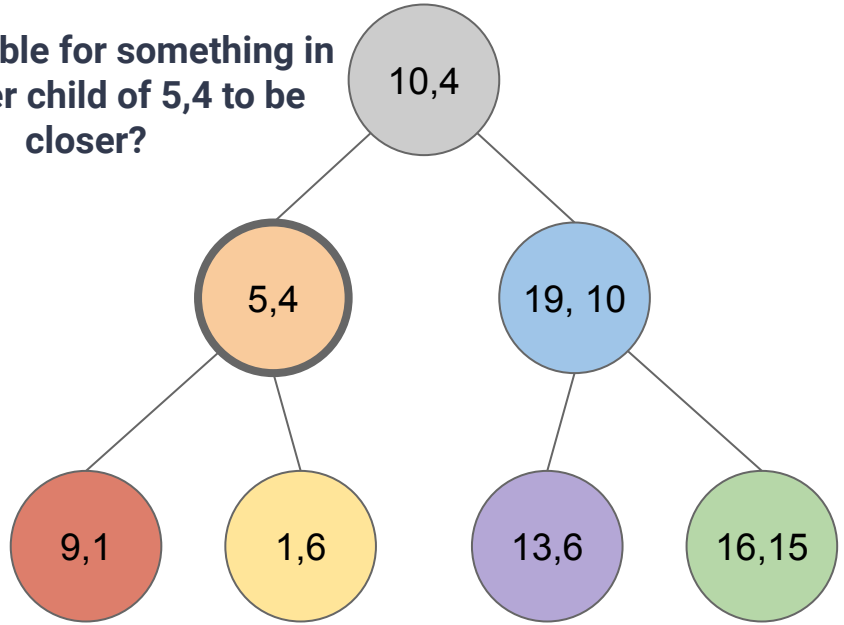


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

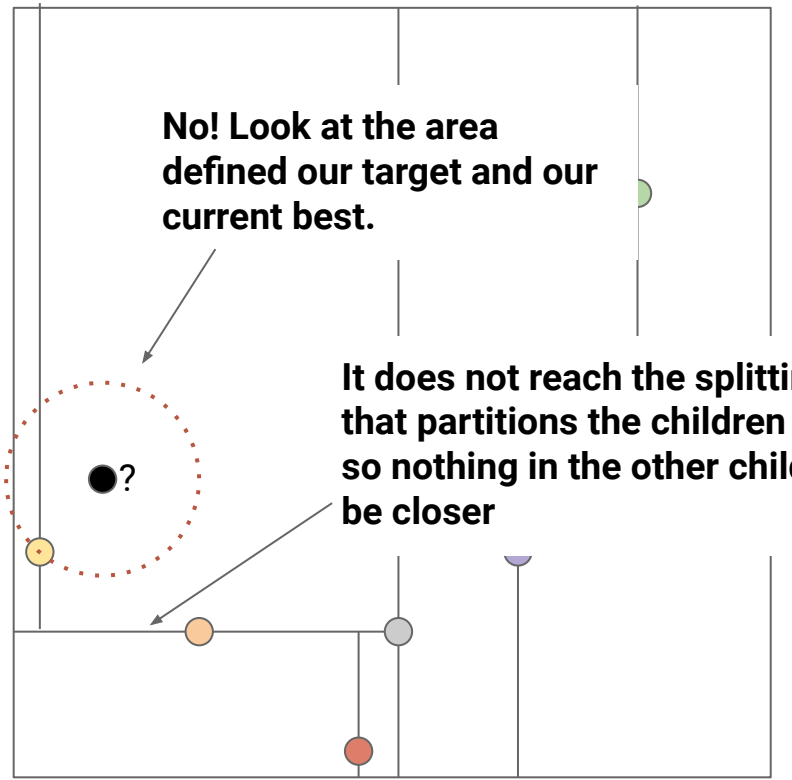


Is it possible for something in the other child of 5,4 to be closer?

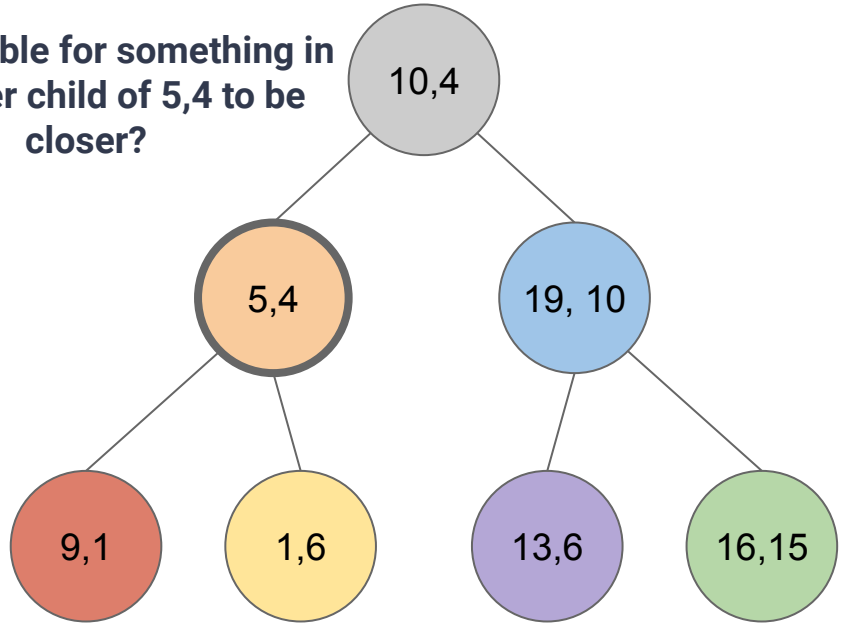


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

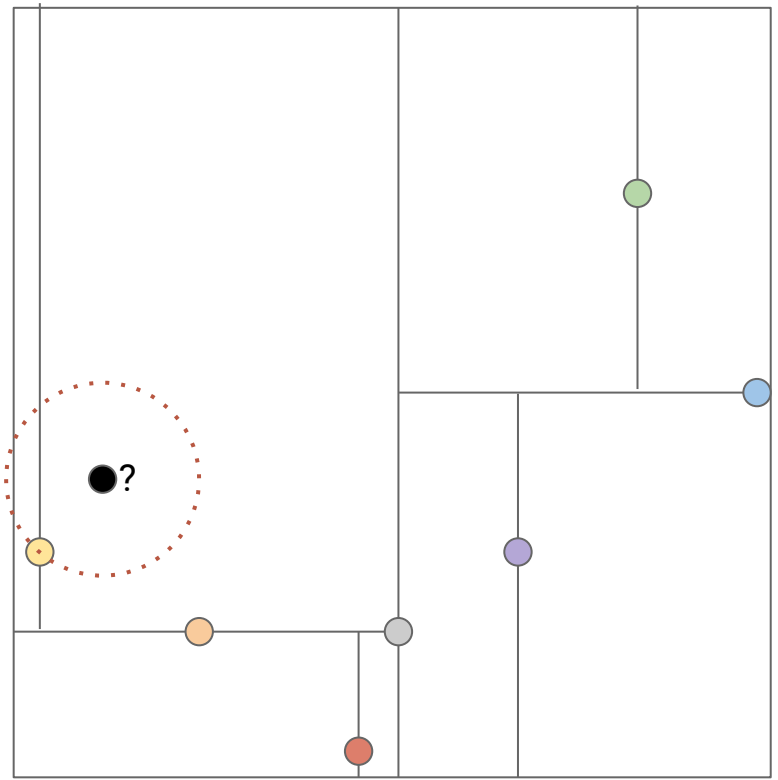


Is it possible for something in the other child of 5,4 to be closer?

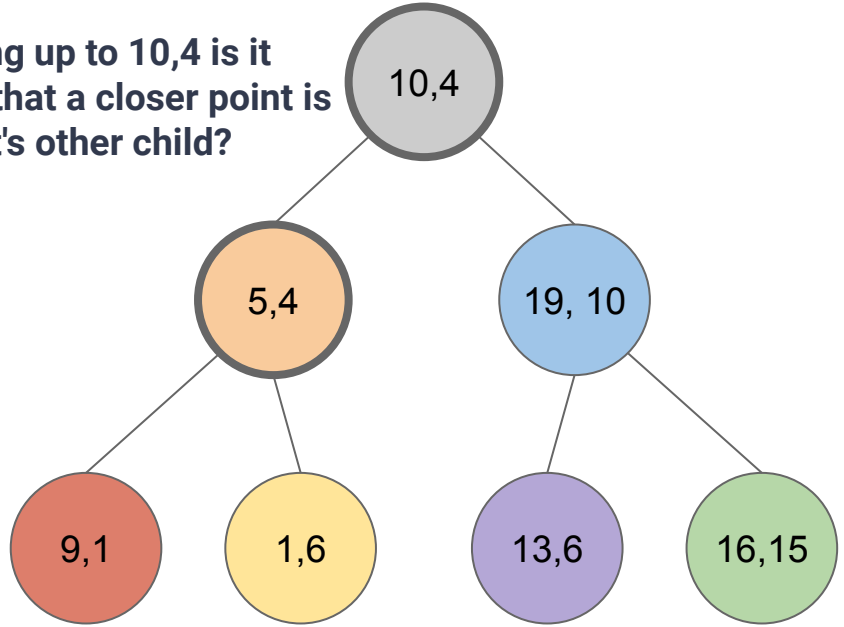


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

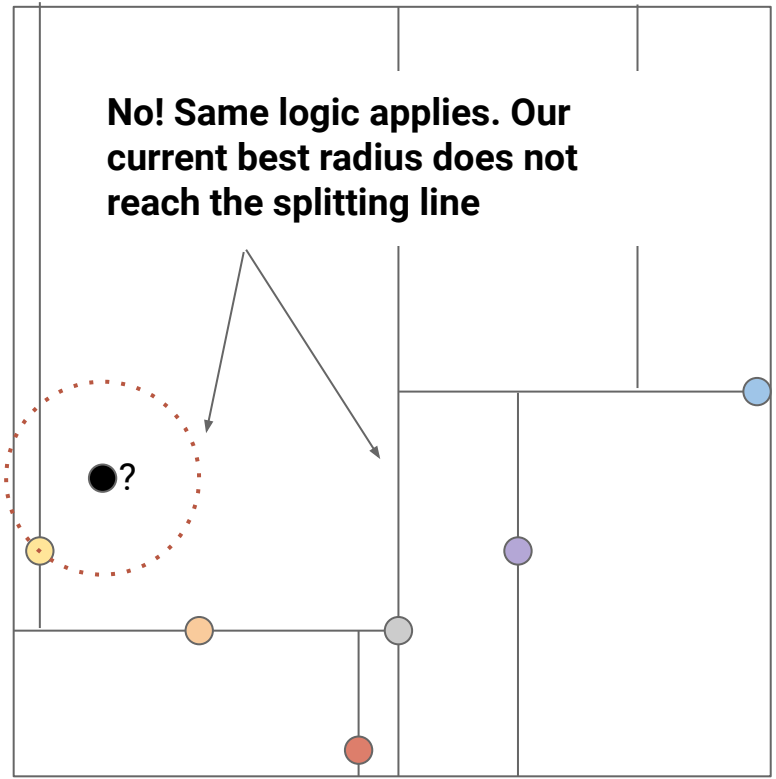


Moving up to 10,4 is it possible that a closer point is in it's other child?

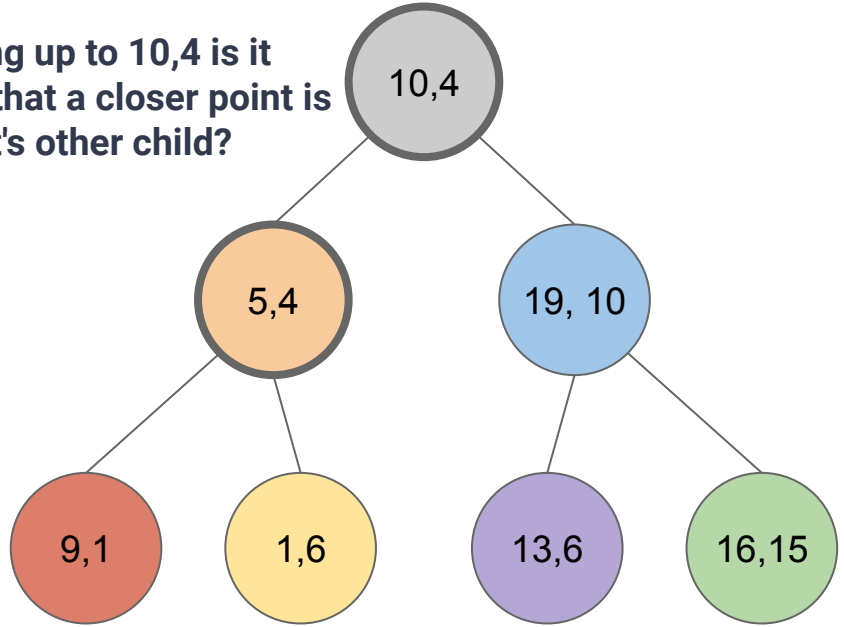


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

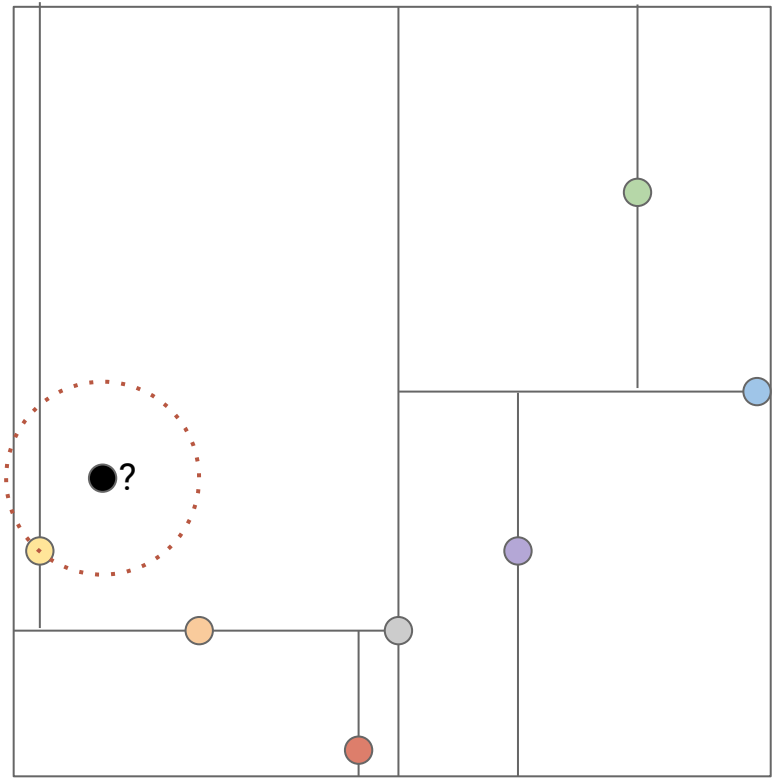


Moving up to 10,4 is it possible that a closer point is in it's other child?

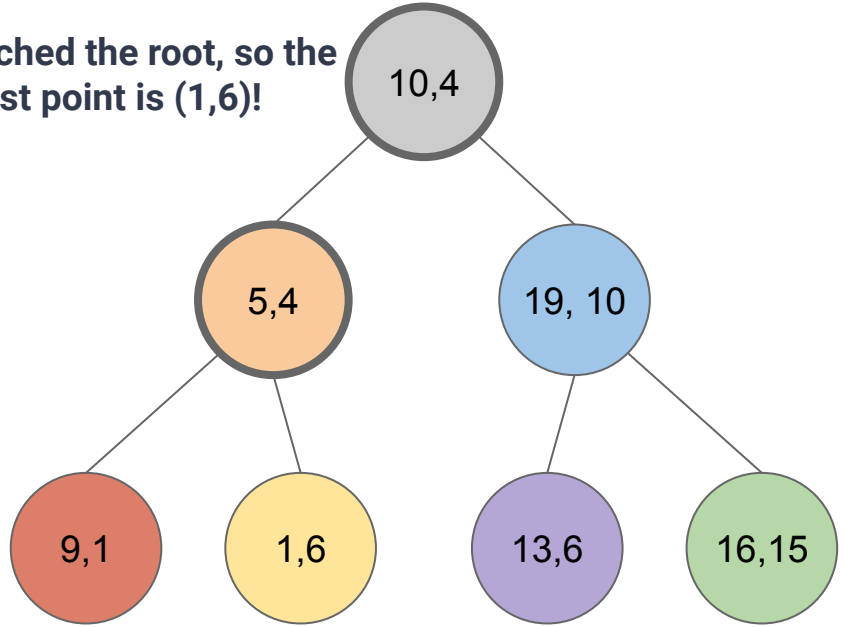


Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 1

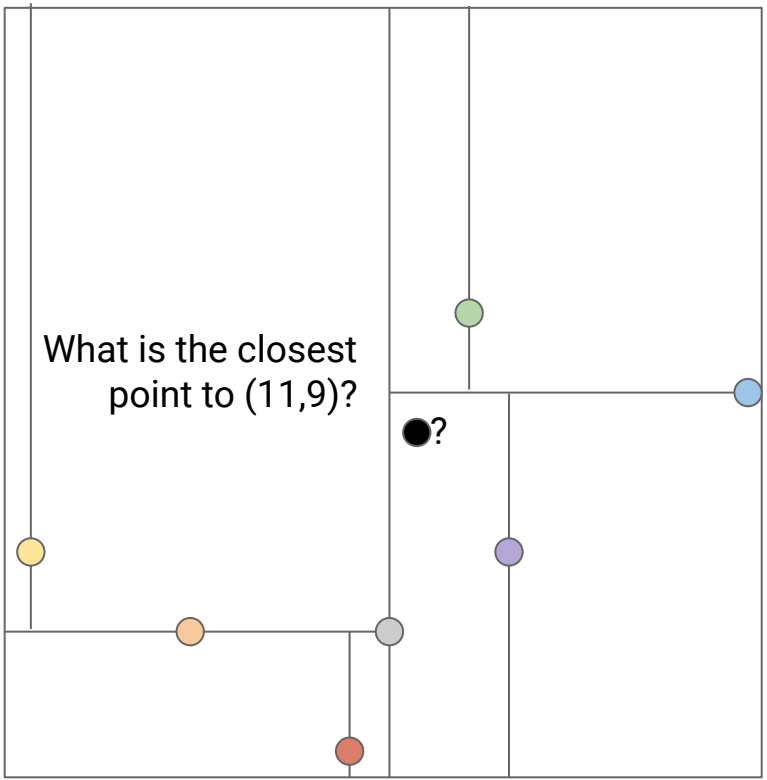


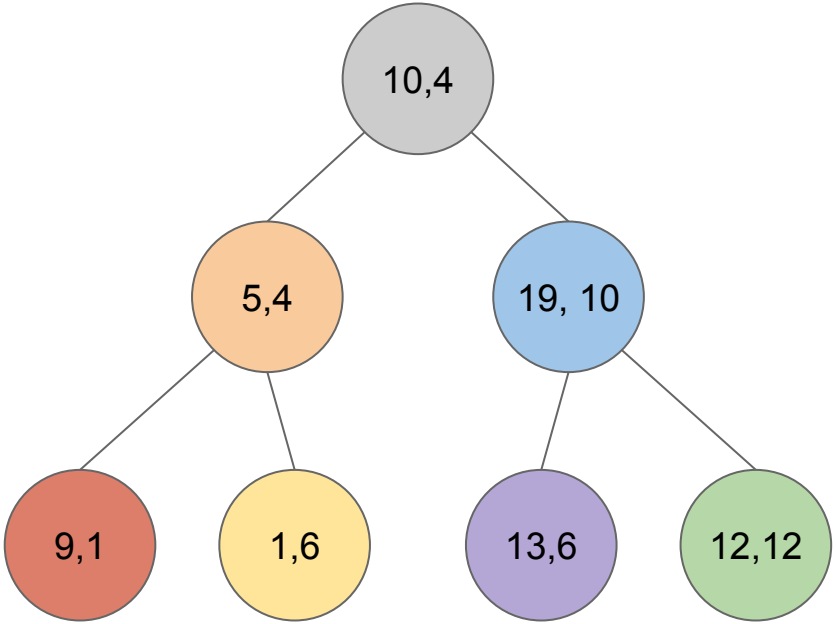
We've reached the root, so the closest point is (1,6)!



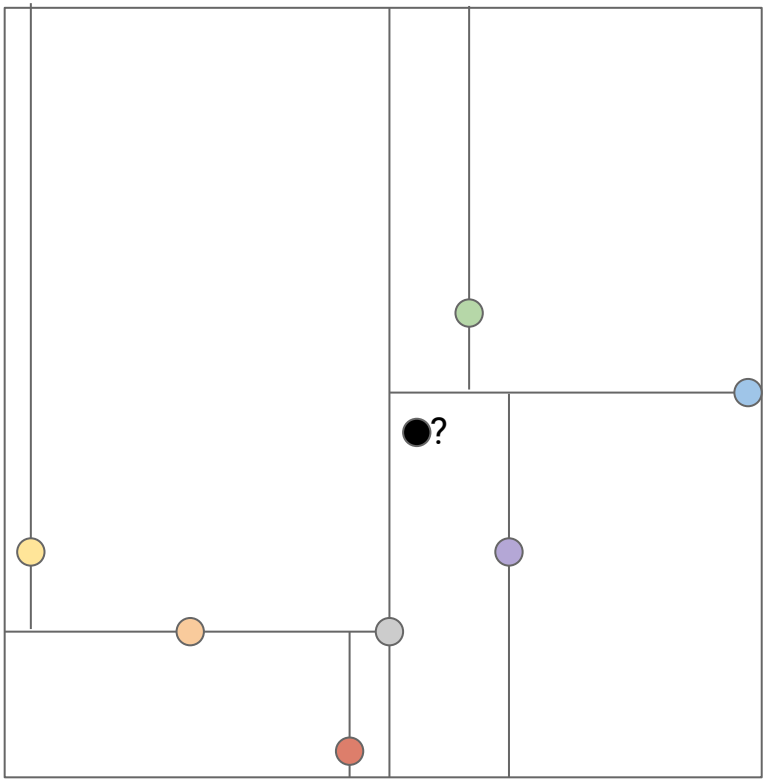
Found a leaf! This is our "closest so far"
It is 2.828 units away from our target

Nearest Neighbor - Example 2

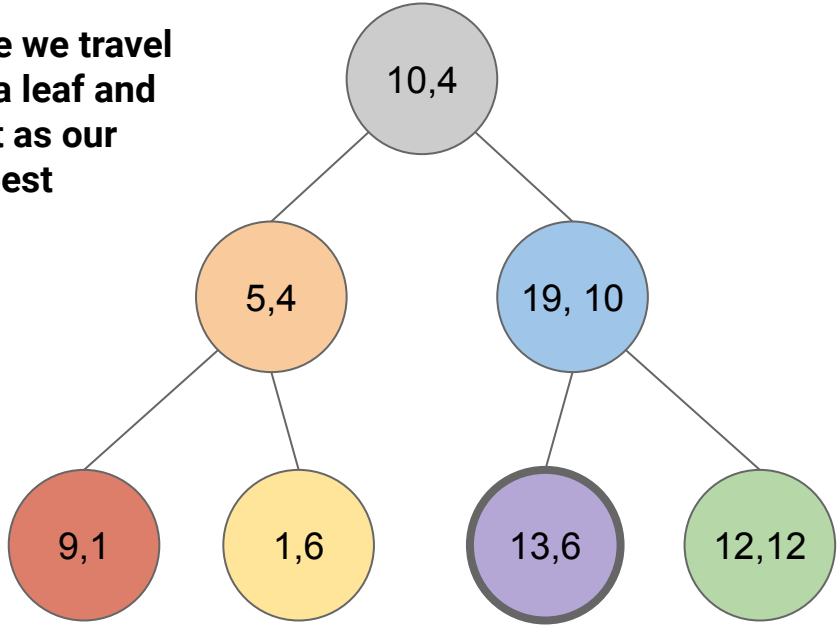




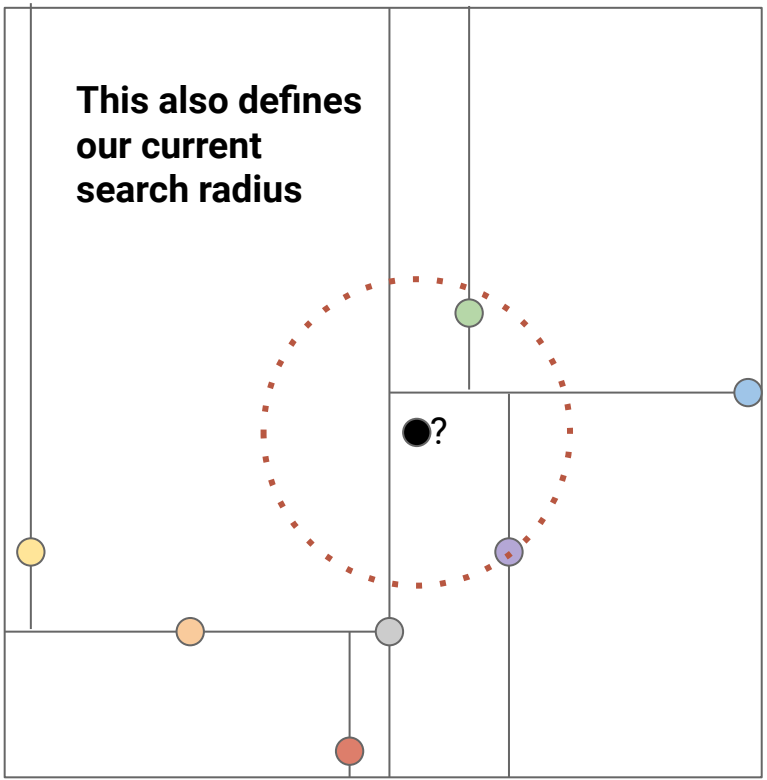
Nearest Neighbor - Example 2



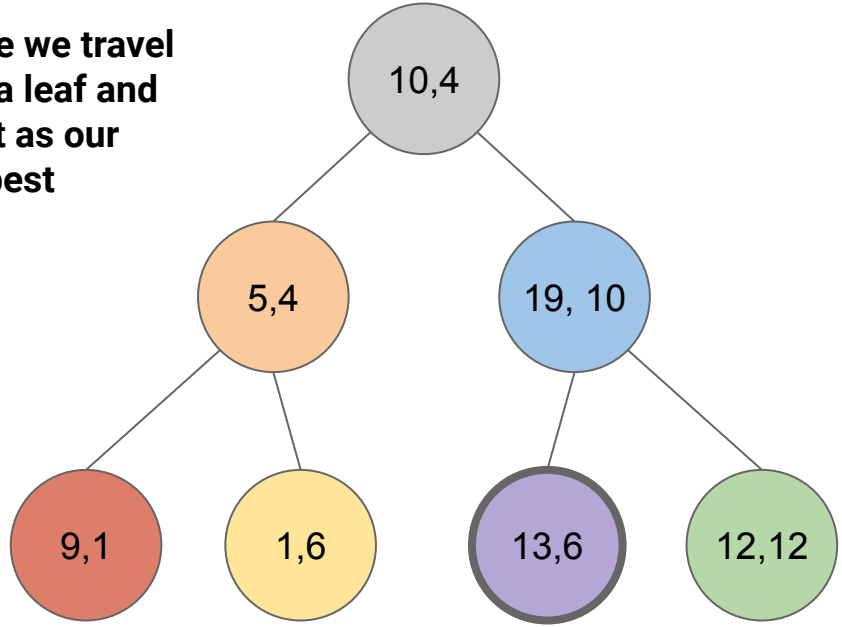
As before we travel down to a leaf and treat that as our current best



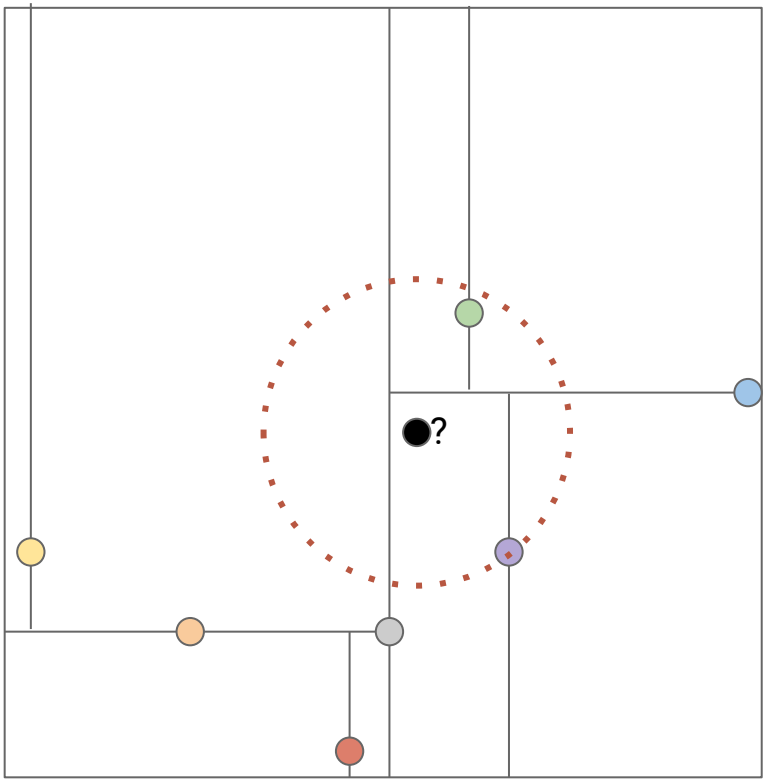
Nearest Neighbor - Example 2



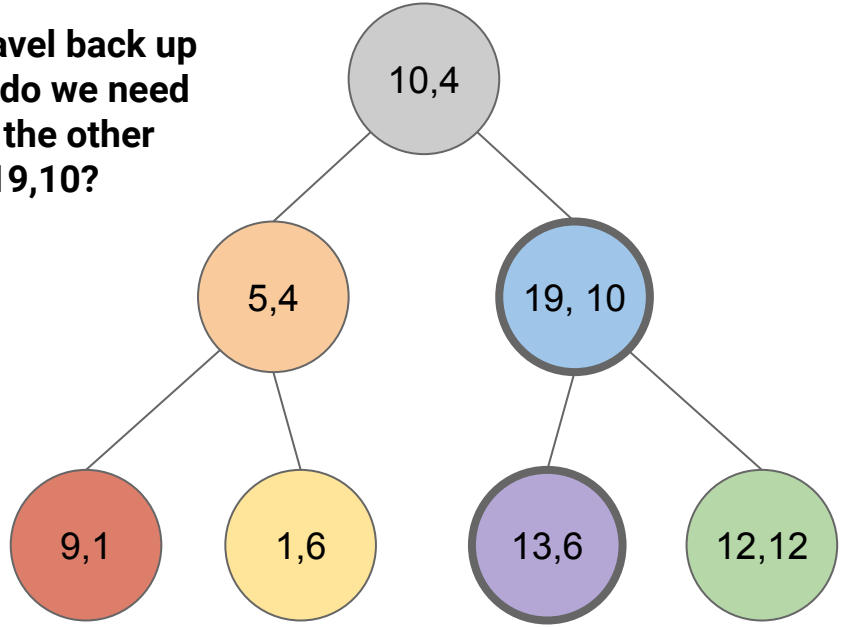
As before we travel down to a leaf and treat that as our current best



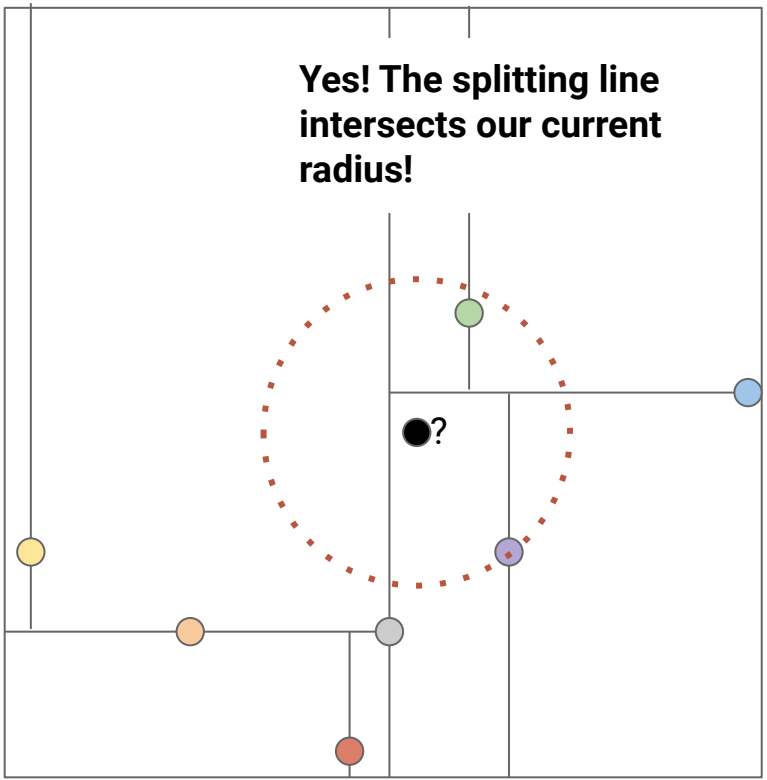
Nearest Neighbor - Example 2



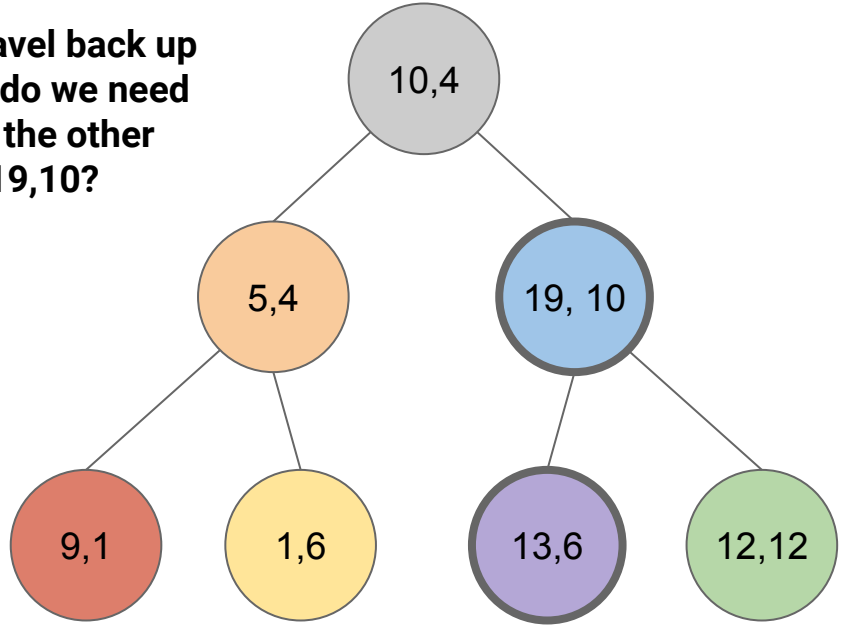
As we travel back up the tree, do we need to check the other child of 19,10?



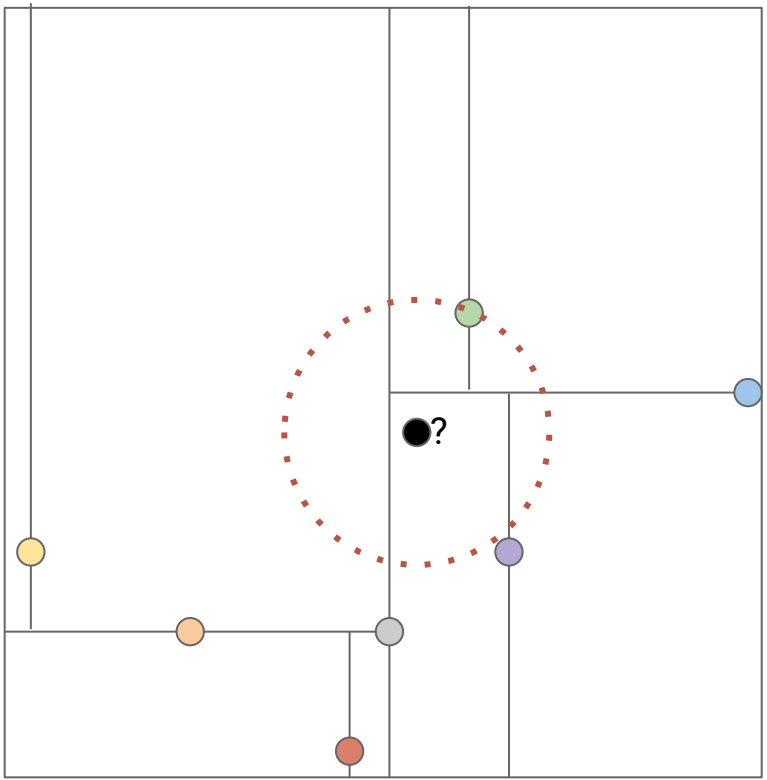
Nearest Neighbor - Example 2



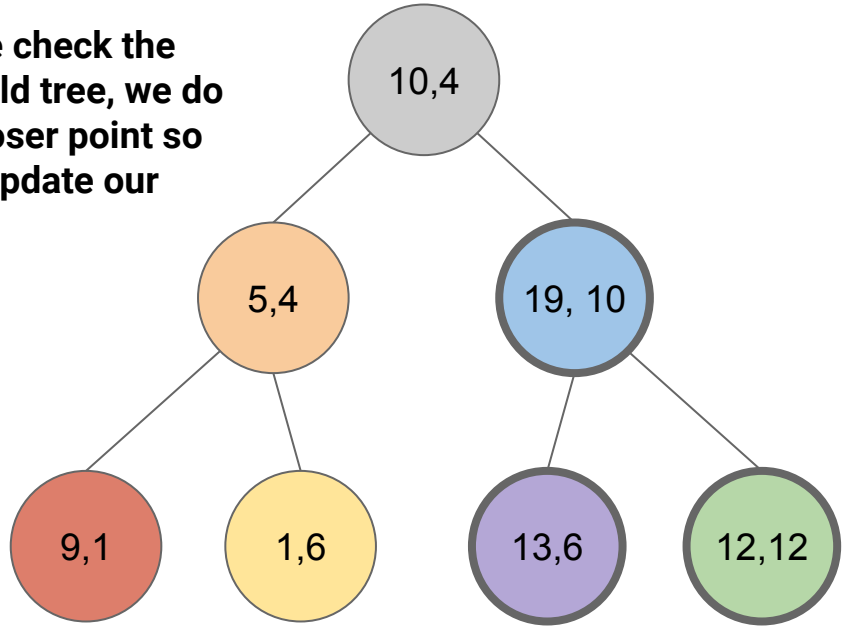
As we travel back up the tree, do we need to check the other child of 19,10?



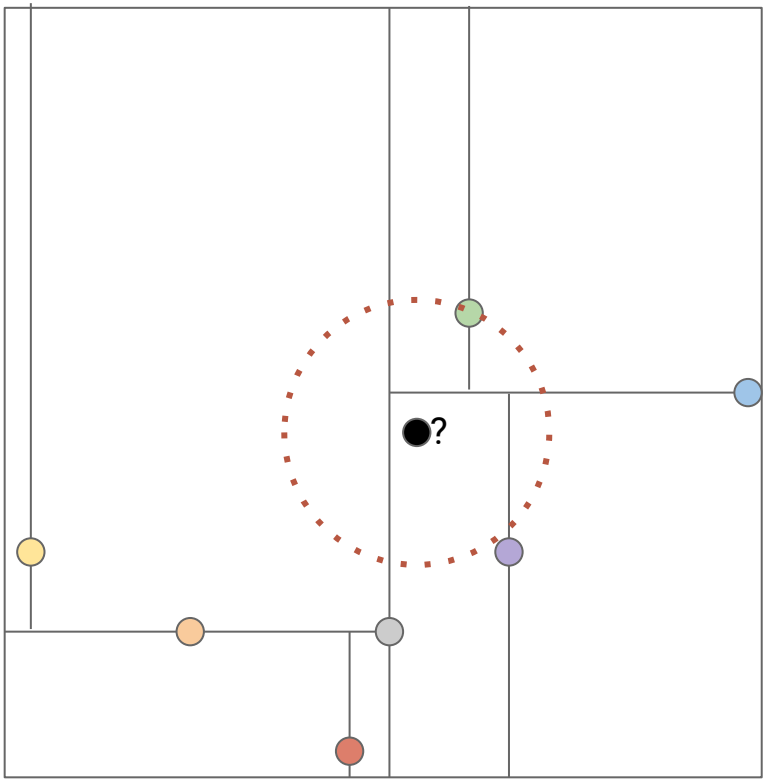
Nearest Neighbor - Example 2



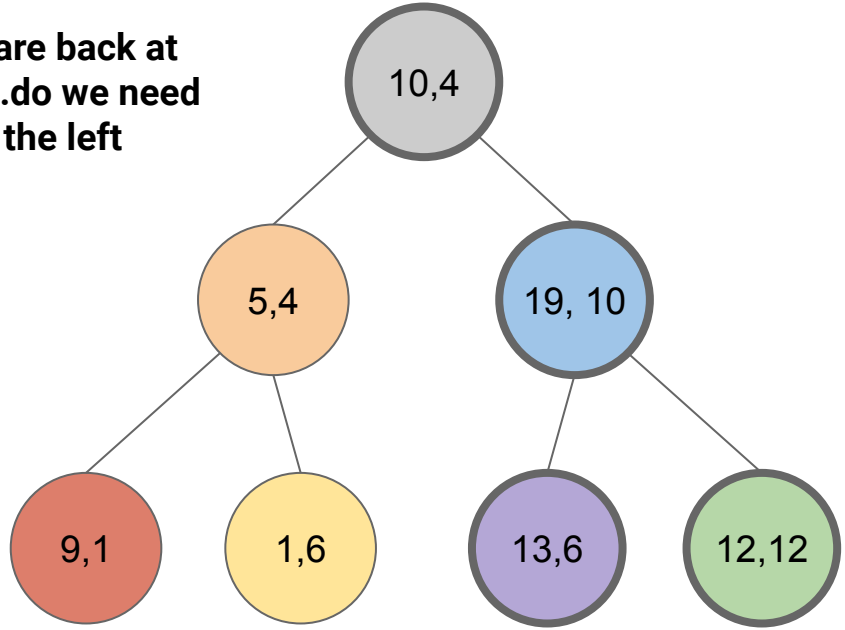
When we check the other child tree, we do find a closer point so we can update our radius.



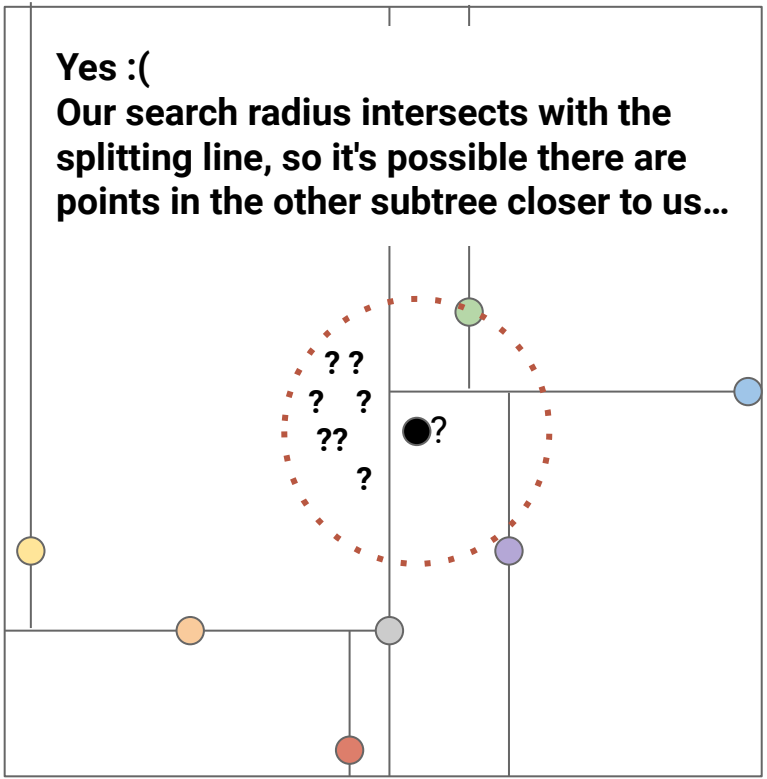
Nearest Neighbor - Example 2



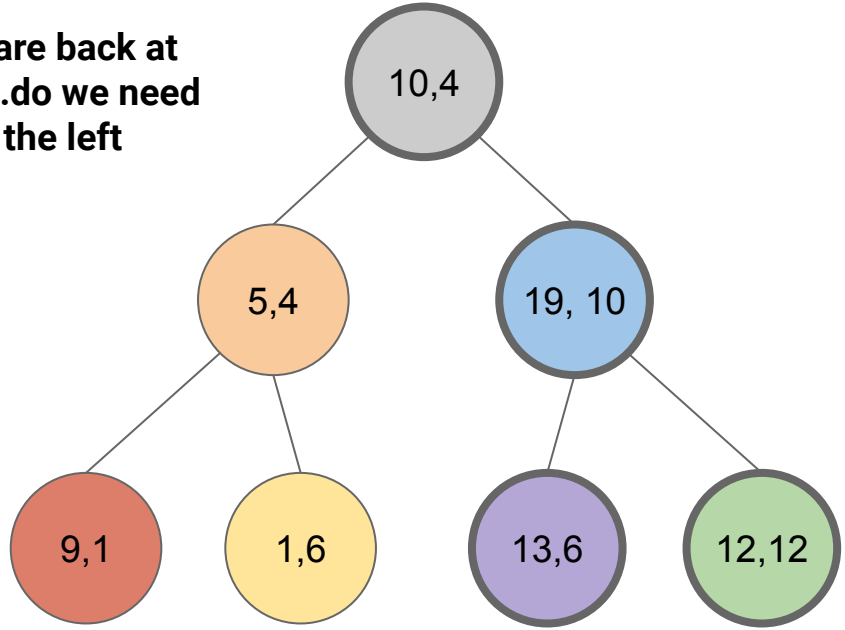
Now we are back at the root...do we need to check the left subtree?



Nearest Neighbor - Example 2



Now we are back at the root…do we need to check the left subtree?



Generalization: k-Nearest Neighbors

Finding one point can be as fast as $O(\log(n))$, but as slow as $O(n)$...

What if we want to find the k-Nearest Neighbors instead?

Idea: Keep a list of the k-nearest points, and the furthest point defines our "search radius"

k-D Trees

Can generalize to k>2 dimensions

- Depth 0: Partition on Dimension 0
- Depth 1: Partition on Dimension 1
- …
- Depth k-1: Partition on Dimension k-1
- Depth k: Partition on Dimension 0
- Depth k+1: Partition on Dimension 1
- Depth i: Partition on Dimension (i mod k)

The name k-D tree comes from this generalization (k-Dimensional Tree)

In practice, `range()` and `knn()` become ~ $O(n)$ for k > 3

- If a subtree's range overlaps with the target in even one dimension, we need to search it. (Curse of Dimensionality)