

A short

b't of BST + KD Tree

Data Structures

CS 225

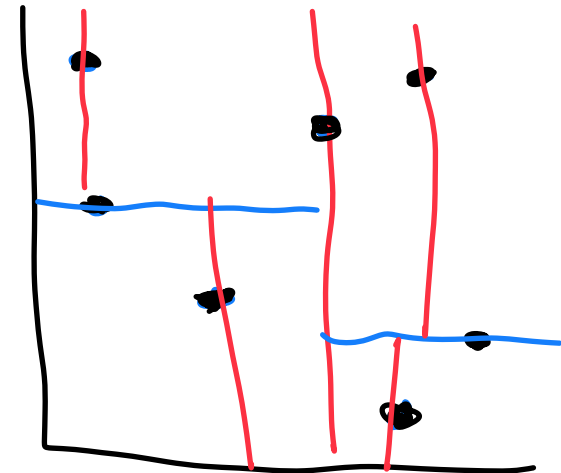
Brad Solomon

September 23, 2026



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



Informal Early Feedback Released!

A larger anonymous survey designed to give feedback to staff

Collective extra credit opportunity!

70% or more complete survey

Studying what aspects of class are most / least helpful



f2

FC

Exam 2 (9/30 — 10/02)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Register now

<https://courses.engr.illinois.edu/cs225/exams/>

Exam 2 (9/30 — 10/02)

Lab for this week is **EXAM 2 REVIEW SESSION**

Strongly encouraged to attend!

Extra credit lab releases this week (lab_trees)!

Those are
hard
questions

Learning Objectives

Review BST Efficiency

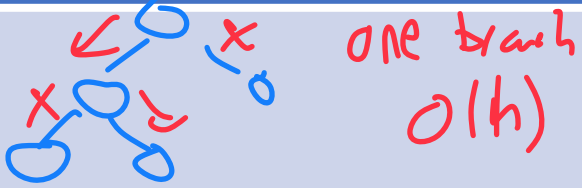
Explore the need and use of range search in 1D and 2D+

Introduce the KD Tree

Is on exam
(non coding)

Not
on
exam

BST Analysis – Running Time

Operation	BST Worst Case	
find	 one branch $O(h)$	$\frac{O(h)}{\sim}$
insert	Do find() Insert at ref pointer	$\frac{O(h)}{O(1)}$
remove		$\frac{O(h)}{\sim}$
traverse		

BST Analysis

Every operation on a BST depends on the **height** of the tree.

... how do we relate $O(h)$ to n , the size of our dataset?

BST Analysis

Join Code: 225



BST

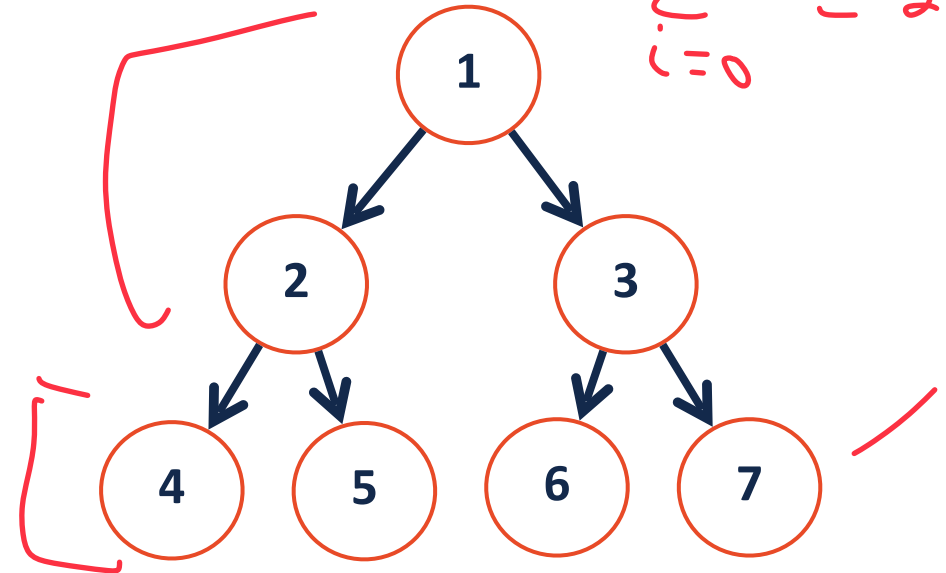
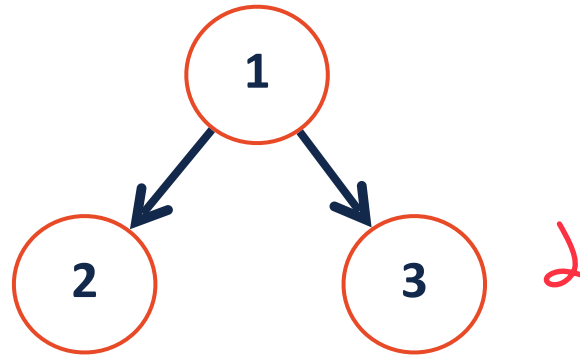
What is the **max** number of nodes in a tree of height h ?

BST Analysis

Join Code: 225



What is the **max** number of nodes in a tree of height h ?



$$\sum_{i=0}^h 2^i = 2^{h+1} - 1$$

of
nodes
in
perfect tree

2^h
100%

2^{h+1}
300%

$$2^{h+1} - 1$$

53%

of nodes in perfect tree

BST Analysis

Join Code: 225



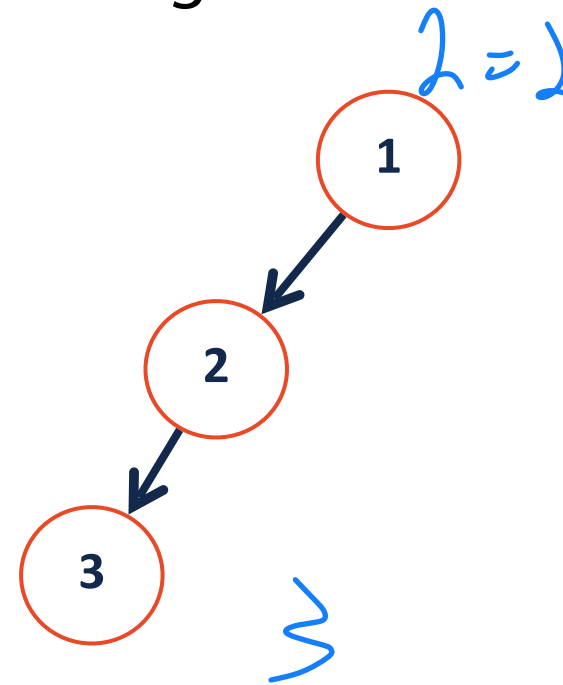
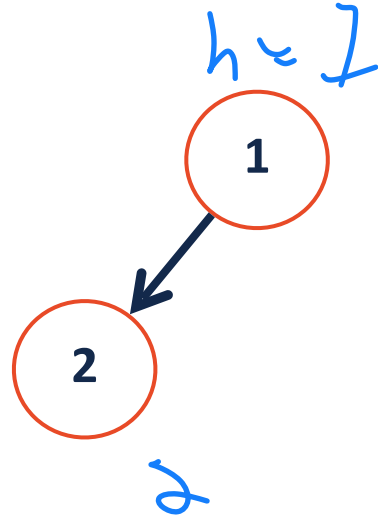
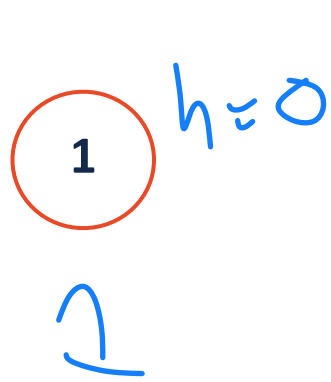
What is the **min** number of nodes in a tree of height h ?

BST Analysis

Join Code: 225



What is the **min** number of nodes in a tree of height h ?



h
15%

$h+1$
75%



BST Analysis

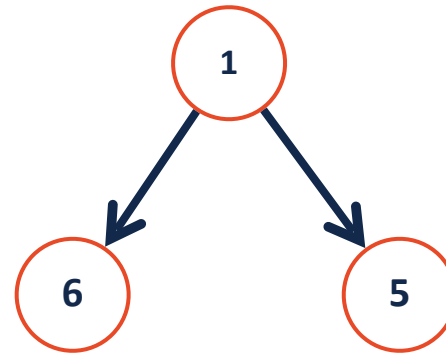
A BST of n nodes has a height between:

Lower-bound: $O(\log n)$

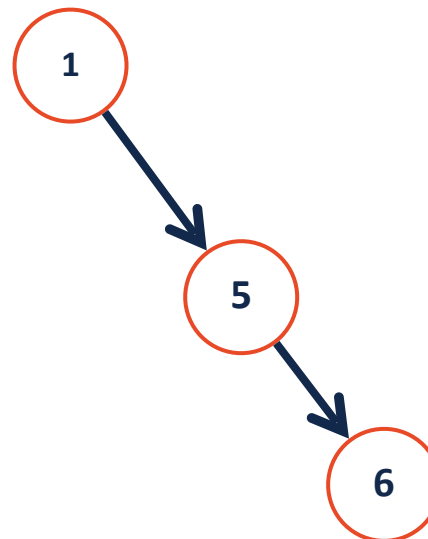
for a BST

Upper-bound: $O(n)$

$$O(h) = O(n)$$



perfect tree



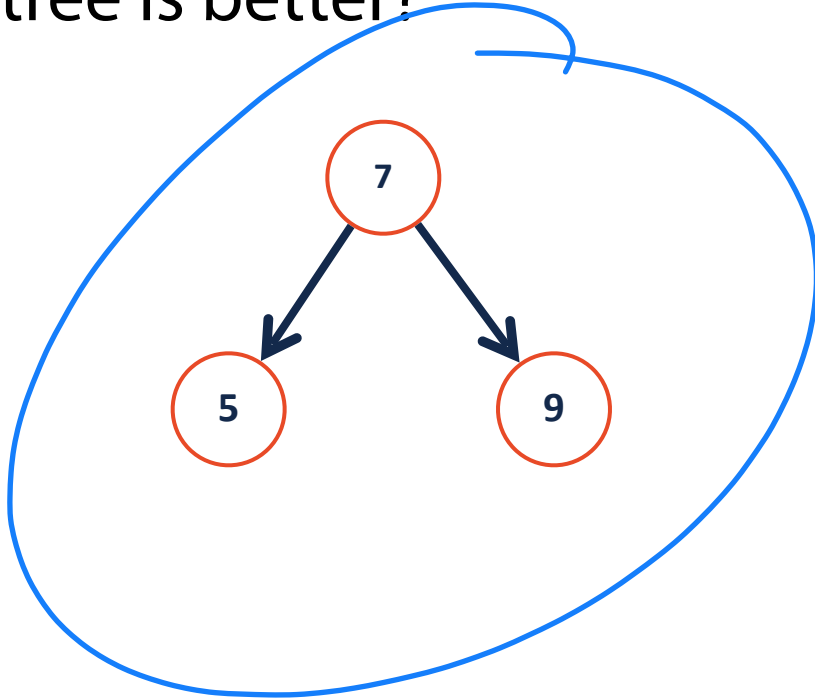
linked list

Height-Balanced Tree

Join Code: 225

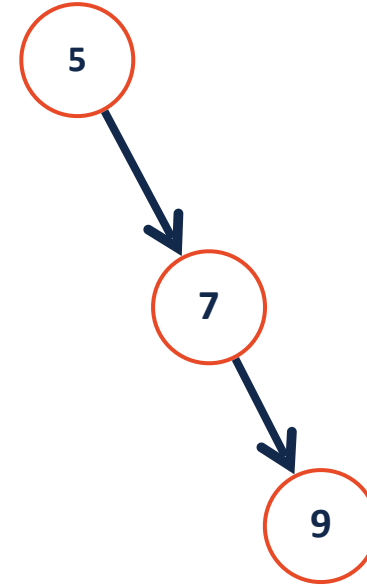


What tree is better?



A

Height bound $O(\log n)$



B

$O(n)$

Height-Balanced Tree

root $\rightarrow$ $\odot$
leaf $\rightarrow$ $\circ$

$\odot$

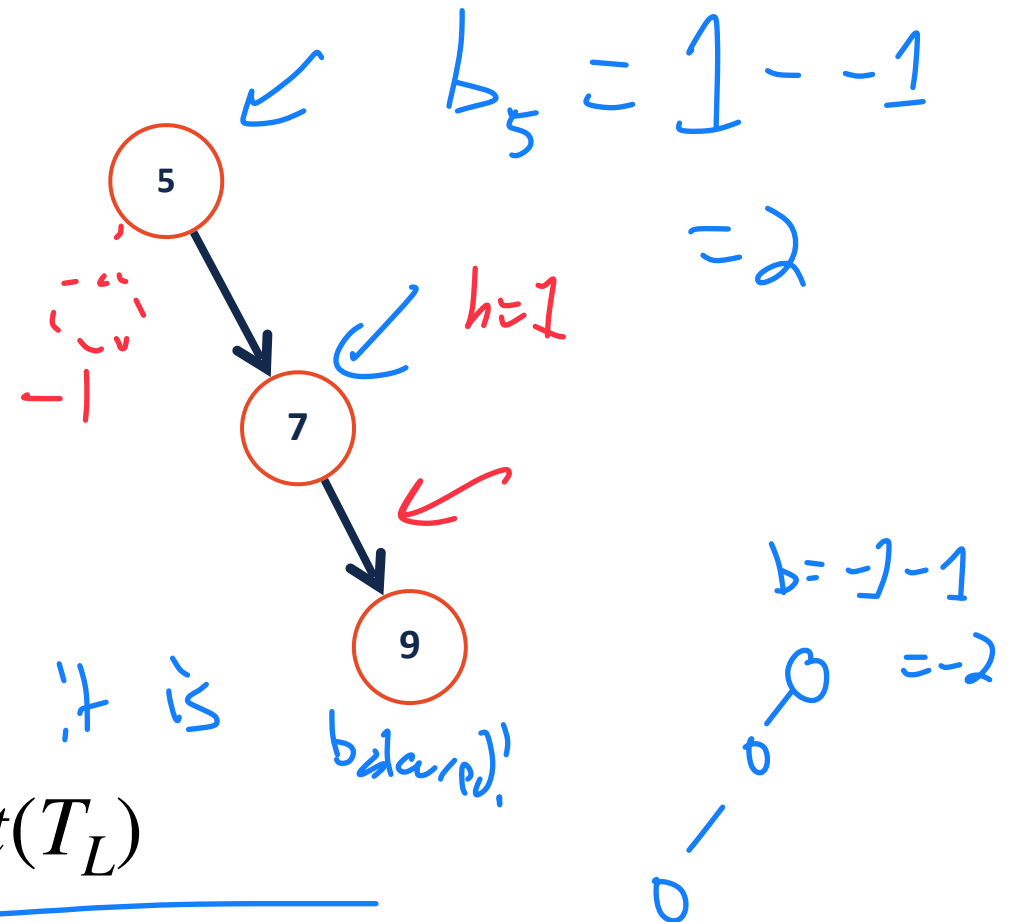
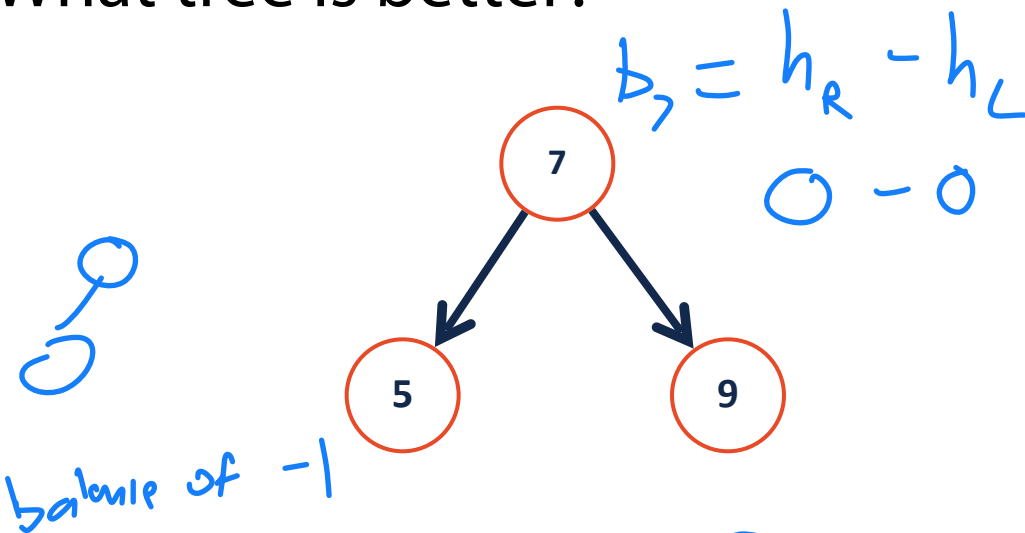
$h \leq 1$

$h = 0$

(0 edges)

$\odot$

What tree is better?



Is better b/c it is

Height balance: $b = \text{height}(T_R) - \text{height}(T_L)$

A tree is "balanced" if:

$$|b| \leq 1$$

Today: Creating a multidimensional 'balanced' tree

Next Week: Enforcing a balanced BST

Extensions to Tree Search

What is a situation where you want to find an item...

...or its nearest neighbor?

↳ Grocery Store Search

↳ MP_Mosaics → color matching

↳ Protein binding → Get closest known match

What is a situation where you want to find all items in a range?

↳ Any data filtering

Nearest Neighbor Tree Search

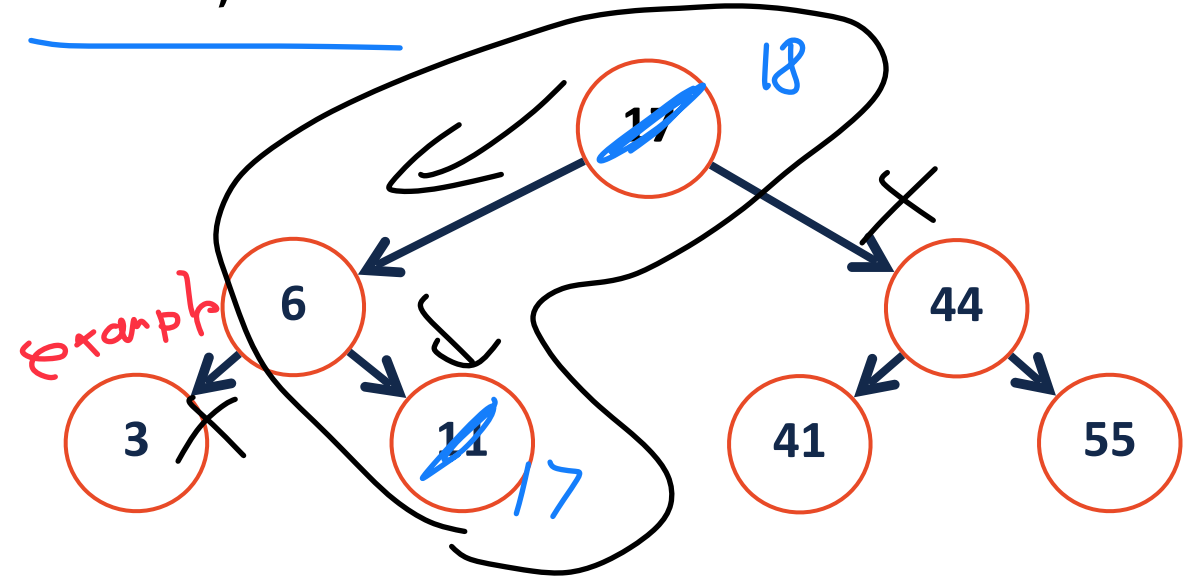
Consider a collection of points on a 1D line: $\mathbf{p} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$

If I wanted to find the closest item to 16, what would I do?

Start w/ root

↳ compare query to root
node ! if children are closer

X Doesn't write in blue



Simple: Do find!

Do find!
On recursion "backtrack" - check each value along path to see if it is < loser
Best = 17

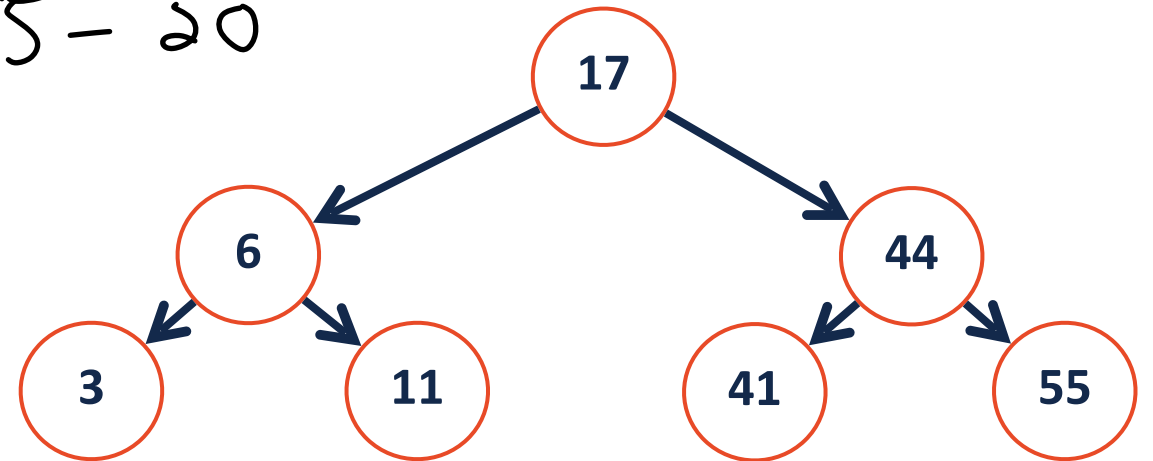
Range-based Tree Search

Consider a collection of points on a 1D line: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

If I want to find all values between $[A, B]$, how could I implement this?

Do find(5) * = begin
Do find(20) * nearest neighbor
= erring on smaller
end

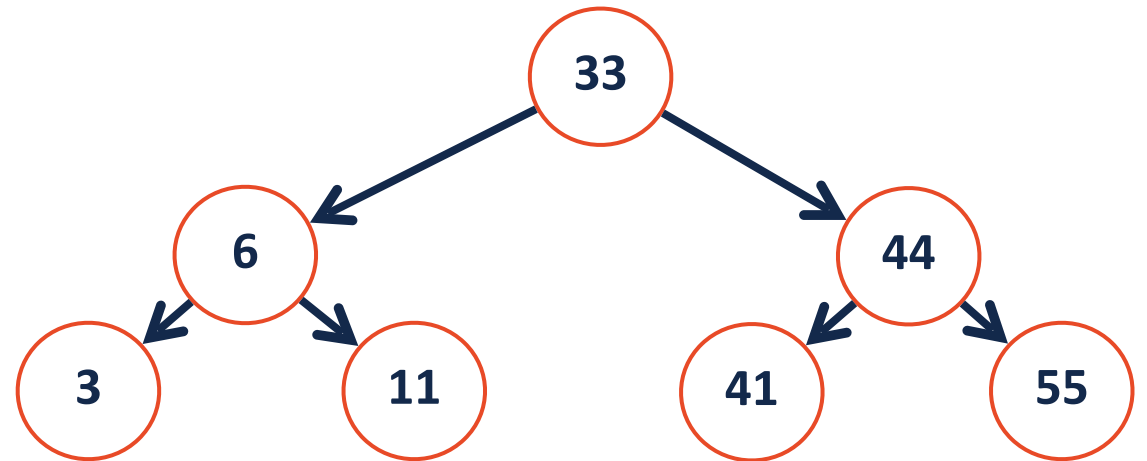
5 - 20



Range-based Tree Search

Consider a collection of points on a 1D line: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

If I want to find all values between $[A, B]$, how could I implement this?



Iterators!

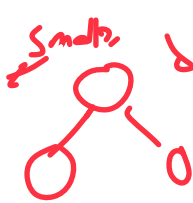
```
1  
2 for(auto it = myMap.lower_bound(A); it != myMap.upper_bound(B); ++it) {  
3  
4 // Do Stuff  
5 }
```

Nearest-Neighbor 2D Search

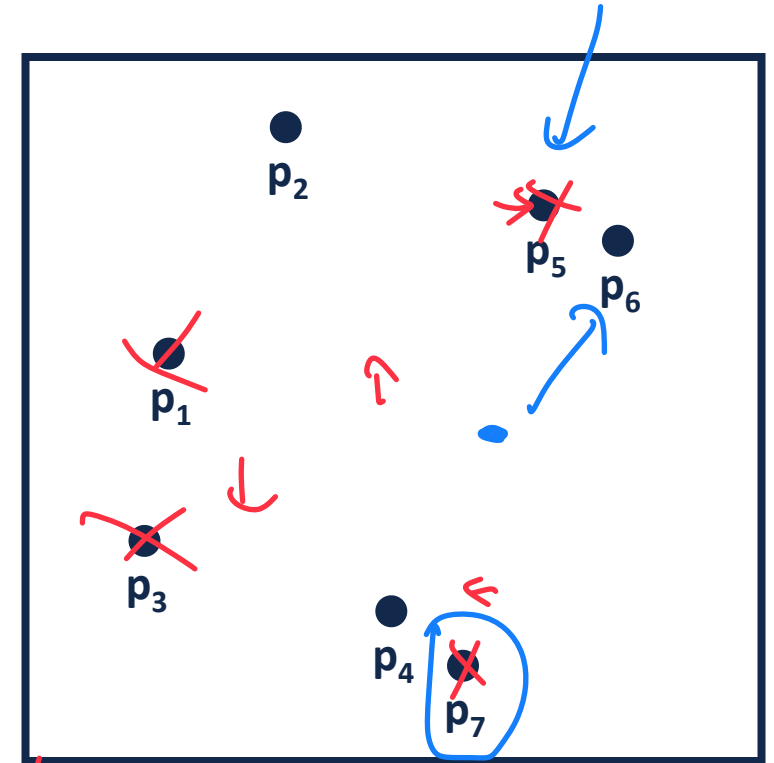
Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

What is nearest point to (x_1, y_1) ?

↓
Break into multiple 1D trees
smaller ↓ lower



↑ This won't guarantee correctness



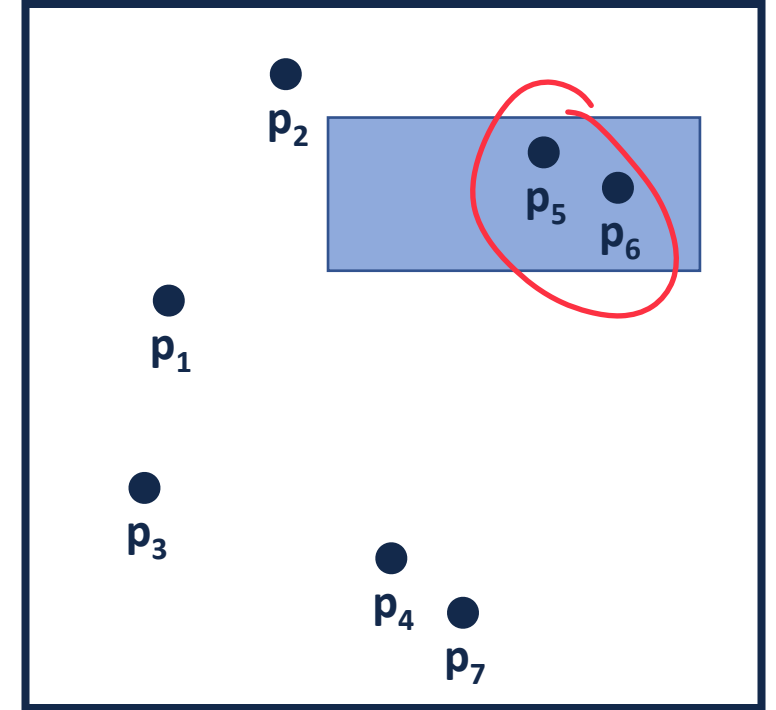
Maybe tree where value is distance
← per subset → X →

↳ would work but so would naive calc dist for every point

Range-based 2D Search

Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

What points in rectangle $[(x_1, y_1), (x_2, y_2)]$?



Constructing a multi-dimensional tree



Naive walks or 1D trees can't help us here! Need something that...

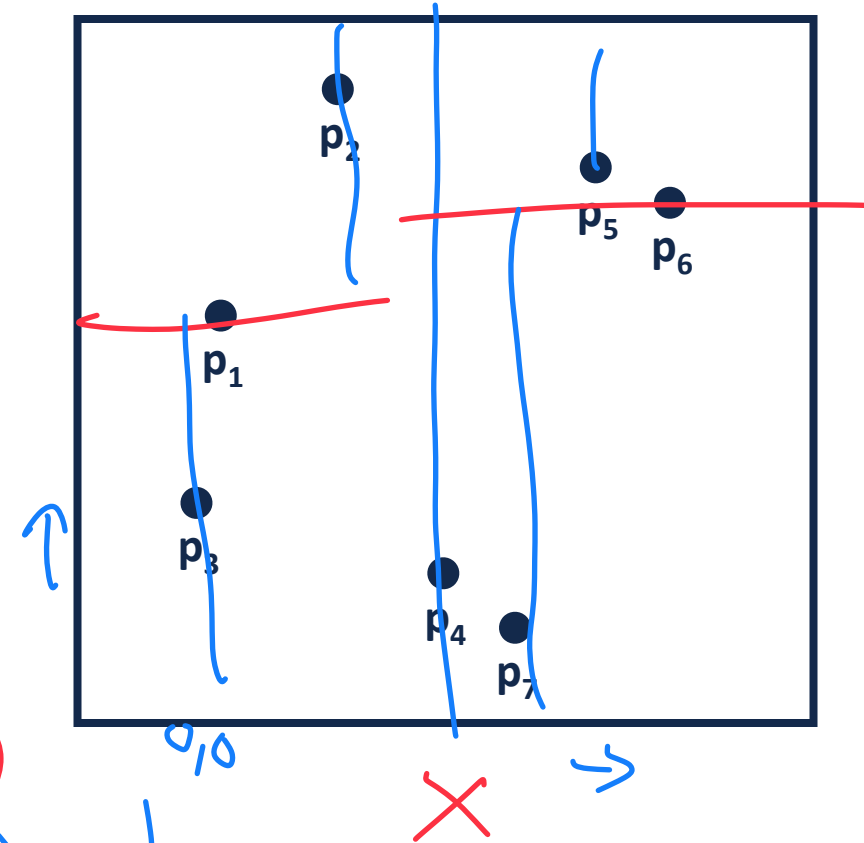
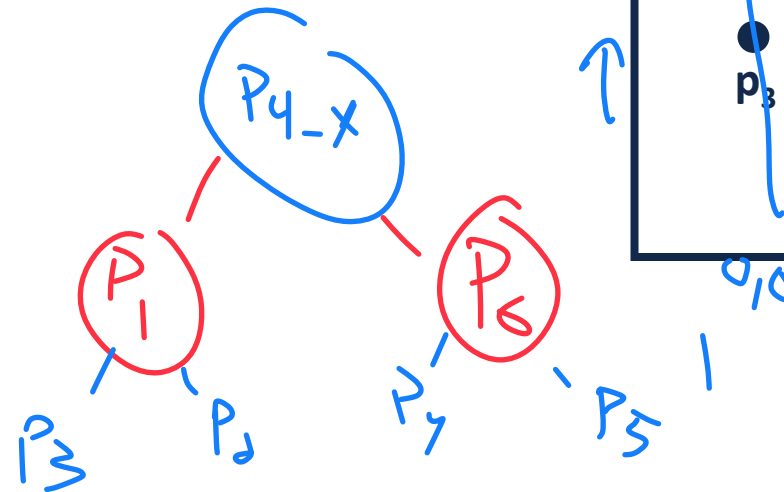
1. Starts with a binary search in k dimensions
2. Tries to enforce a balanced tree
3. At each node, can make a branch decision in $O(1)$ time

Constructing a multi-dimensional tree

Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

Tree Construction:

- 1) Start by picking one dimension
↳ pick median point as root (we want balance)
- 2) Repeat in different dim for children

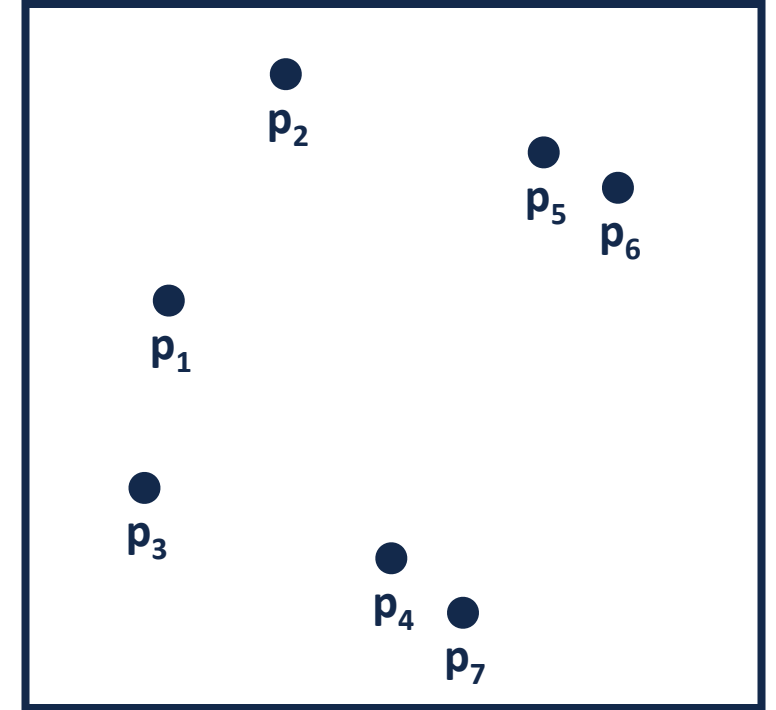


Constructing a multi-dimensional tree

Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

Tree Construction:

1. Find **median point** along a particular dim
2. Recurse left and right, **swapping** dimension

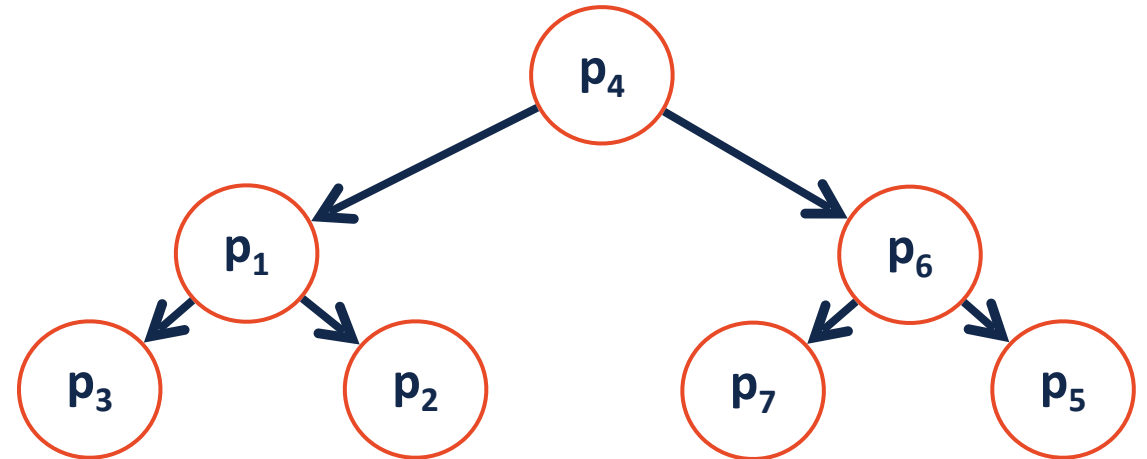
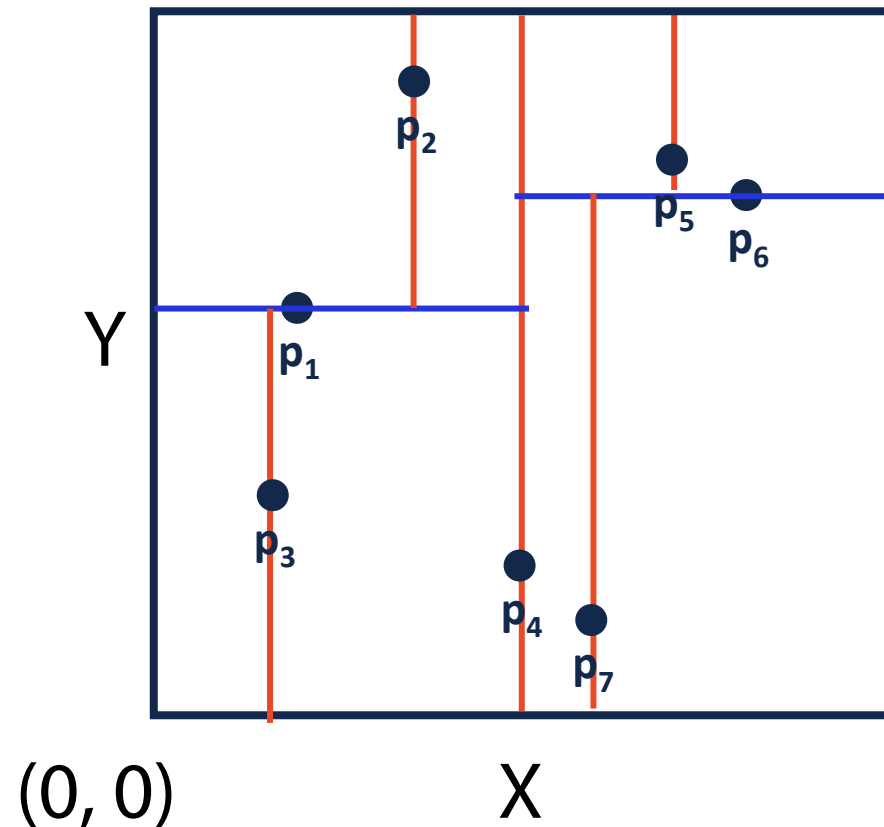


Constructing a multi-dimensional tree

We now have a tree which partitions a space into 'cells'

... Why did we choose to split on median point?

Build a balanced tree!

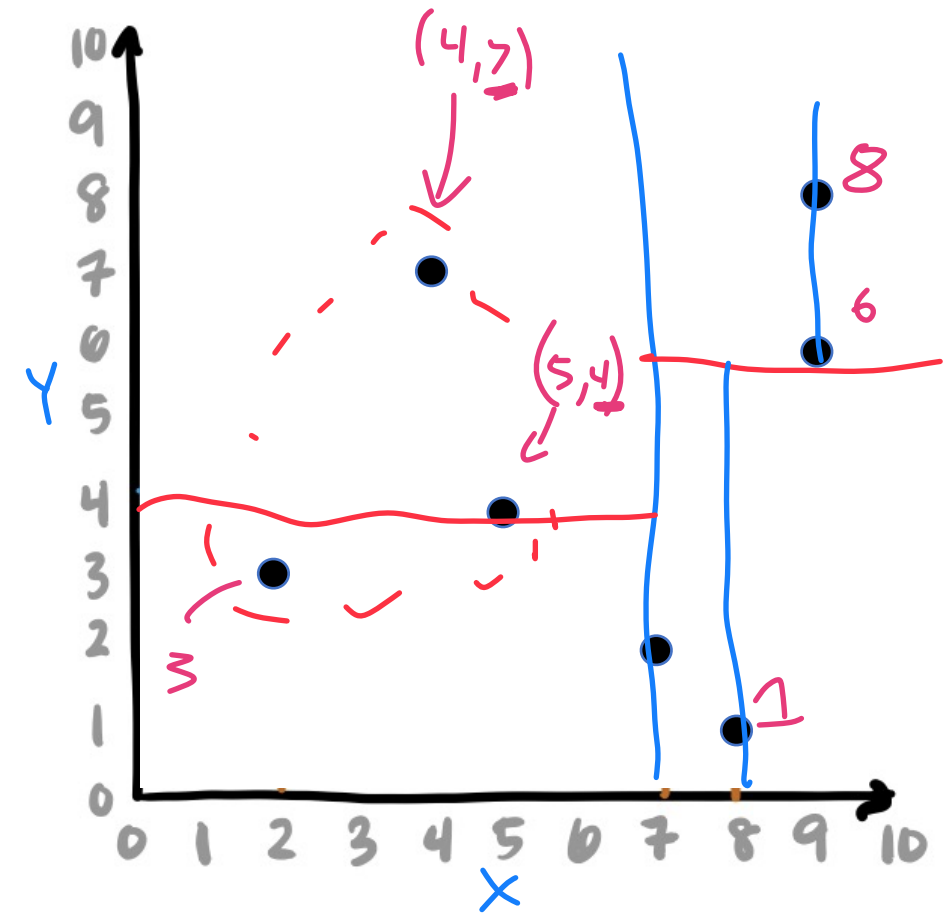
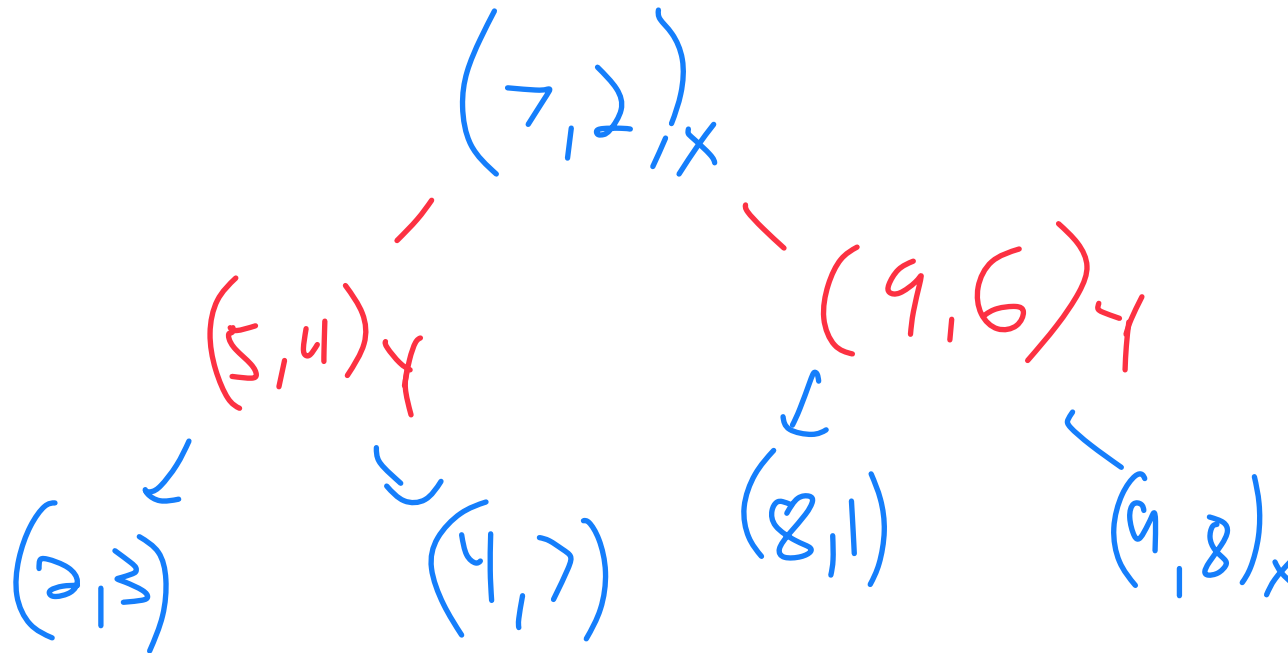




Nearest Neighbor: k-d tree

This construction strategy builds a **balanced** k-dimension (k-d tree):

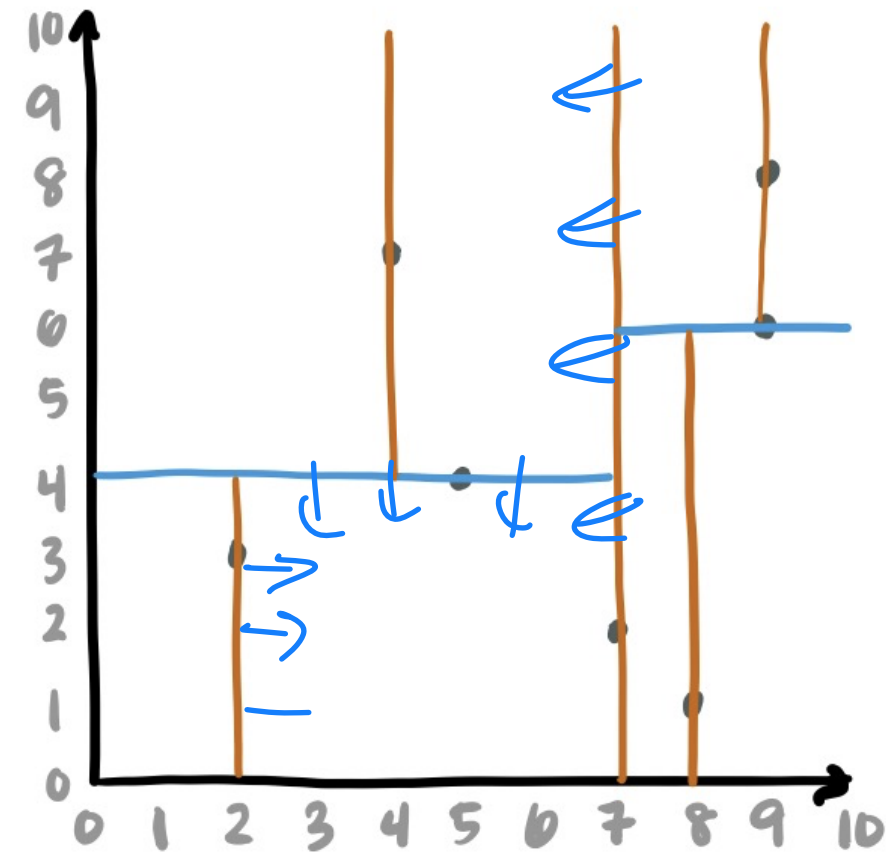
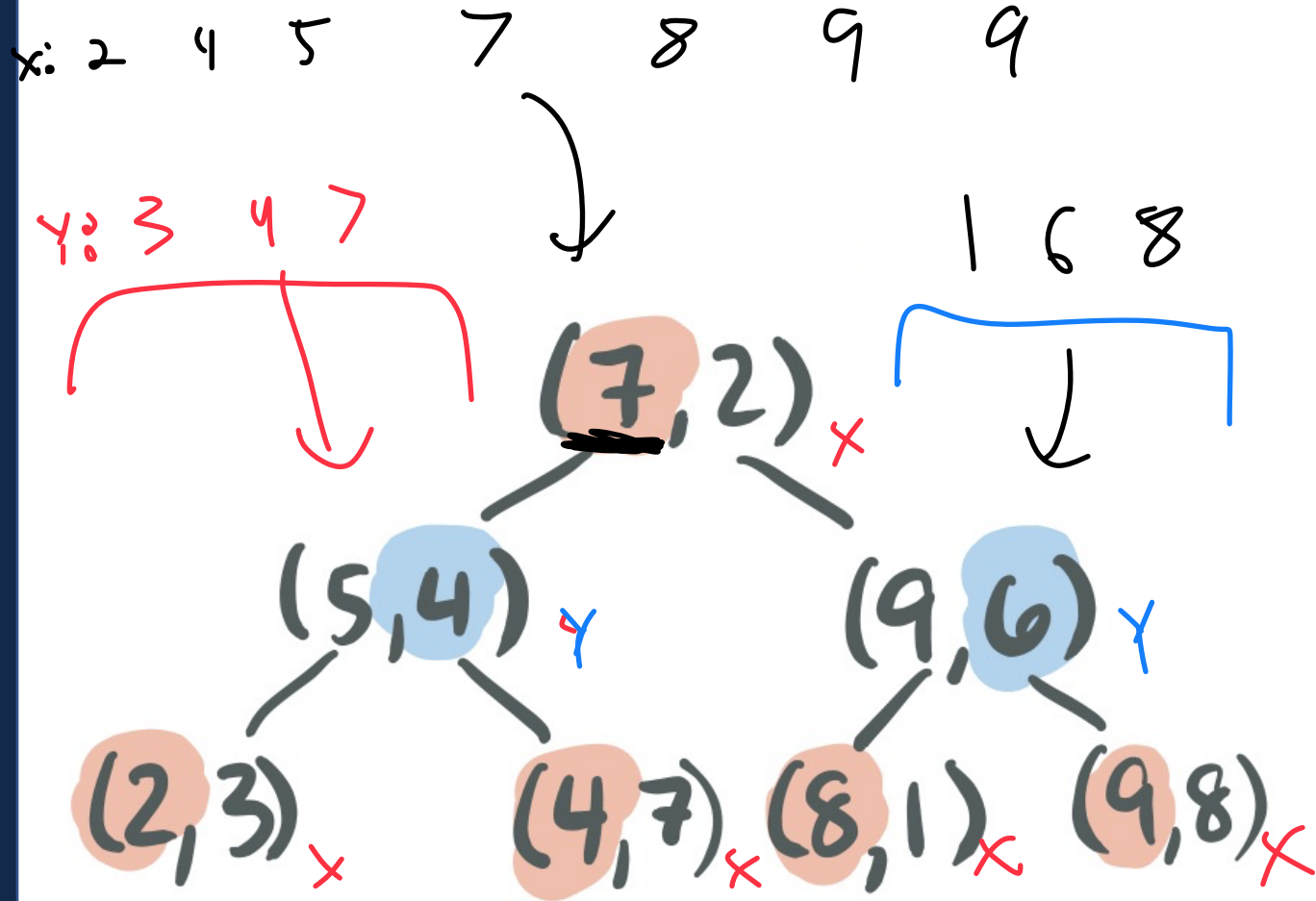
$(7,2)$, $(5,4)$, $(9,6)$, $(4,7)$, $(2,3)$, $(8,1)$, $(9,8)$



Nearest Neighbor: k-d tree



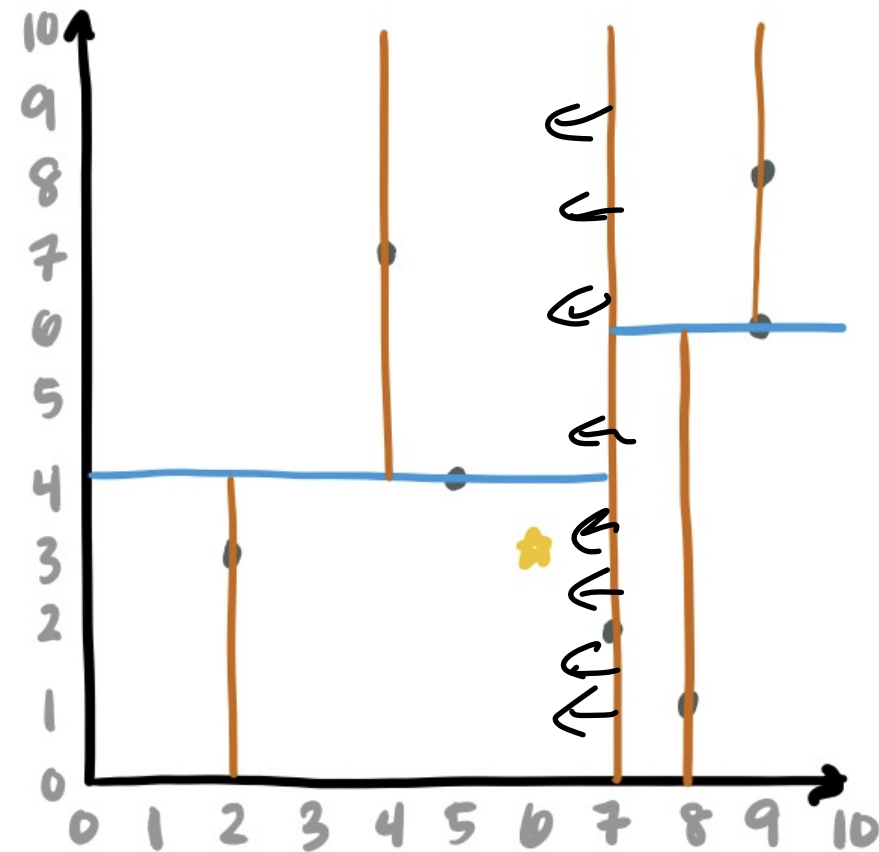
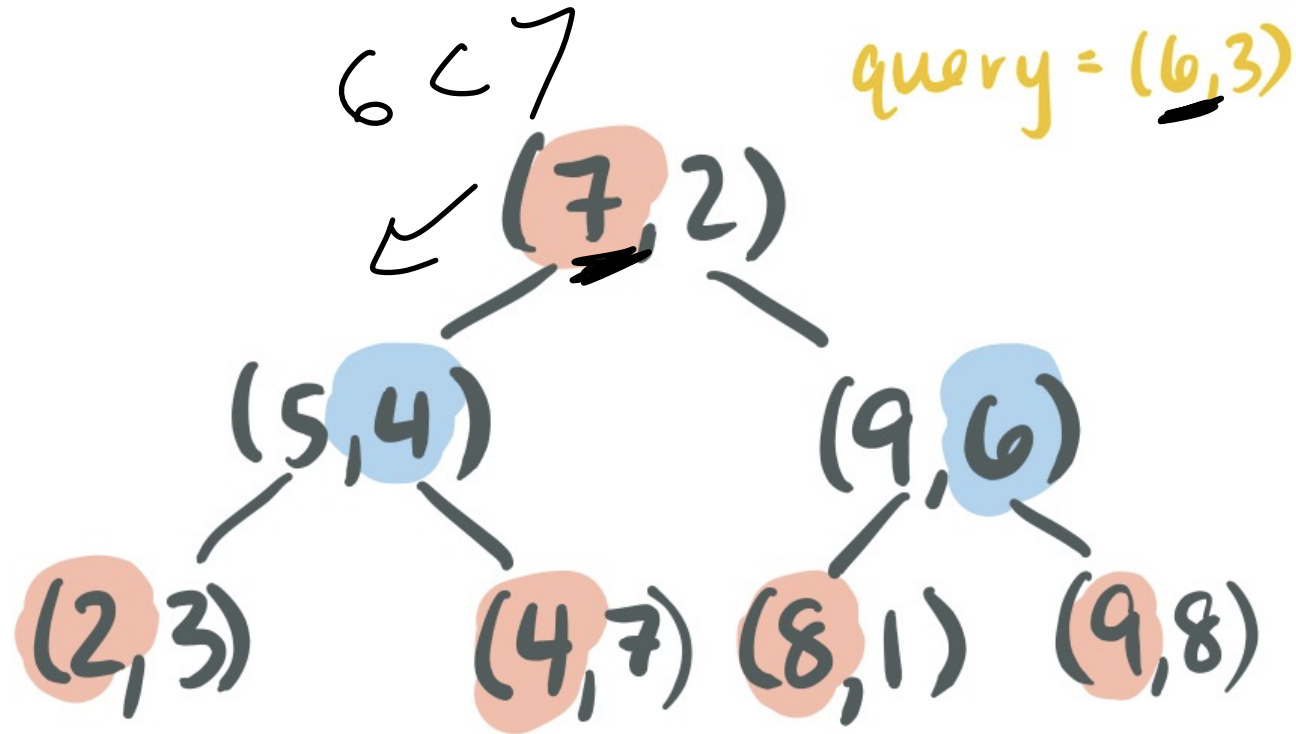
Claim: This tree allows sorted search in multiple dimensions!



Nearest Neighbor: k-d tree

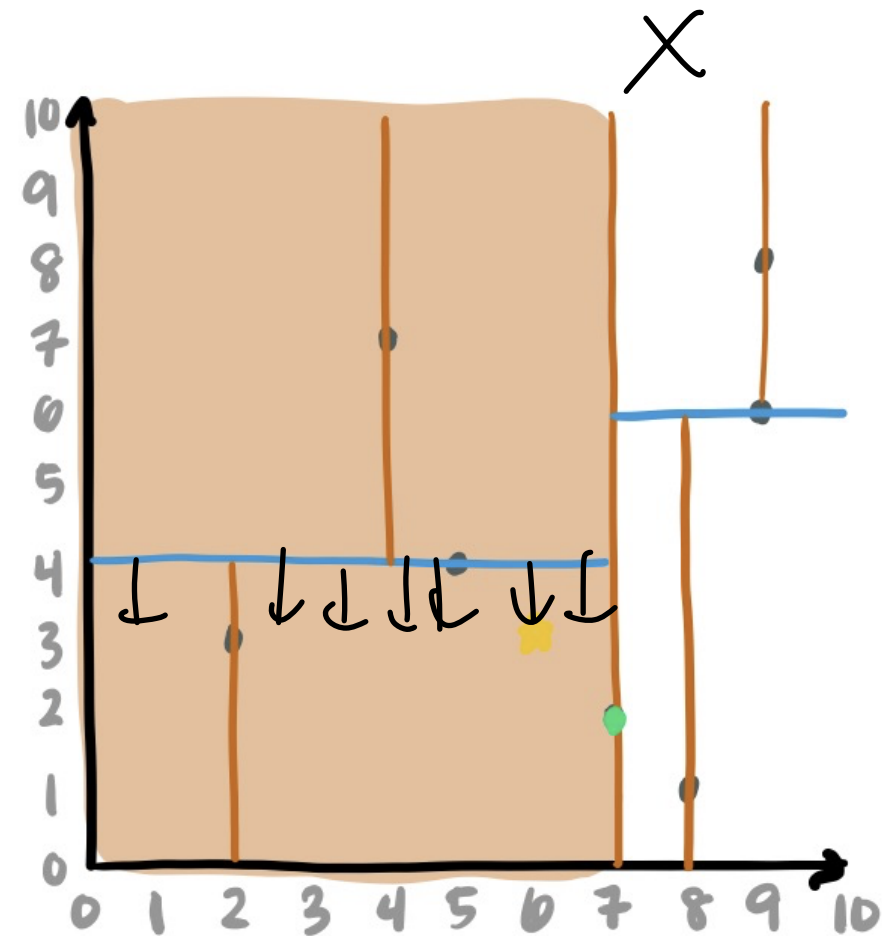
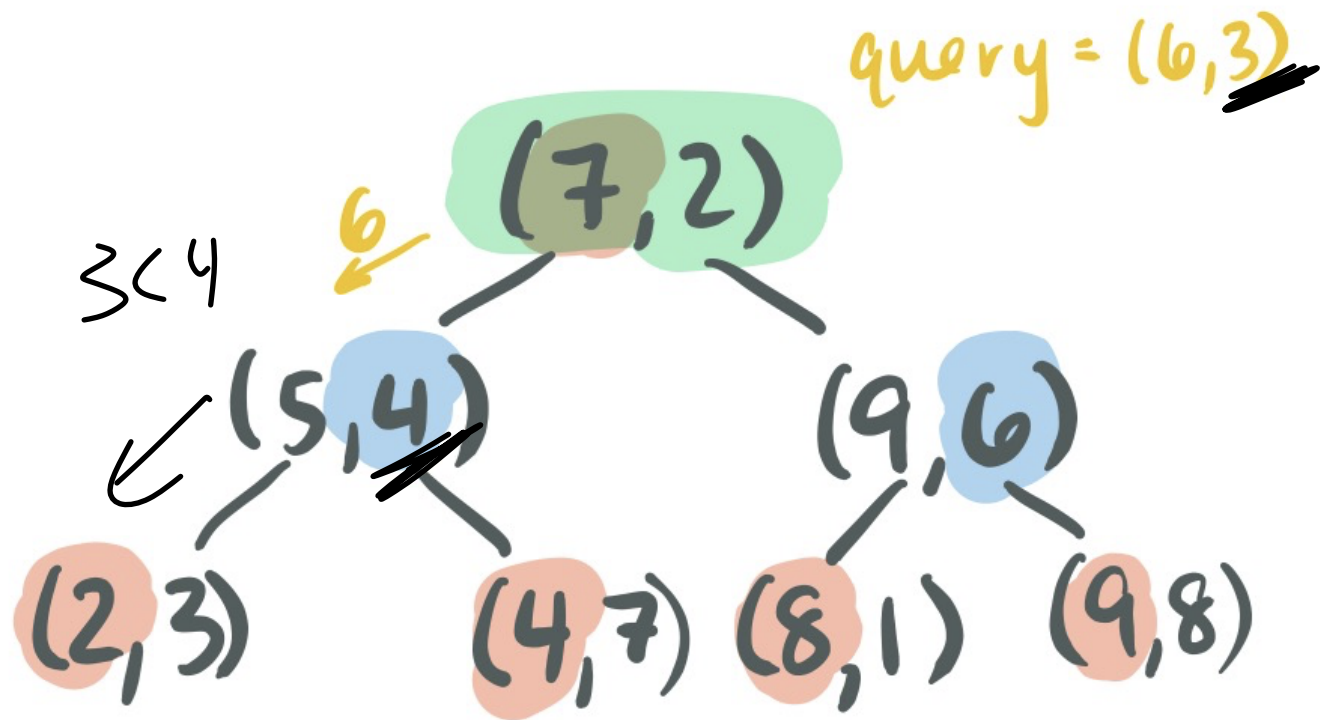
When querying a k-d tree, it acts like a BST* at first...

(compare only splitting dimension)



Nearest Neighbor: k-d tree

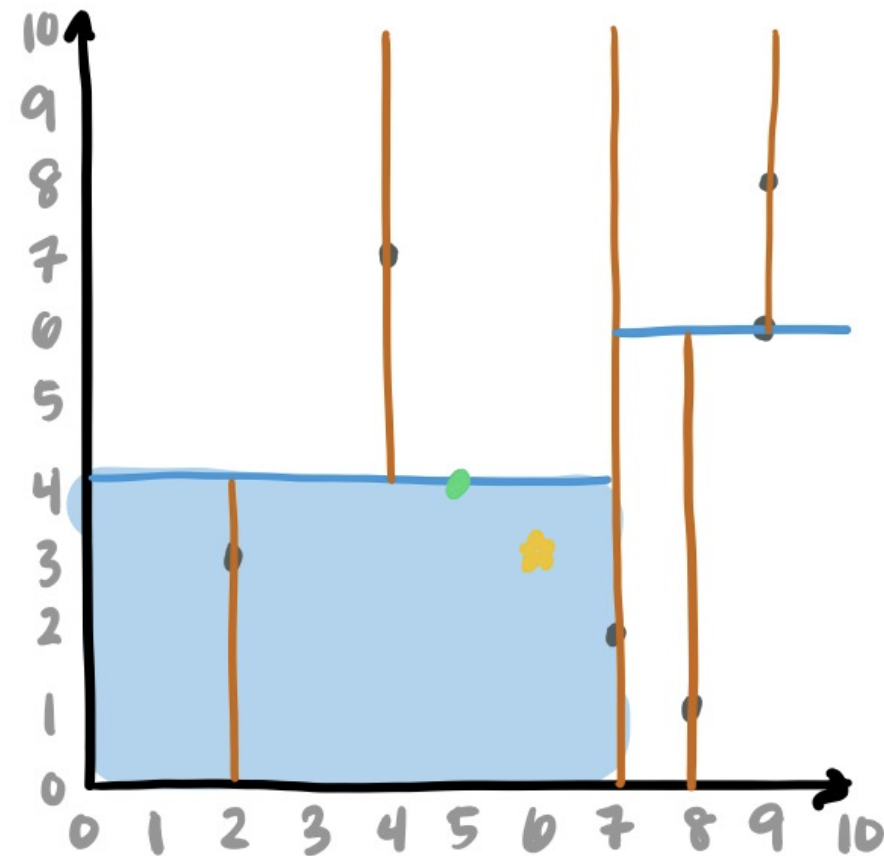
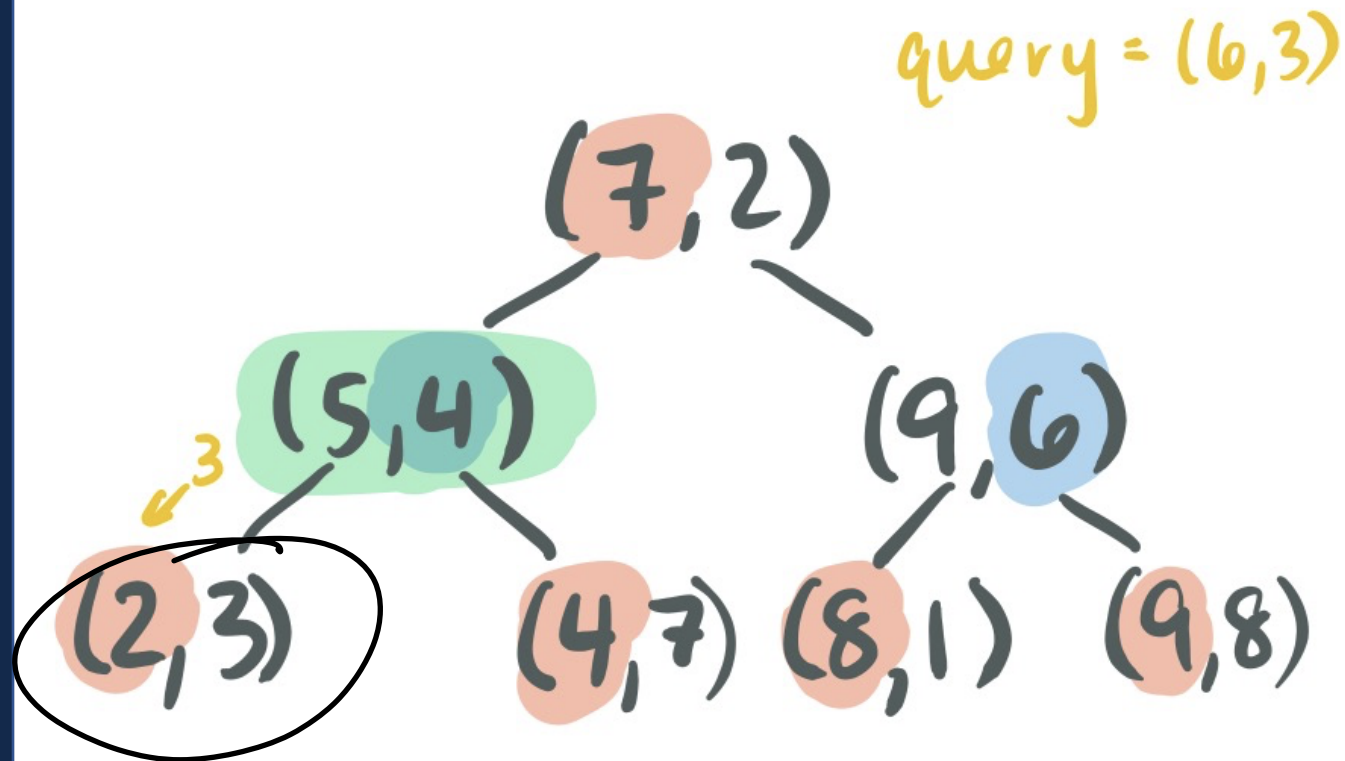
When querying a k-d tree, it acts like a BST* at first...



Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first...

Having reached leaf we have $(2,3)$ $\approx$ Not the nearest neighbor!



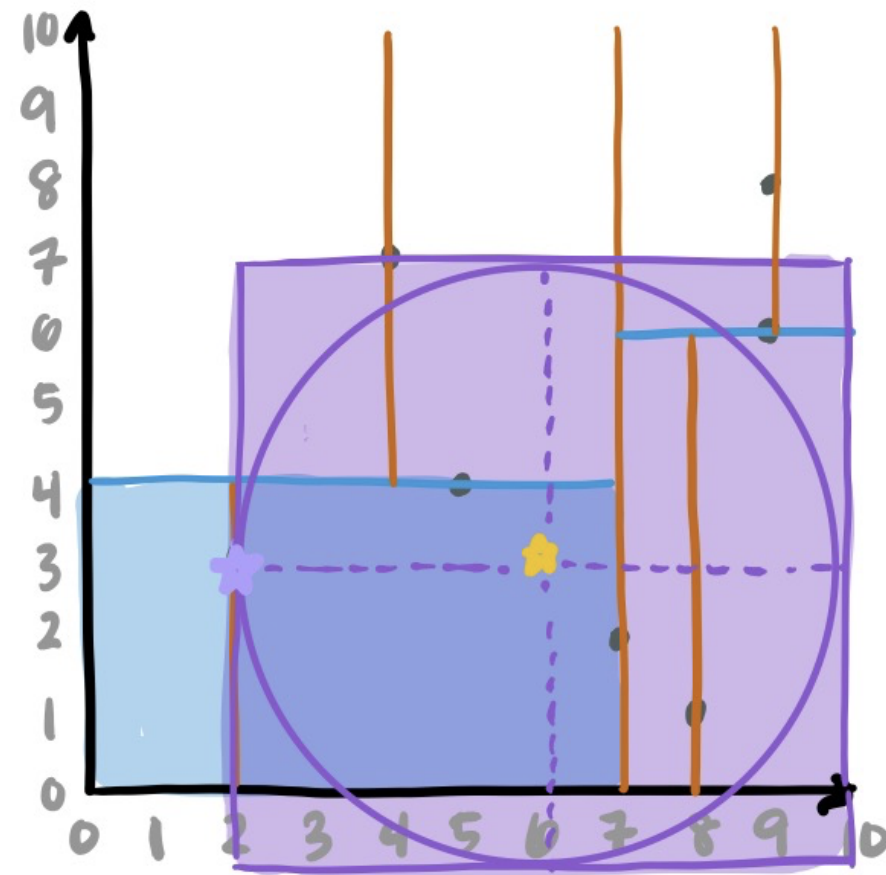
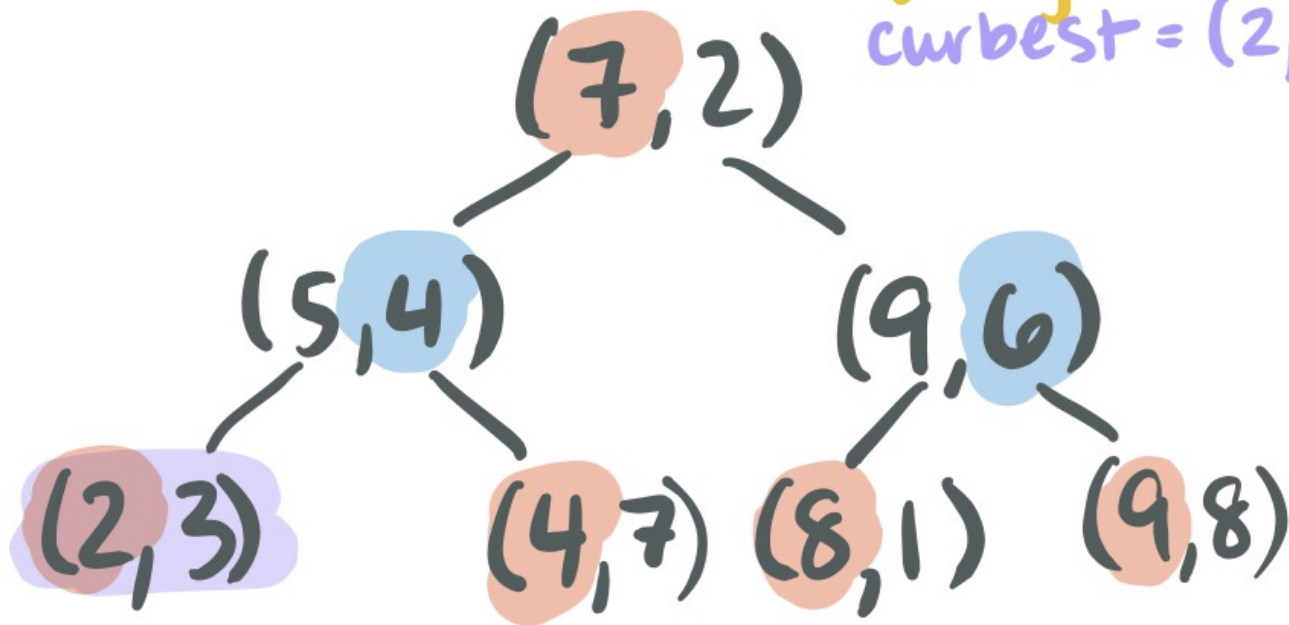
Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first...

... But if we don't find exact match, have to find nearest neighbor

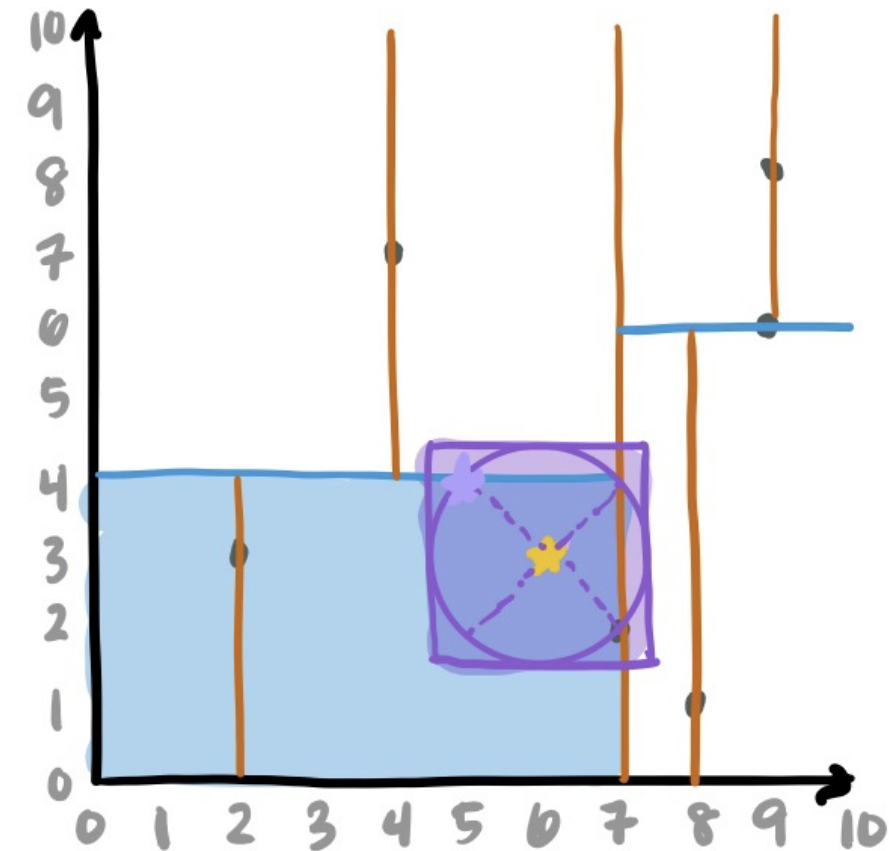
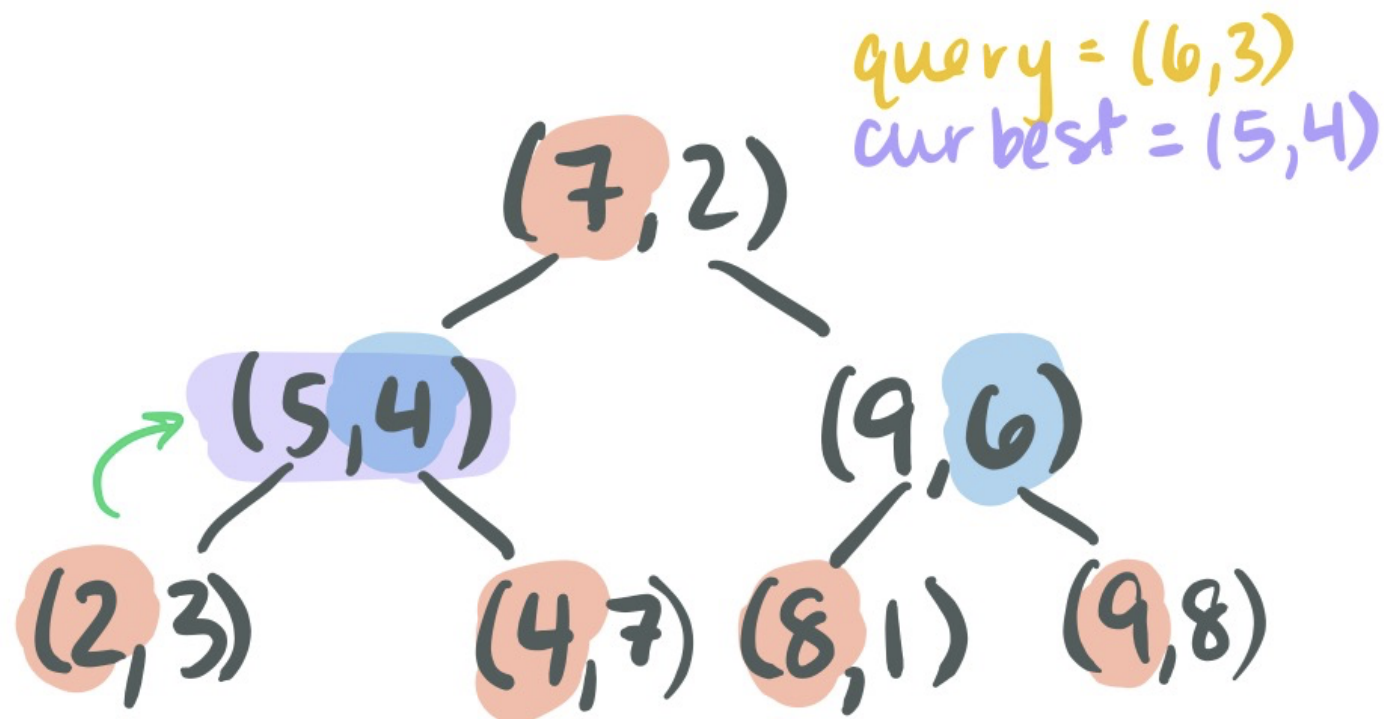
$(2,3)$ is my current best value

query = $(6,3)$
curbest = $(2,3)$

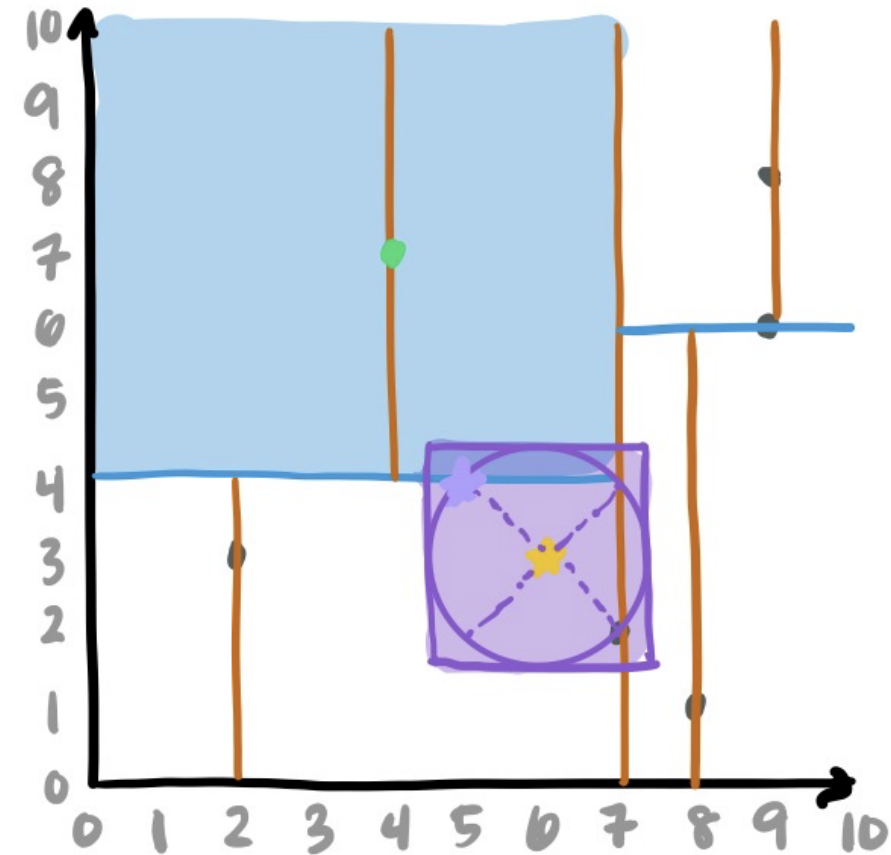
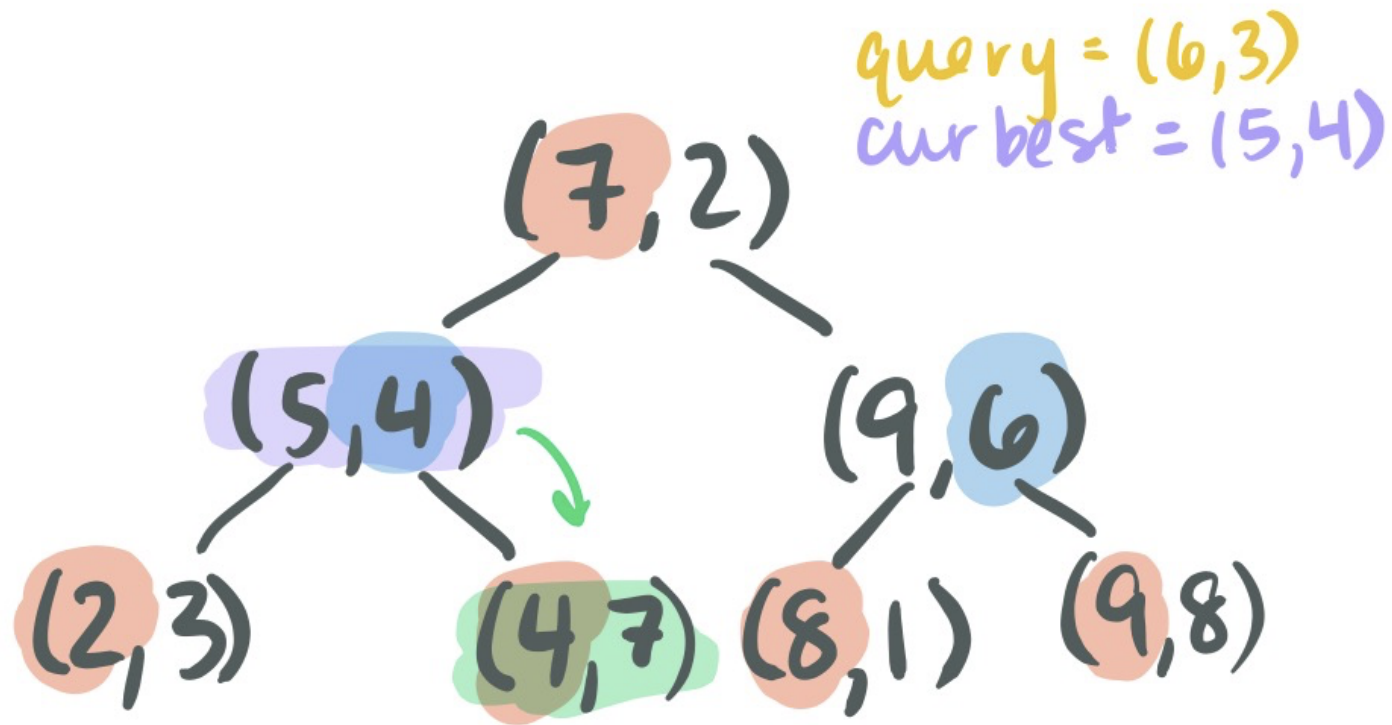


Nearest Neighbor: k-d tree

Backtracking: start recursing backwards -- store "best" possibility as you trace back

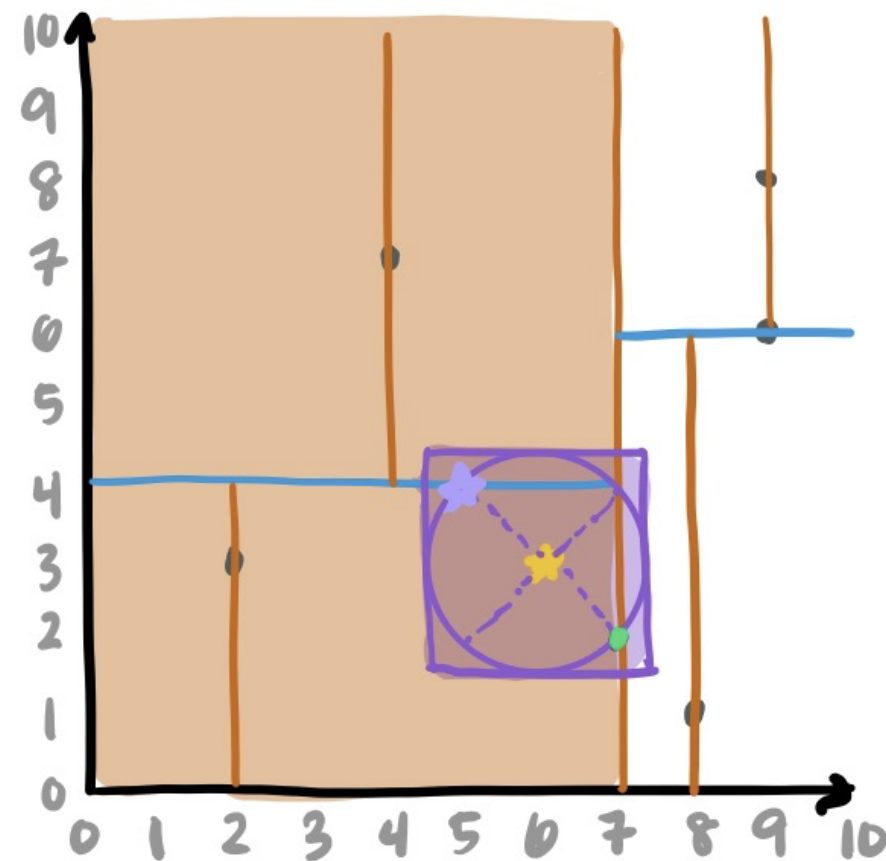
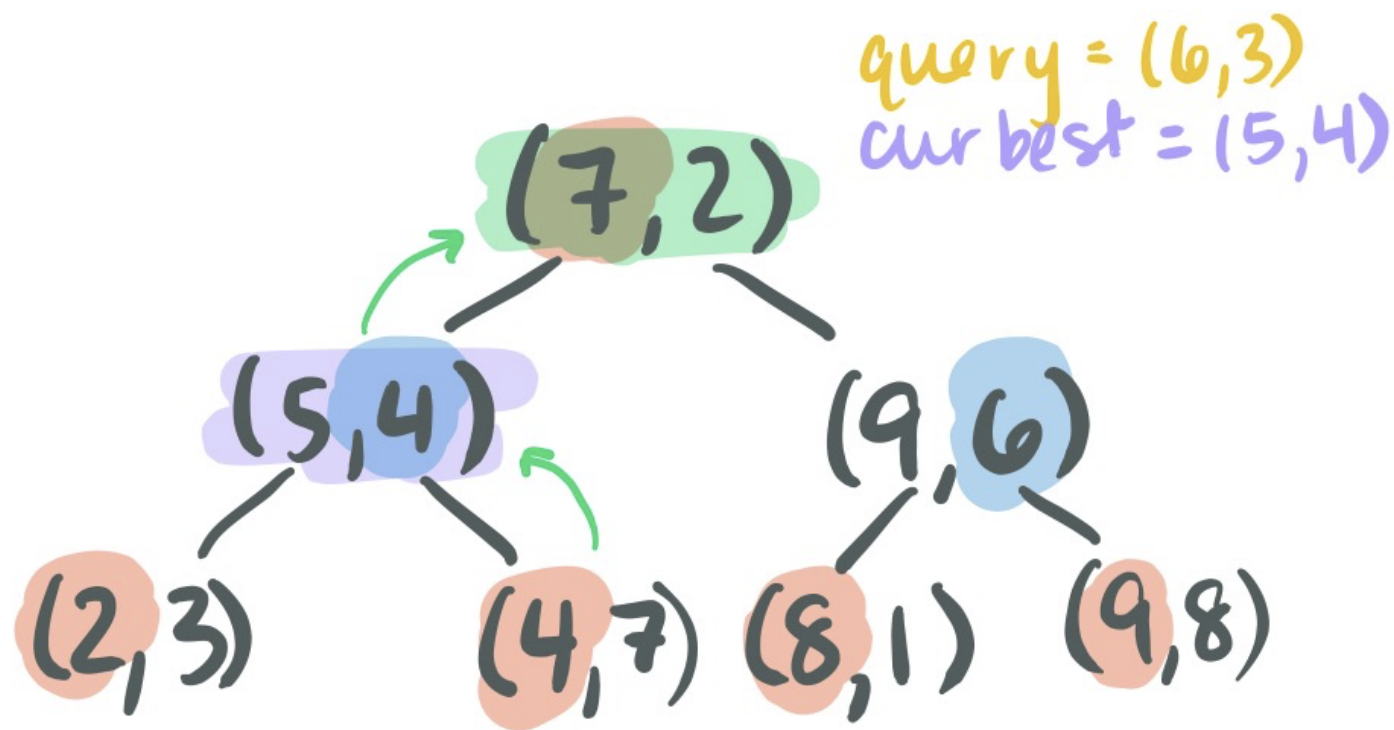


Nearest Neighbor: k-d tree



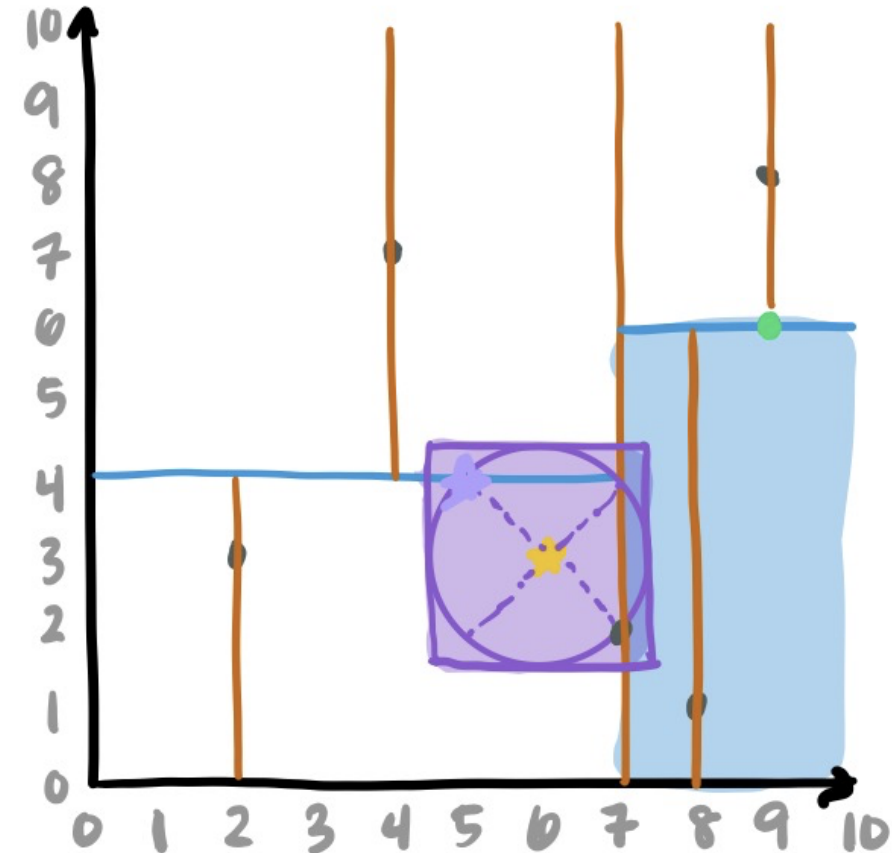
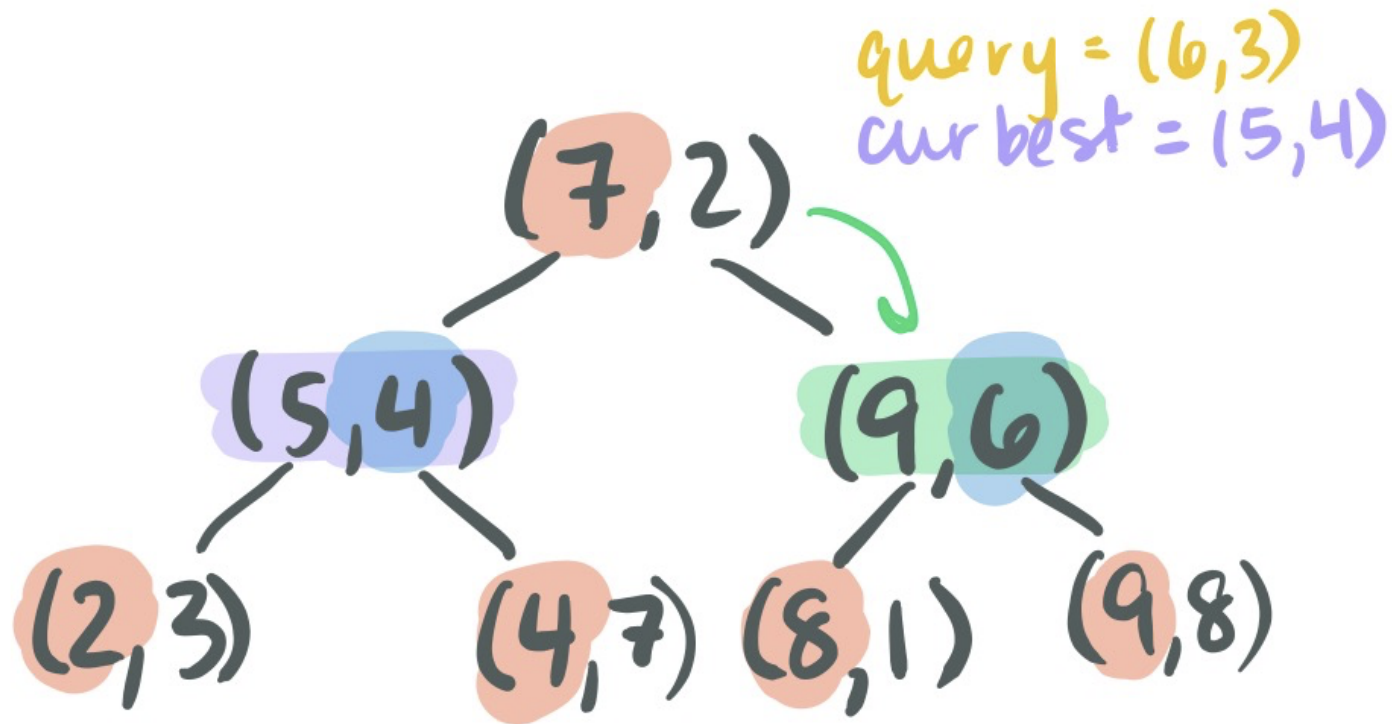
Nearest Neighbor: k-d tree

On ties, use smallerDimVal to determine which point remains curBest

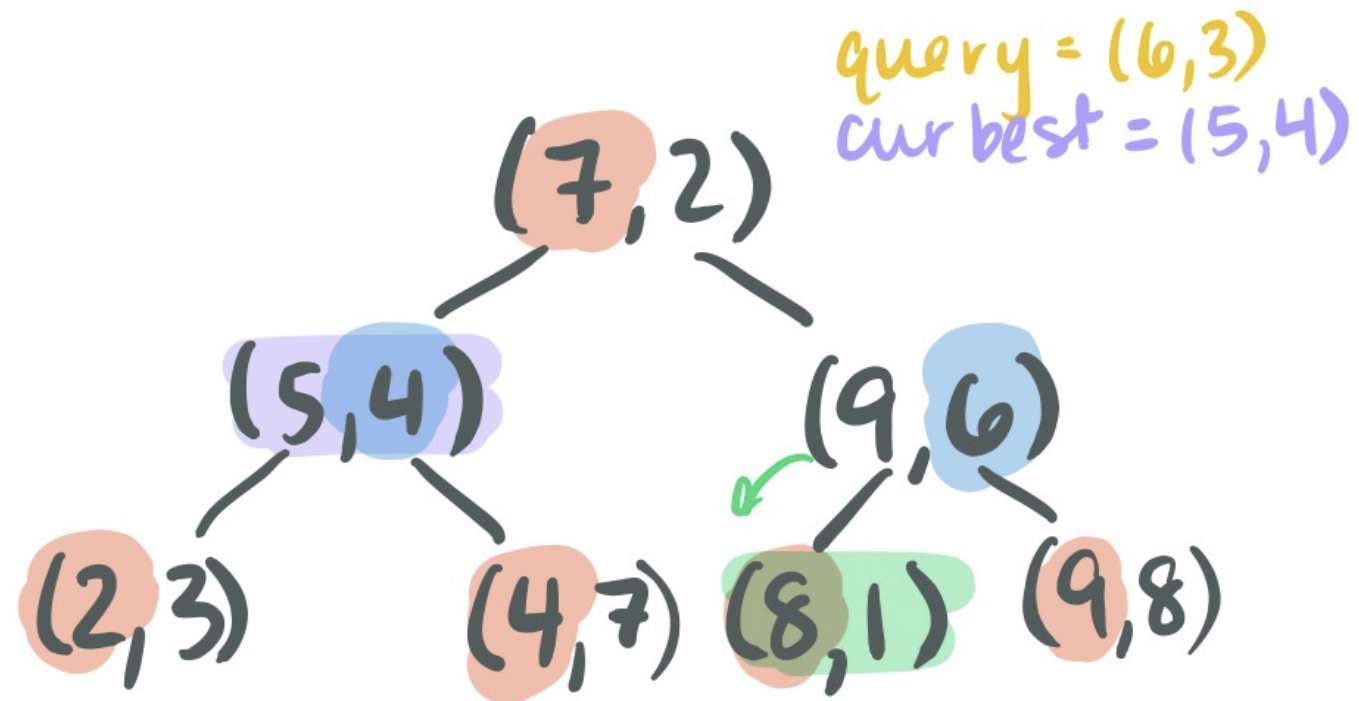


Nearest Neighbor: k-d tree

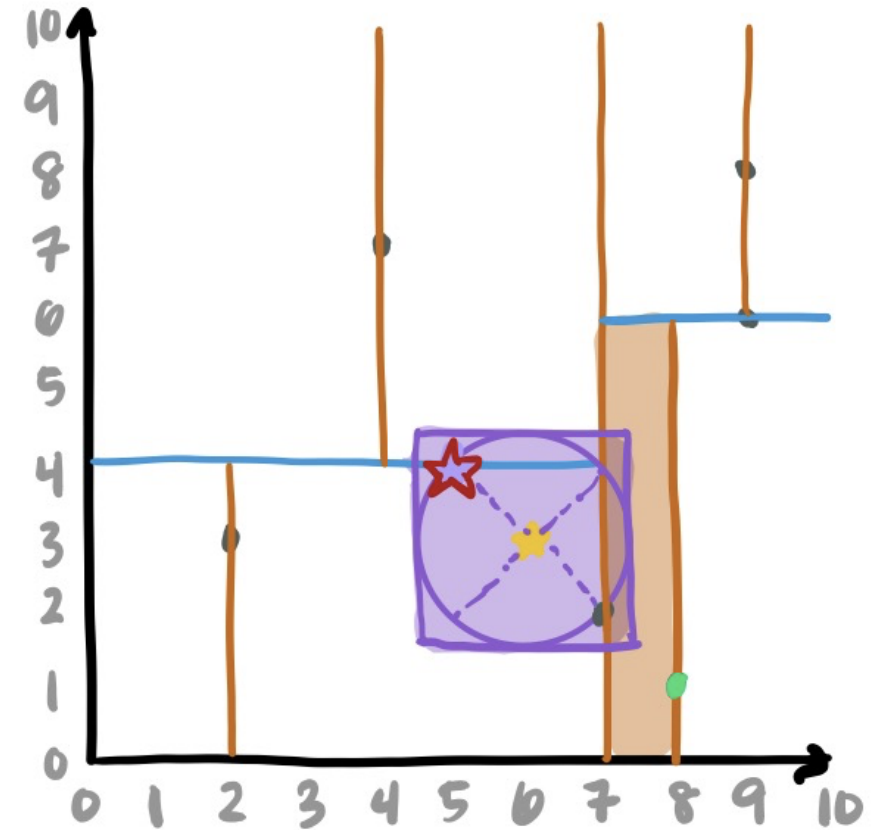
Why do we need to explore this subtree?



Nearest Neighbor: k-d tree



BEST: (5,4)



K-D Tree Runtime Efficiency

Because our search radius can touch every point, search is $O(N)$!

... But if my data is well-formed (uniform distribution), we can make an argument that the average-case search is actually $O(\log n)$

Intuition: More points == more partitions == smaller search radius

MP_Mosaics shows off this theory vs practice!

Next Class...

There's an important algorithm we are neglecting...

Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

Tree Construction:

1. Find **median point** along a particular dim
2. Recurse left and right, swapping dimension

We will also need to understand a few subtle C++ concepts!