

Data Structures

KD Tree

CS 225

Harsha Tirumala

September 25, 2025



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Learning Objectives

BST Remove : Recap

KD tree : Motivation and Creation

KD tree : Interval Search and Nearest Neighbor

Go over C++ concepts for mp_mosaics

remove (51)

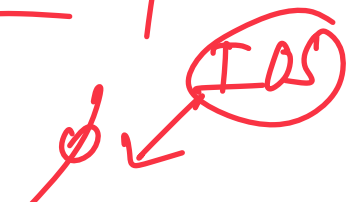
✓ 51738

-
- Hand-drawn diagram of a node in a graph. The node is a circle containing a crossed-out '1'. It is connected to other nodes by red and blue arrows. Red handwritten text '51=51' and '66' are next to it.

TOP

1 child

O child



Range Search : Motivating example

Amazon.com : protein bar

+

← → ↻ 🏠

🔍 https://www.amazon.com/s?k=protein+bar&crd=37M386KEQMCP&srefix=protein+bar%2Caps%2C145&ref=nb_sb_ss_p1...

☆

🔍

🔊

🔒 Enter passphrase

🔄 Finish update

📦

🇨🇵 CPP

🕒 35 years ago, Isaa...

🔖 All Bookmarks

☰ All

👤 Rufus

Amazon Haul

Early Prime Deals

Same-Day Delivery

Medical Care ▾

Luxury Stores

Amazon Basics

Pet Supplies

Household, Health & Baby Care

Amazon Home

Pharmacy

1-48 of over 7,000 results for "protein bar"

Sort by: Featured ▾

Popular Shopping Ideas

Vegan

Peanut Butter

Gluten Free

Crunch

See more

Eligible for Free Shipping

☐ Free Shipping by Amazon

Get FREE Shipping on eligible orders shipped by Amazon

Delivery Day

☐ Get It Today

☐ Get It by Tomorrow

Customer Reviews

★★★★☆ & Up

Deals & Discounts

All Discounts

Today's Deals

Price

\$1 – \$330+

Go

PERFECT BAR

15g 20g

THE ORIGINAL REFRIGERATED PROTEIN BAR

PERFECT BAR

DARK CHOCOLATE CHIP PEANUT BUTTER

Organic Whole Food Protein Bars

Shop Perfect Bar >

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

Real ingredients

10 BARS

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

Perfect Bar, Dark Chocolate Chip Peanut Butter Protein...

4.2 ★★★★★ (4.8k)

prime

24 BARS

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

Perfect Bar Mini, Peanut Butter Protein Bar, Protein...

4.4 ★★★★★ (984)

prime

10 BARS

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

Perfect Bar, Chocolate Chip Cookie Dough Protein Bar,...

4.2 ★★★★★ (4.8k)

prime

Sponsored ⓘ

Results

Check each product page for other buying options.

24 BARS

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PERFECT BAR

PRIMA

ANCESTRAL PROTEIN BAR

100% GRASS-FED PROTEIN

NO SEED OILS

NO REFINED SUGAR

PRIMA

Cacao

ANCESTRAL PROTEIN BAR

100% GRASS-FED PROTEIN

NO SEED OILS

NO REFINED SUGAR

ONE

Chocolate Chip Cookie Dough

12 BARS

20g PROTEIN

1g SUGAR

ONE

Chocolate Chip Cookie Dough

12 BARS

20g PROTEIN

1g SUGAR

PROTEIN CHEWY BARS

PEANUT BUTTER

DARK CHOCOLATE

PROTEIN CHEWY BARS

PEANUT BUTTER

DARK CHOCOLATE

PROTEIN CHEWY BARS

PEANUT BUTTER

DARK CHOCOLATE

PROTEIN CHEWY BARS

PEANUT BUTTER

DARK CHOCOLATE

Want Protein bars that cost between 10\$ and 20\$

Range-based Searches

→ #queries ↑ ⇒ Pre computation can pay off

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

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

↙ $A = 4$ $B = 35$
 $\#queries = n$

① Brute force:

① BF $O(n) \times n^2 = O(n^3)$

② Sorted then

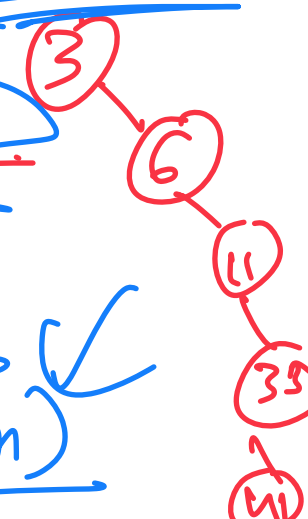
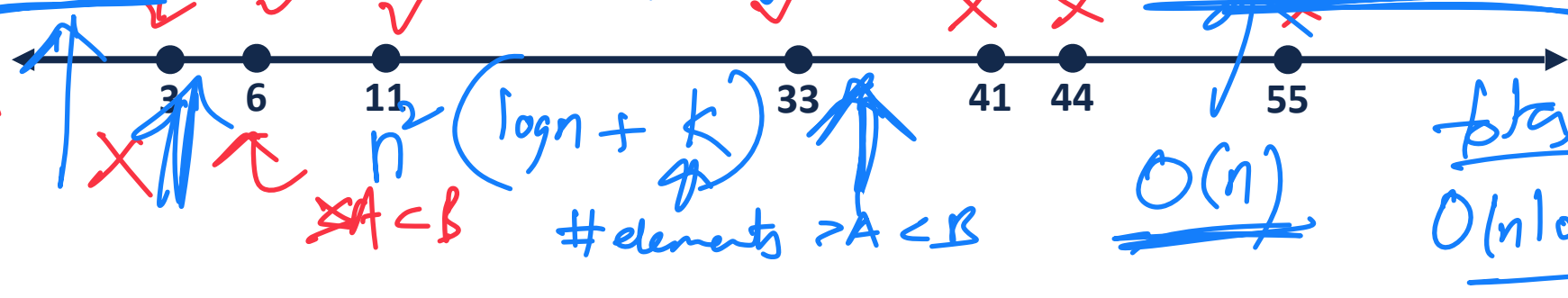
$A \leftarrow O(\log n)$ $O(\log n) \rightarrow B$
 Binary Search $T(n) = O(n)$

② BS: $n \log n + n^2 (\log n)$
 $Pric \approx O(n^2 \log n)$

iterate $O(n \log n)$
 merge sort

for $A \& B$
 iterate from A to B

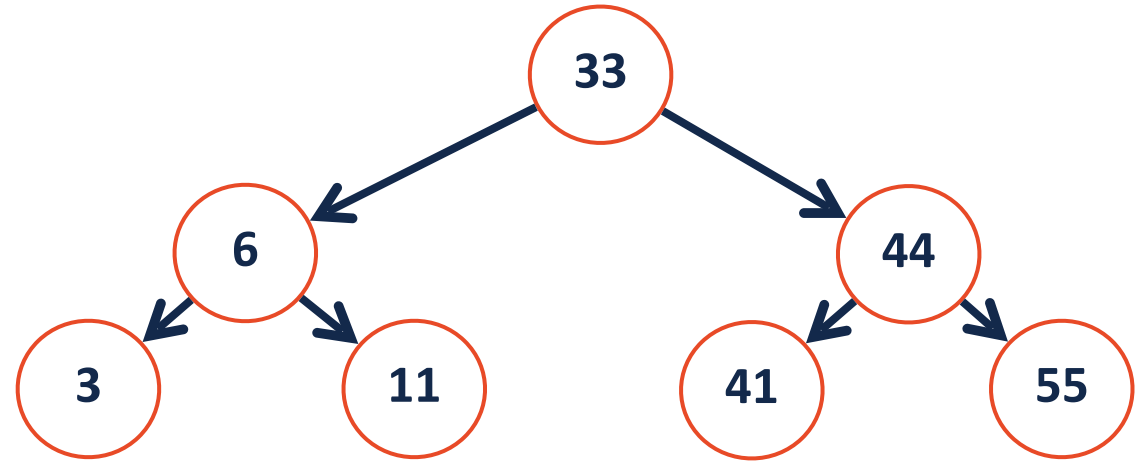
of protein
 $O(n^3)$



Range-based Searches

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?

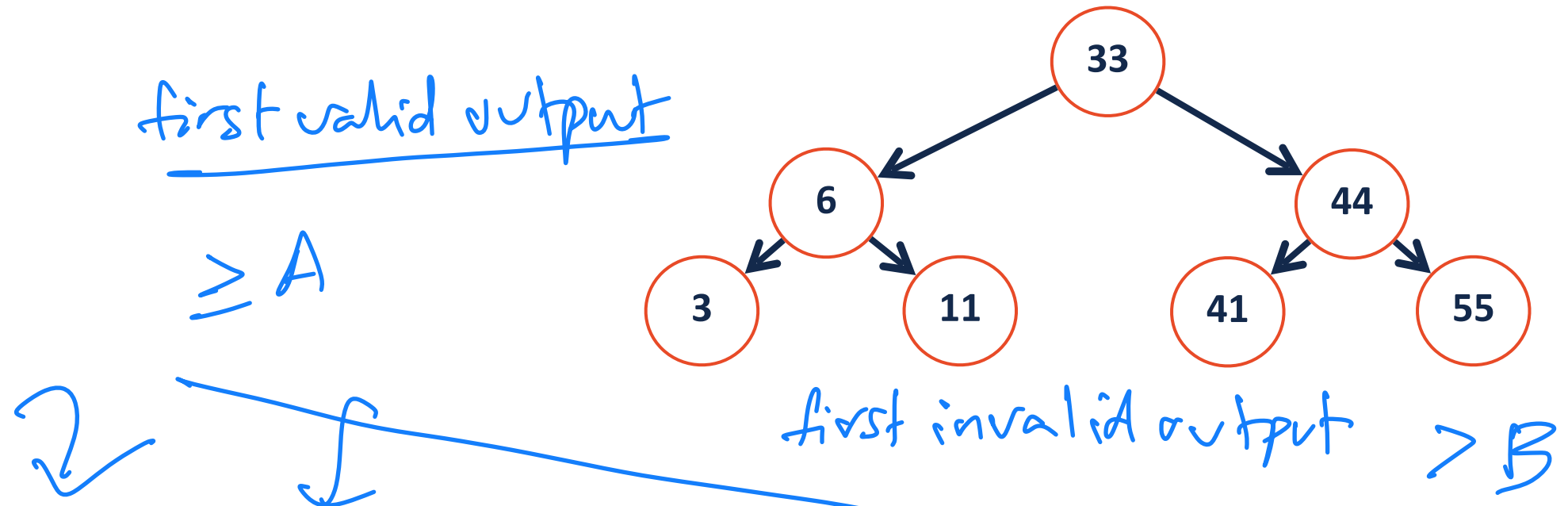


Range-based Searches



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?

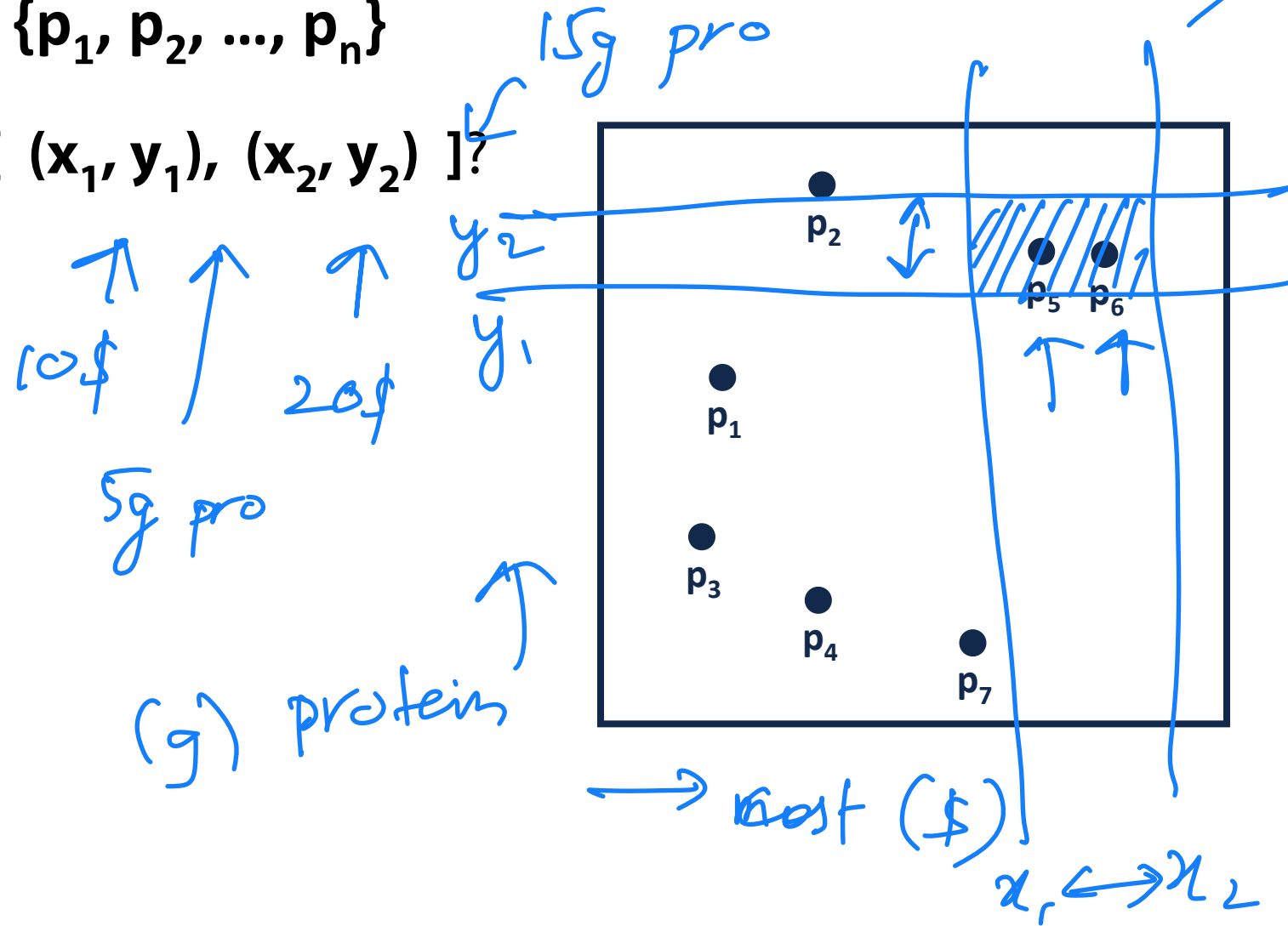


```
1
2 for(auto it = myMap.lower_bound(A); it != myMap.upper_bound(B); ++it){
3
4 // Do Stuff print(*it)
5 }
```

Range-based Searches

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

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



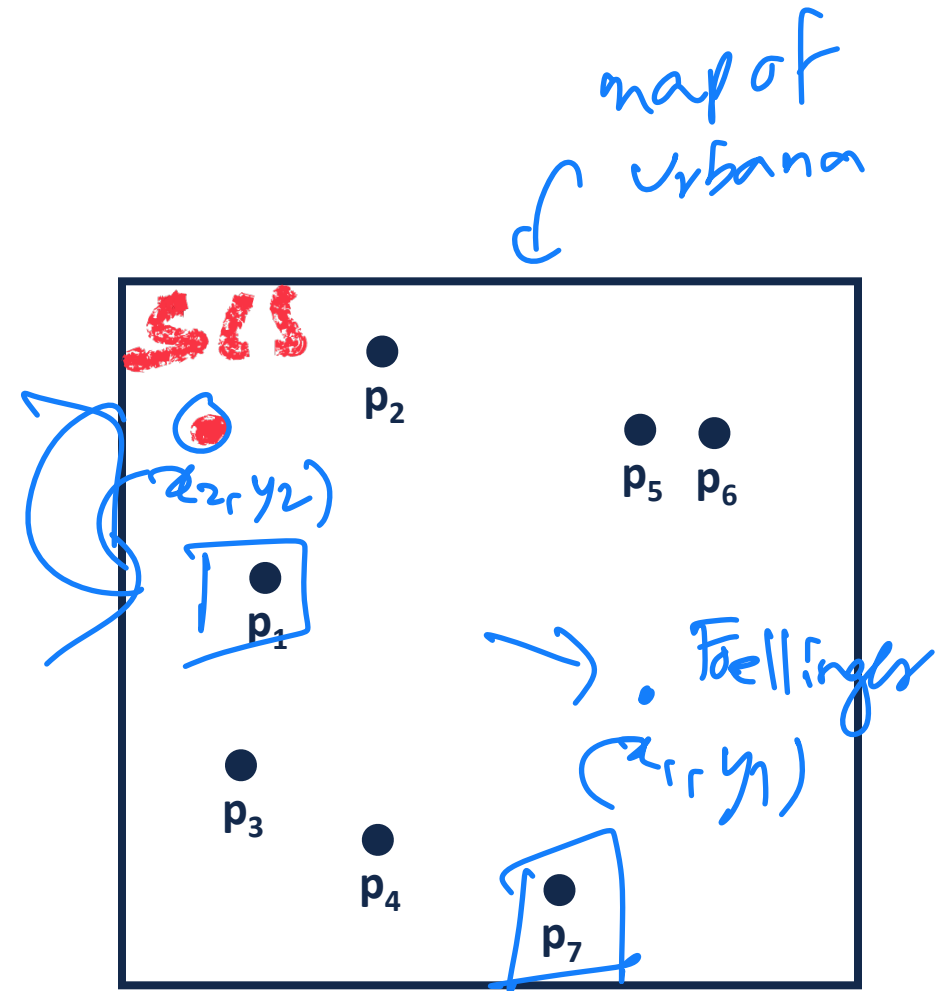
Range-based Searches

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

What is nearest point to (x_1, y_1) ?

query

closest to
bellinger

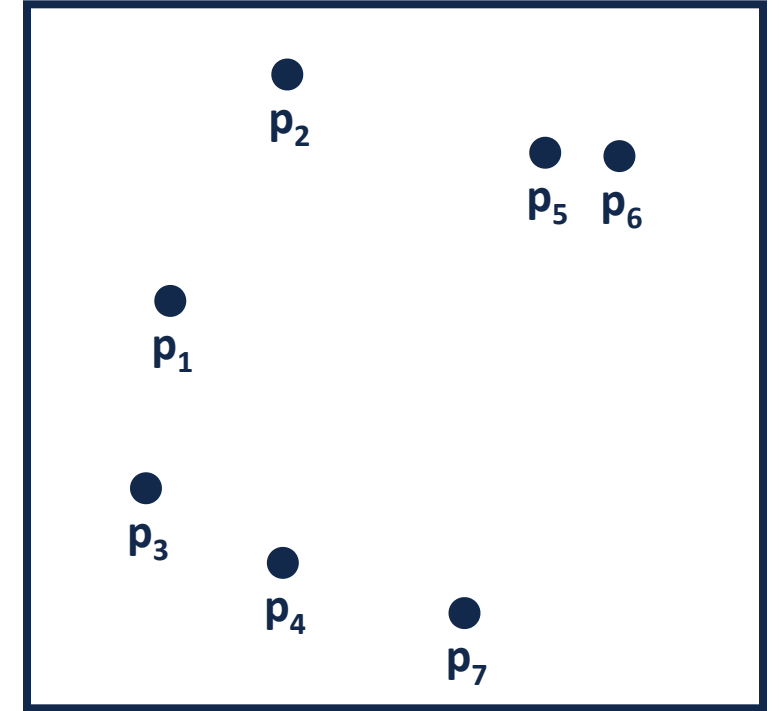


$p_i \rightarrow \text{Restaurant}_i$

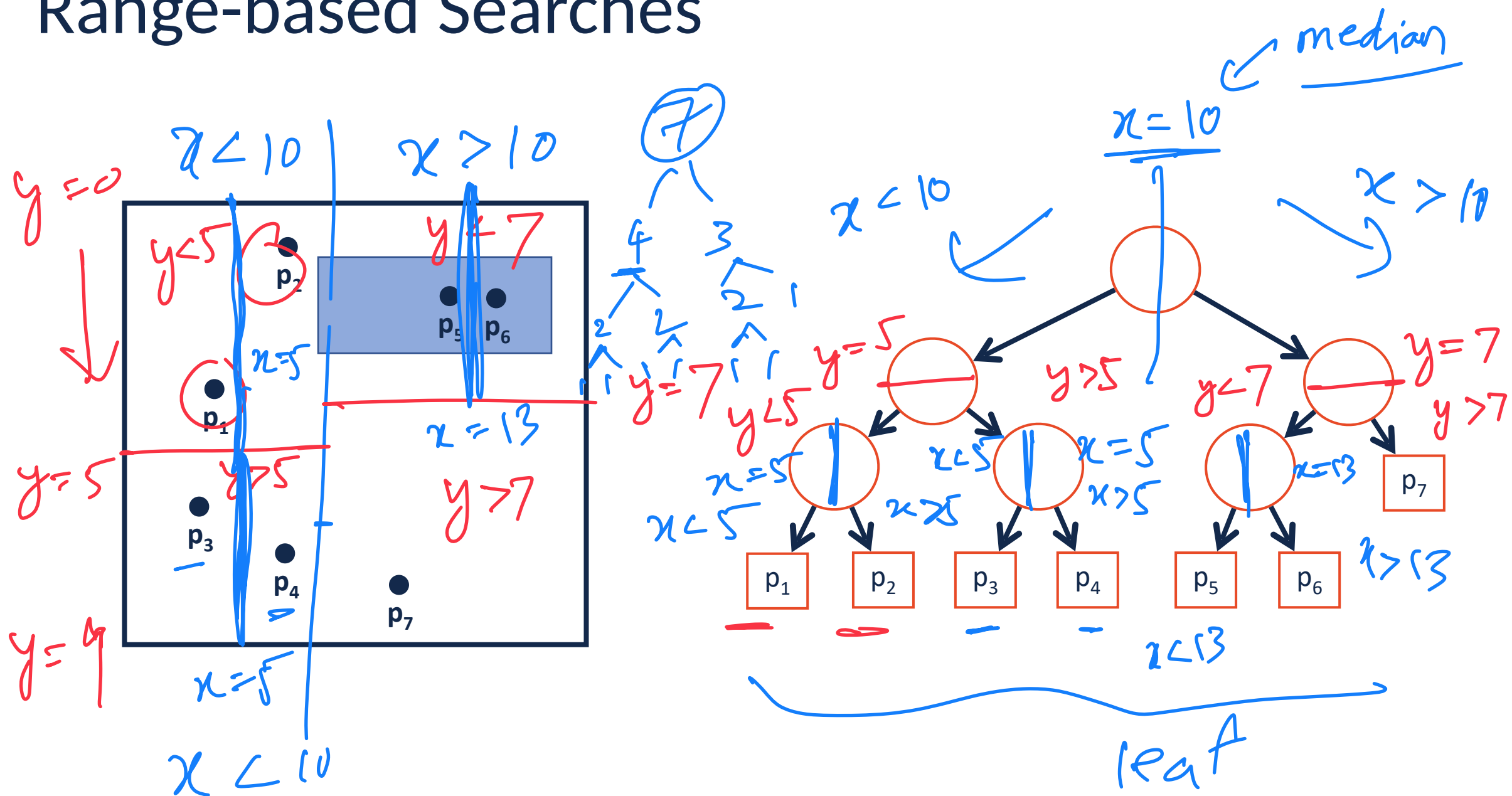
Range-based Searches

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

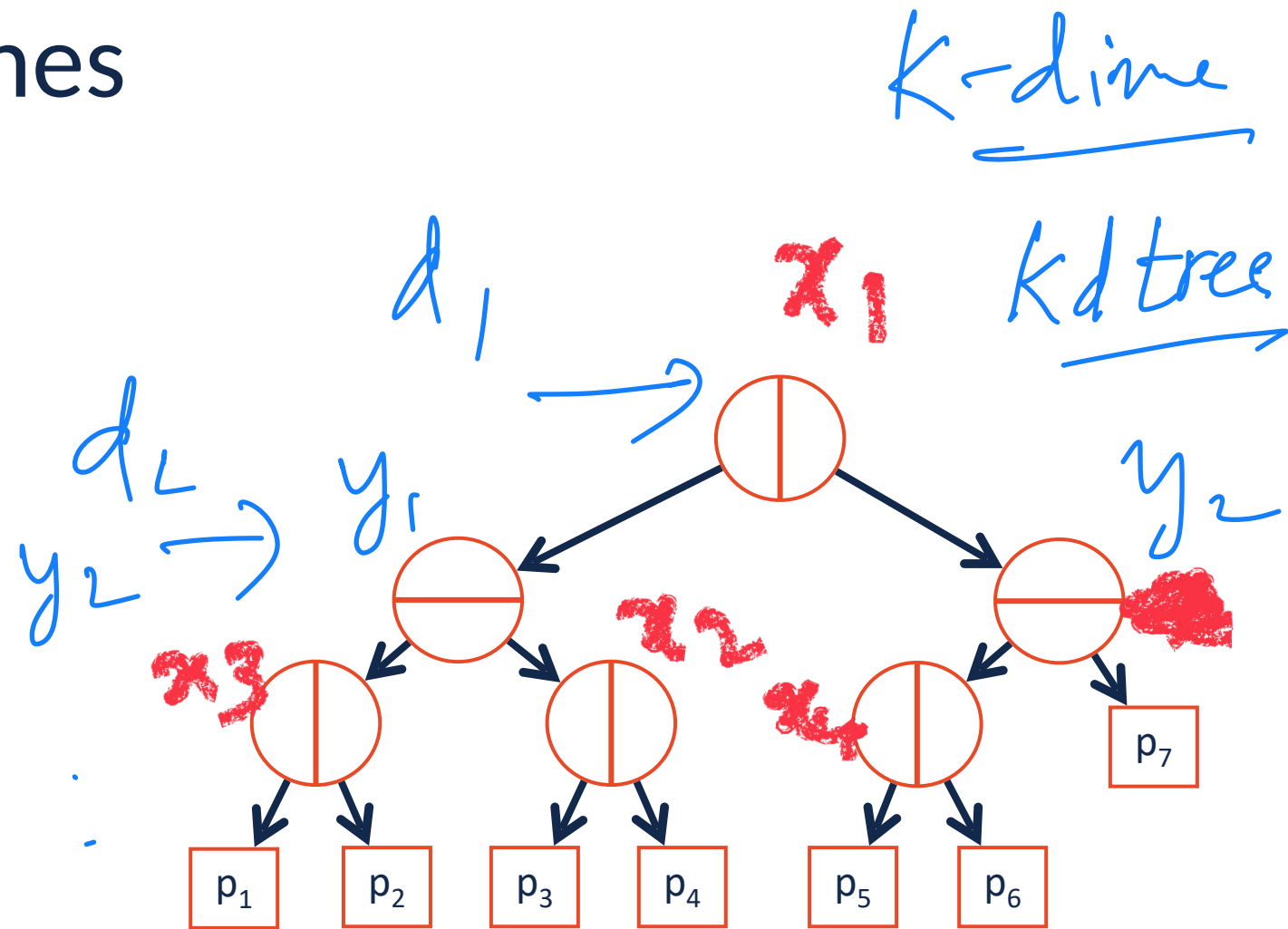
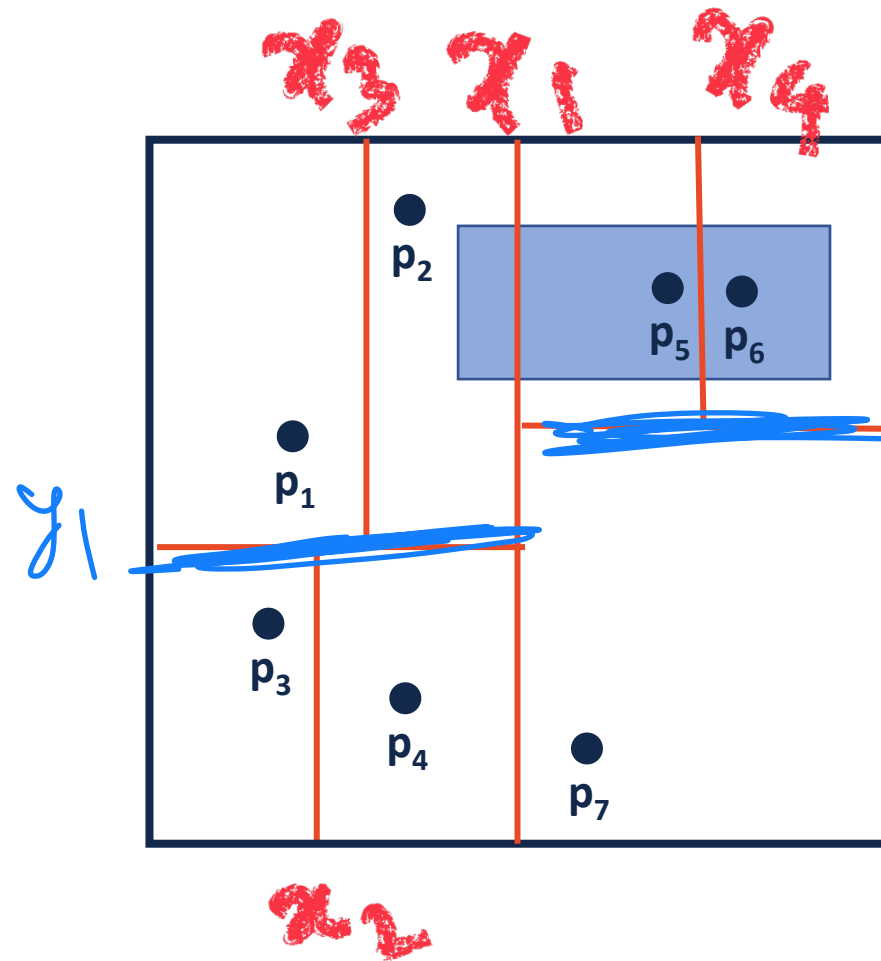
Tree Construction:



Range-based Searches



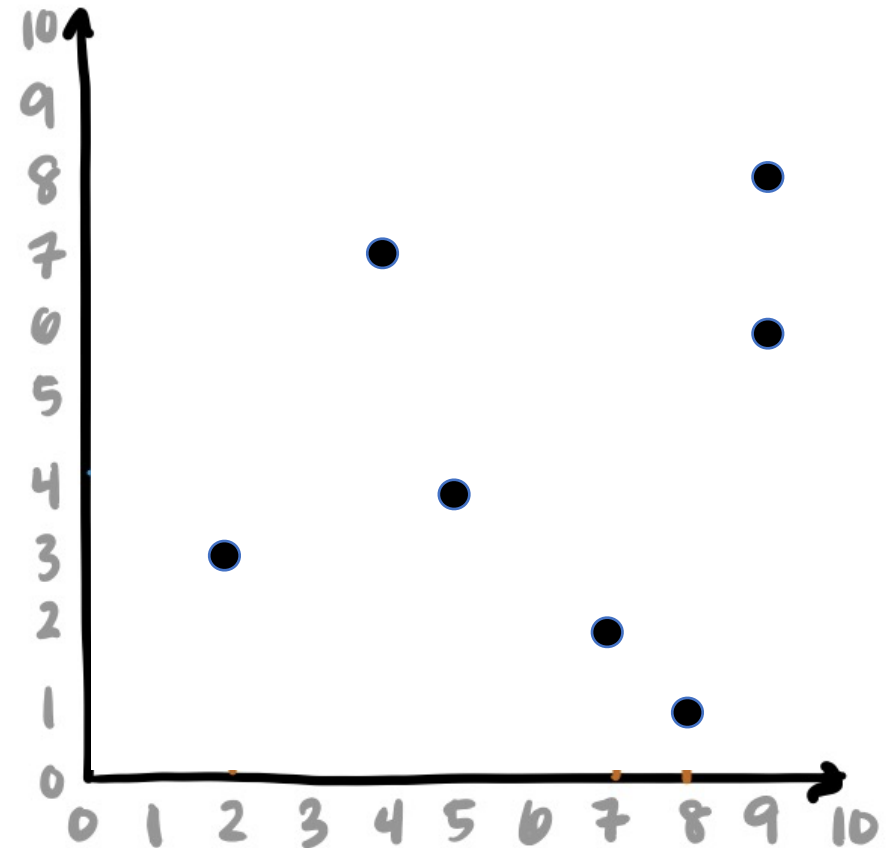
Range-based Searches



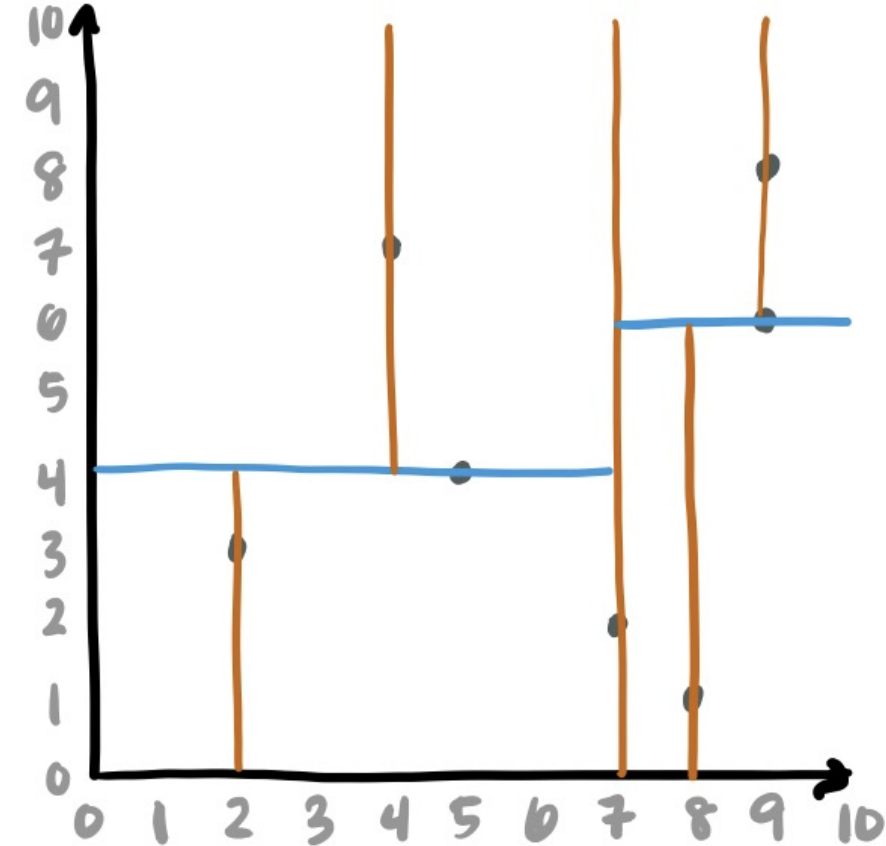
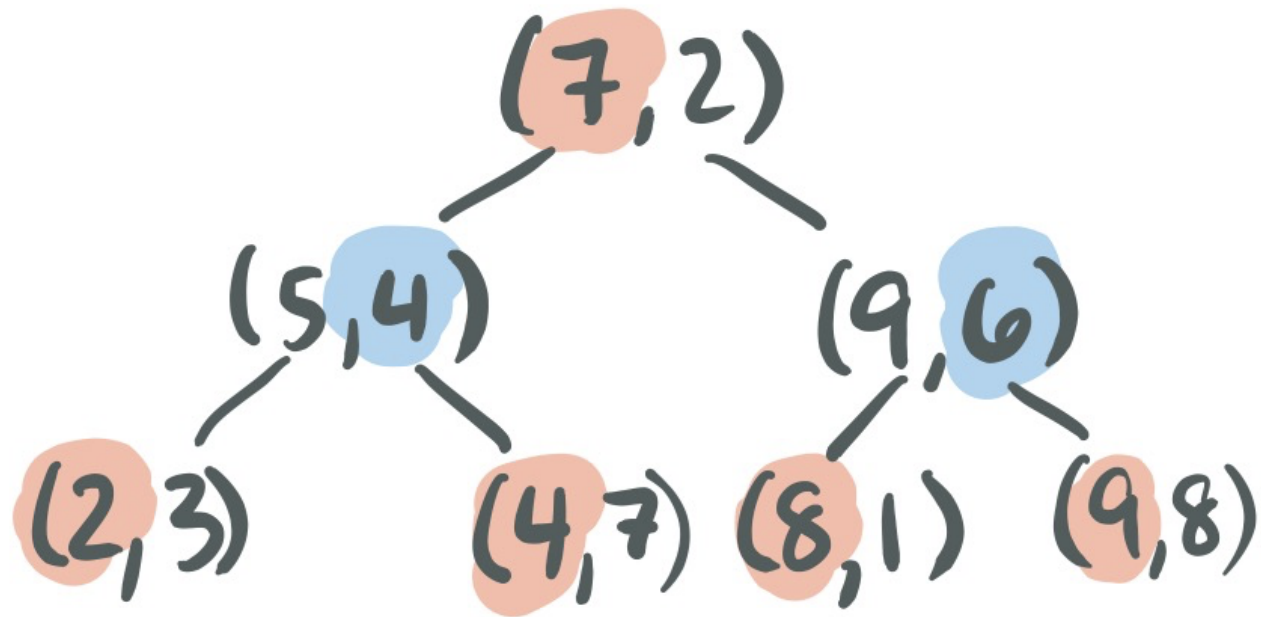
Nearest Neighbor: k-d tree

A **k-d tree** is similar but splits on points:

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

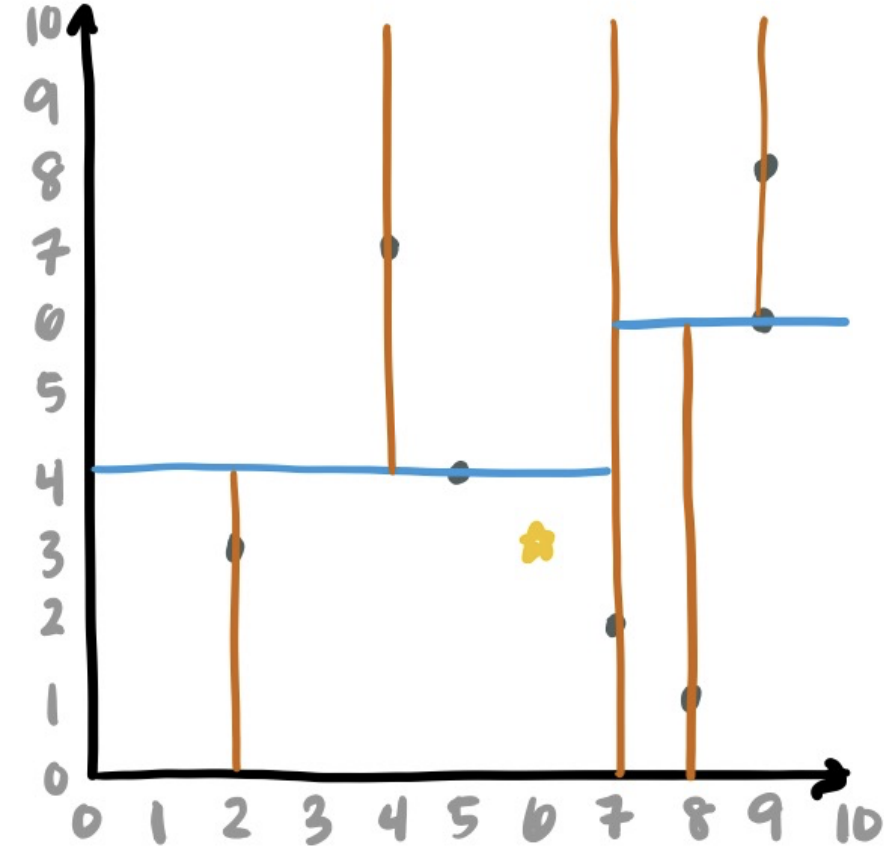
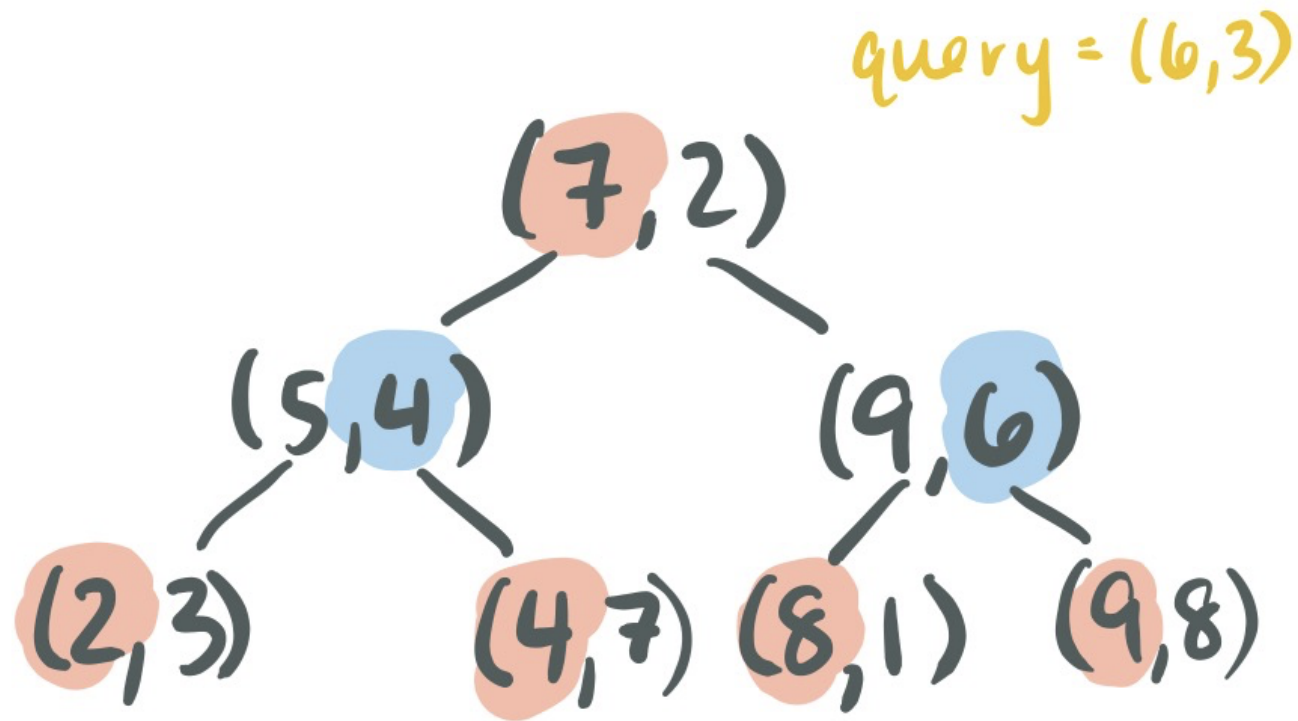


Nearest Neighbor: k-d tree



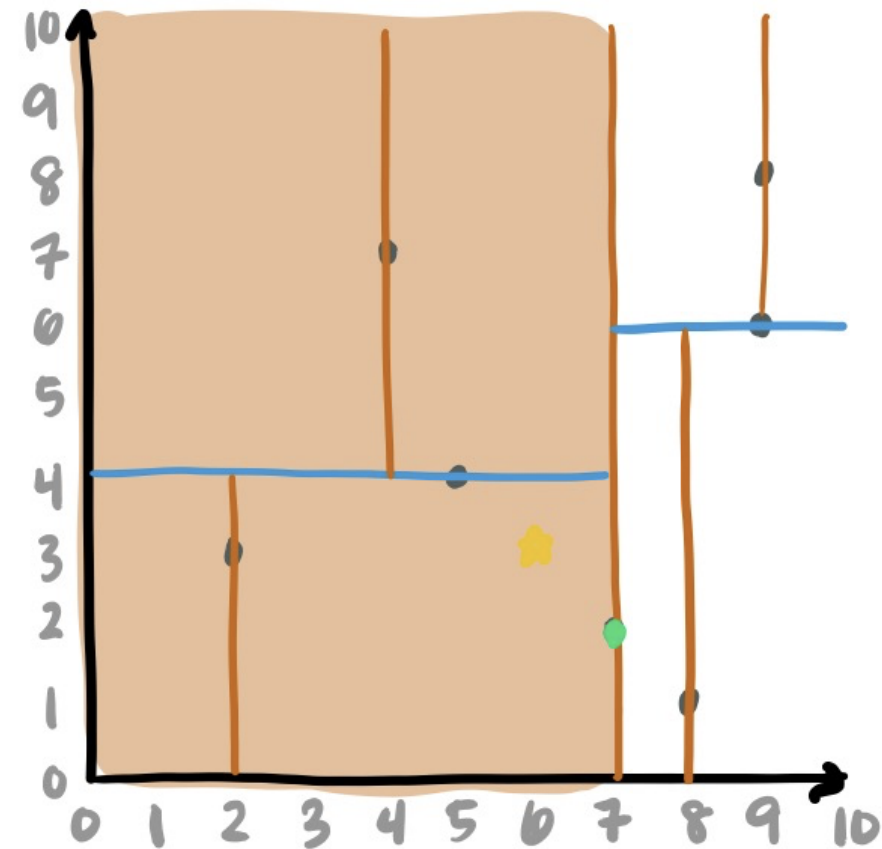
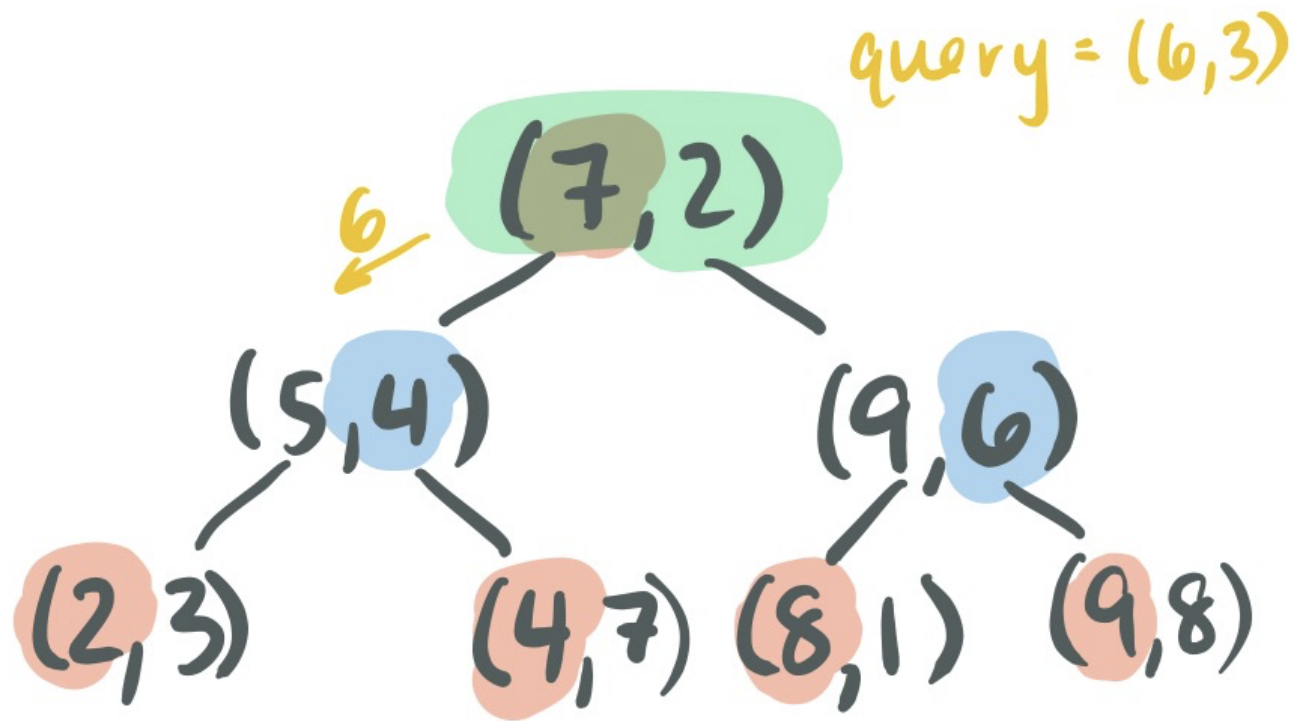
Nearest Neighbor: k-d tree

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



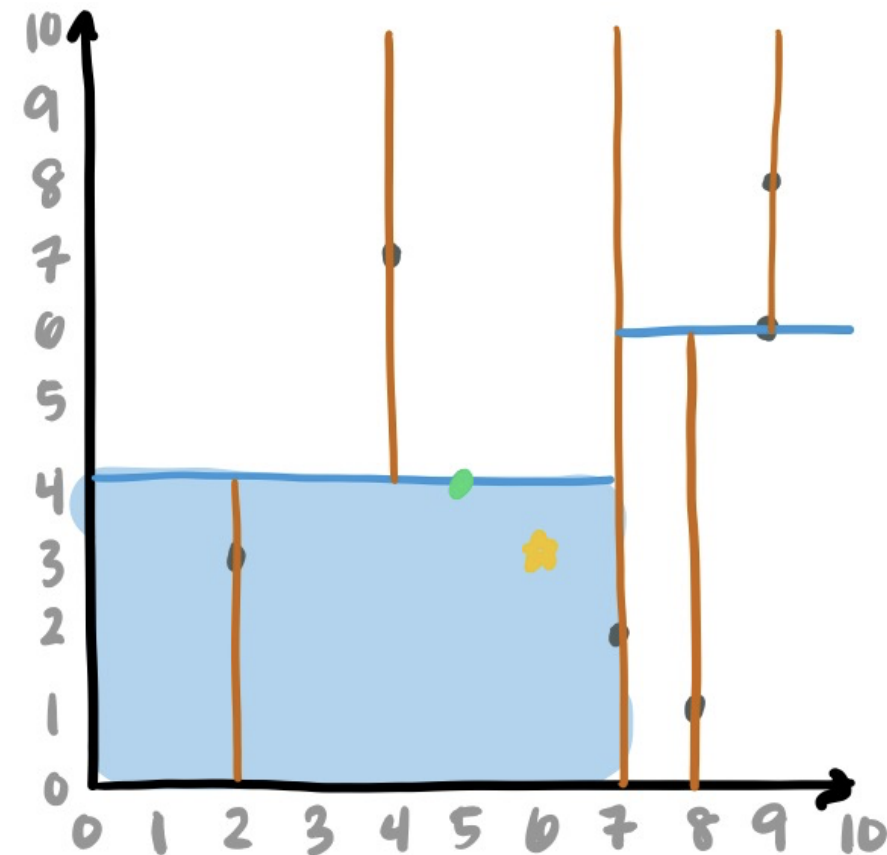
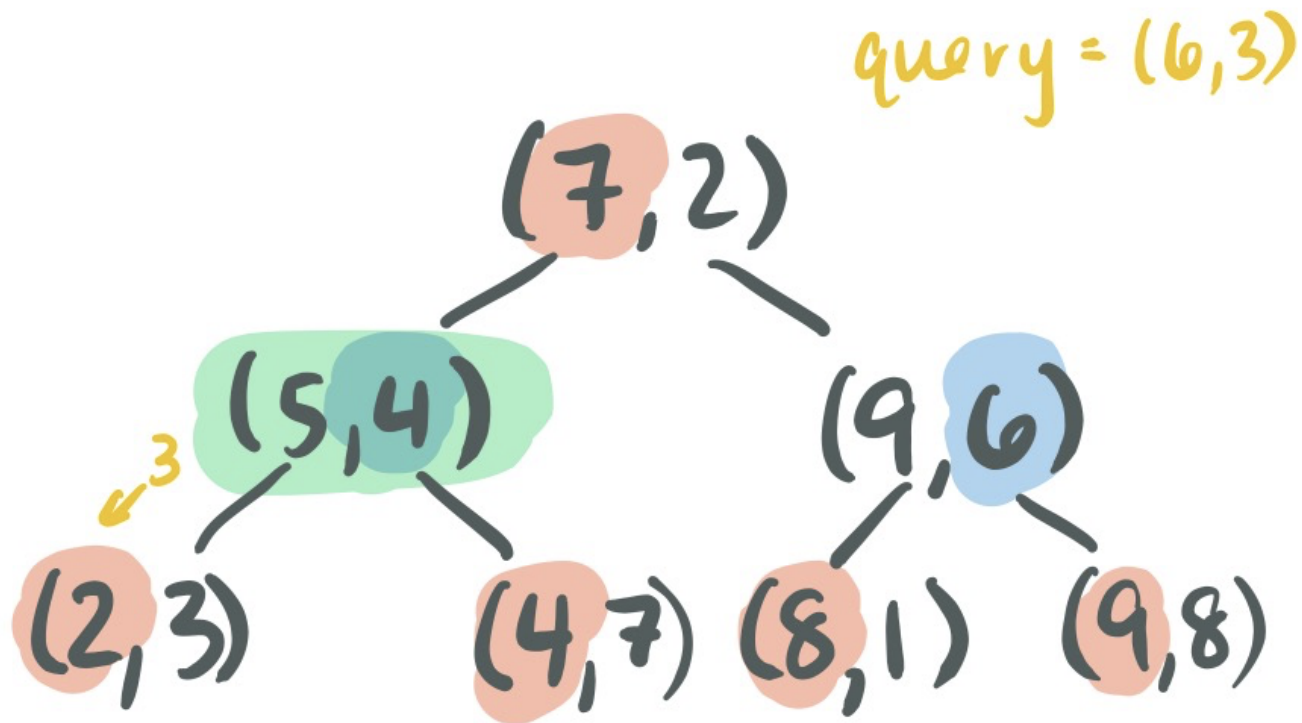
Nearest Neighbor: k-d tree

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



Nearest Neighbor: k-d tree

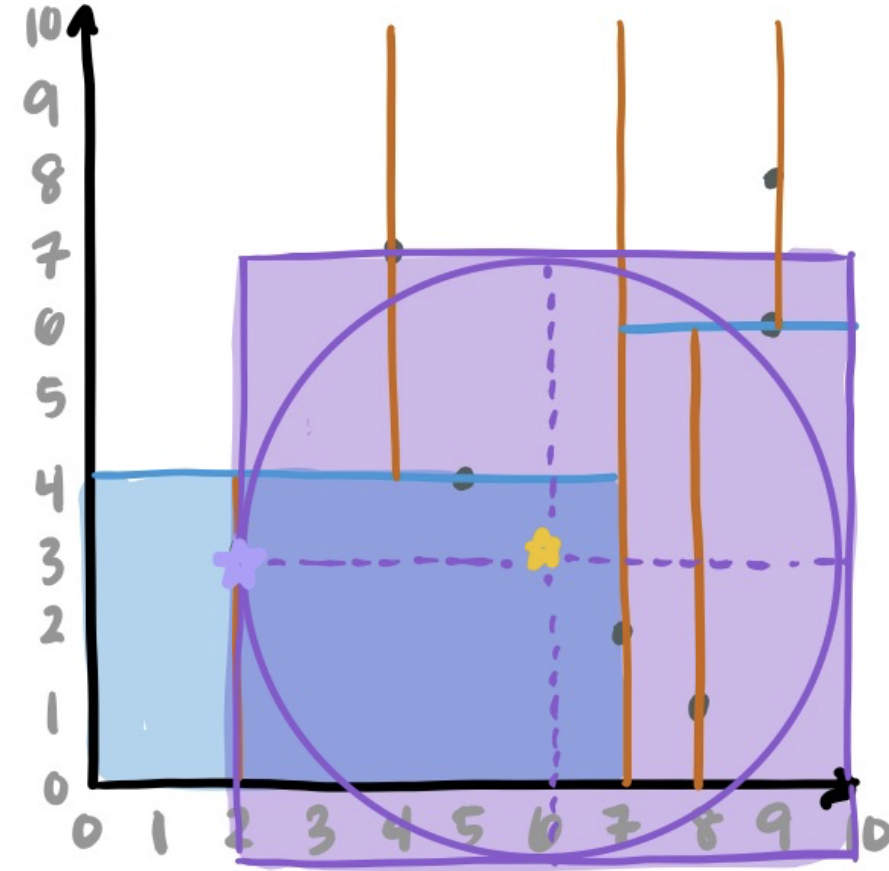
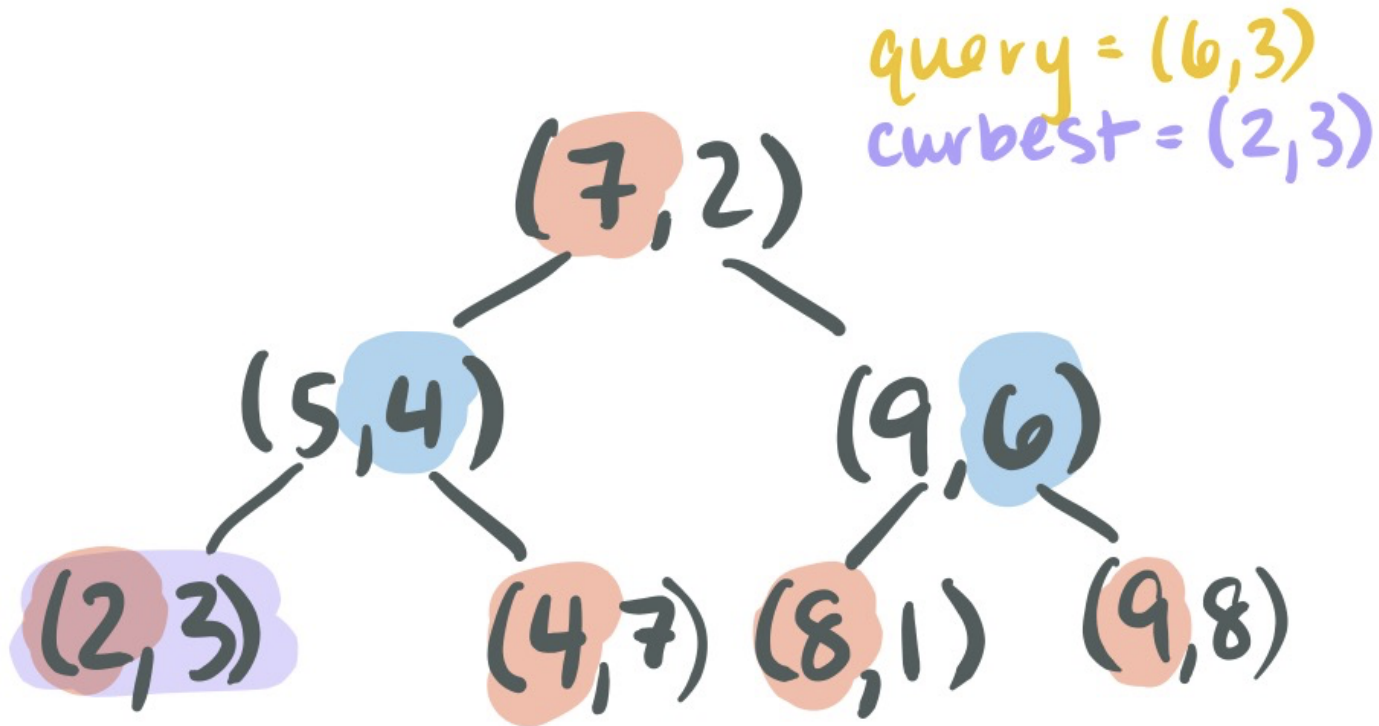
When querying a k-d tree, it acts like a BST* at first...



Nearest Neighbor: k-d tree

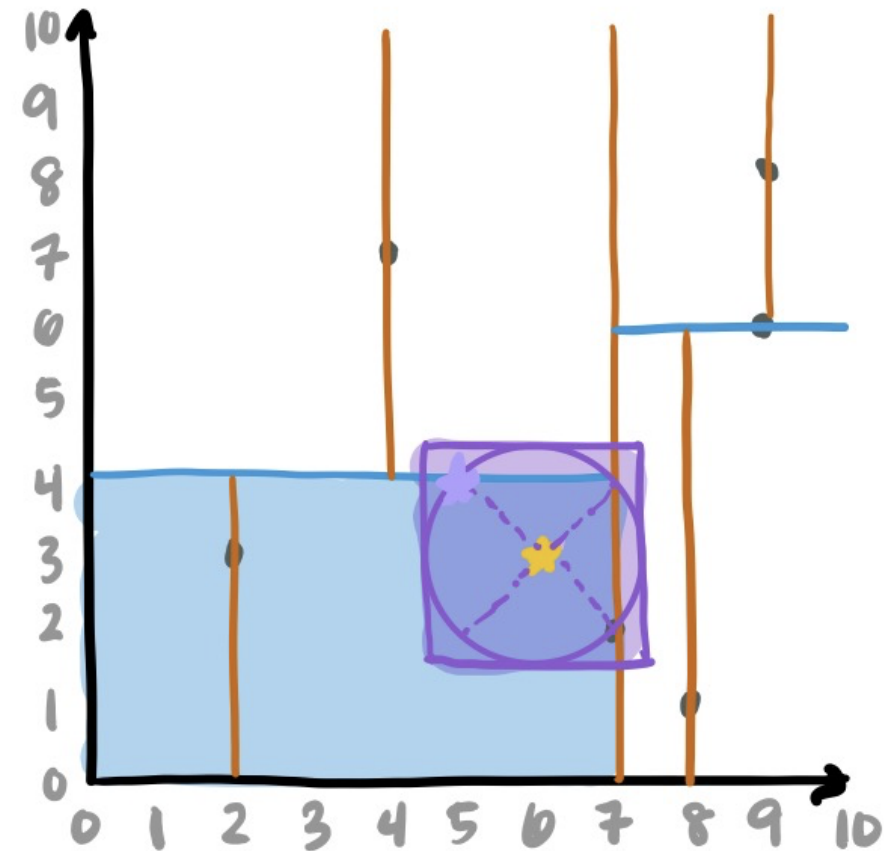
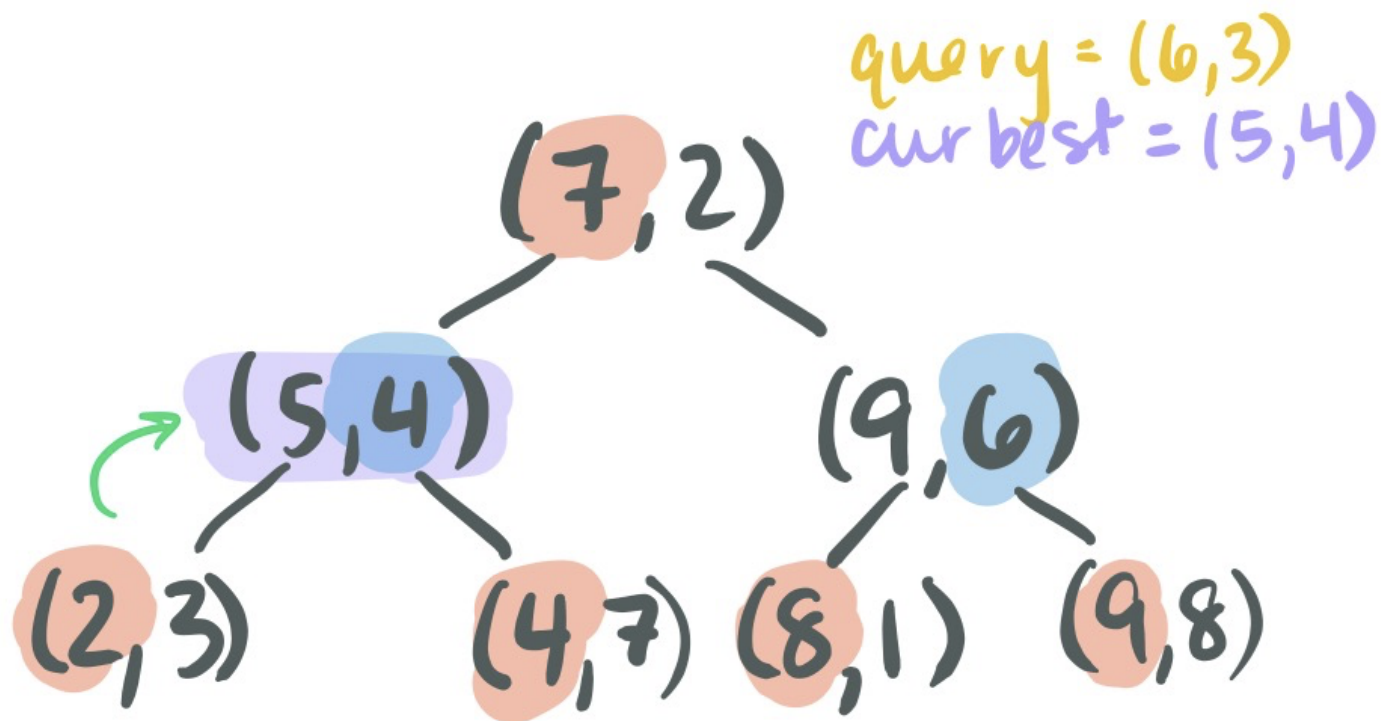
When querying a k-d tree, it acts like a BST* at first...

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

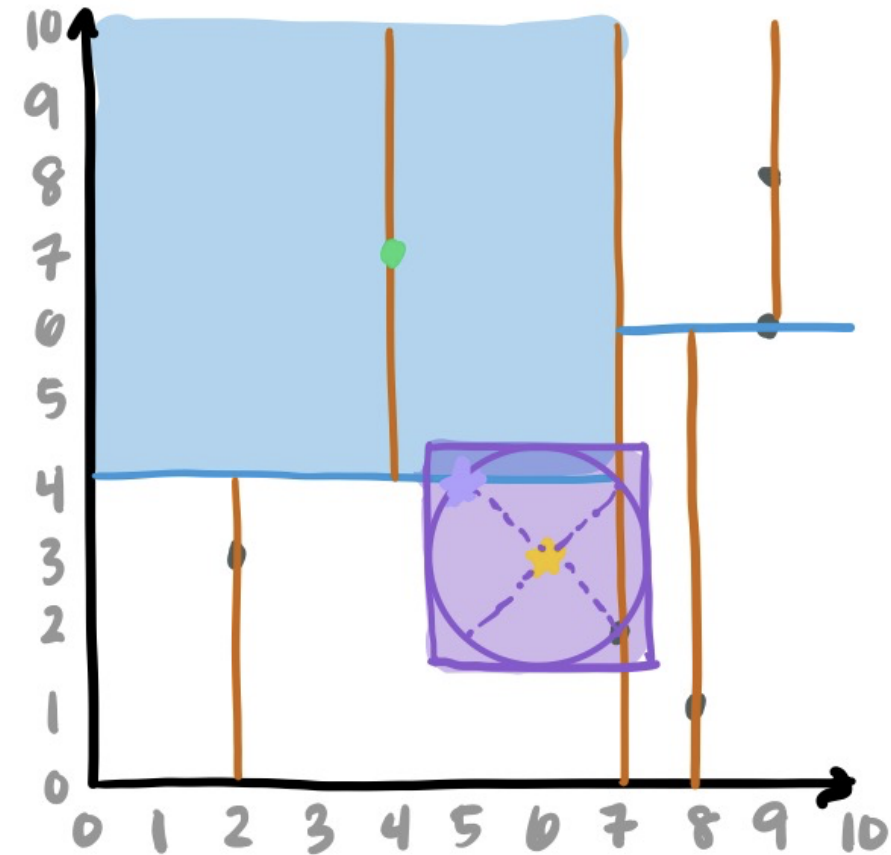
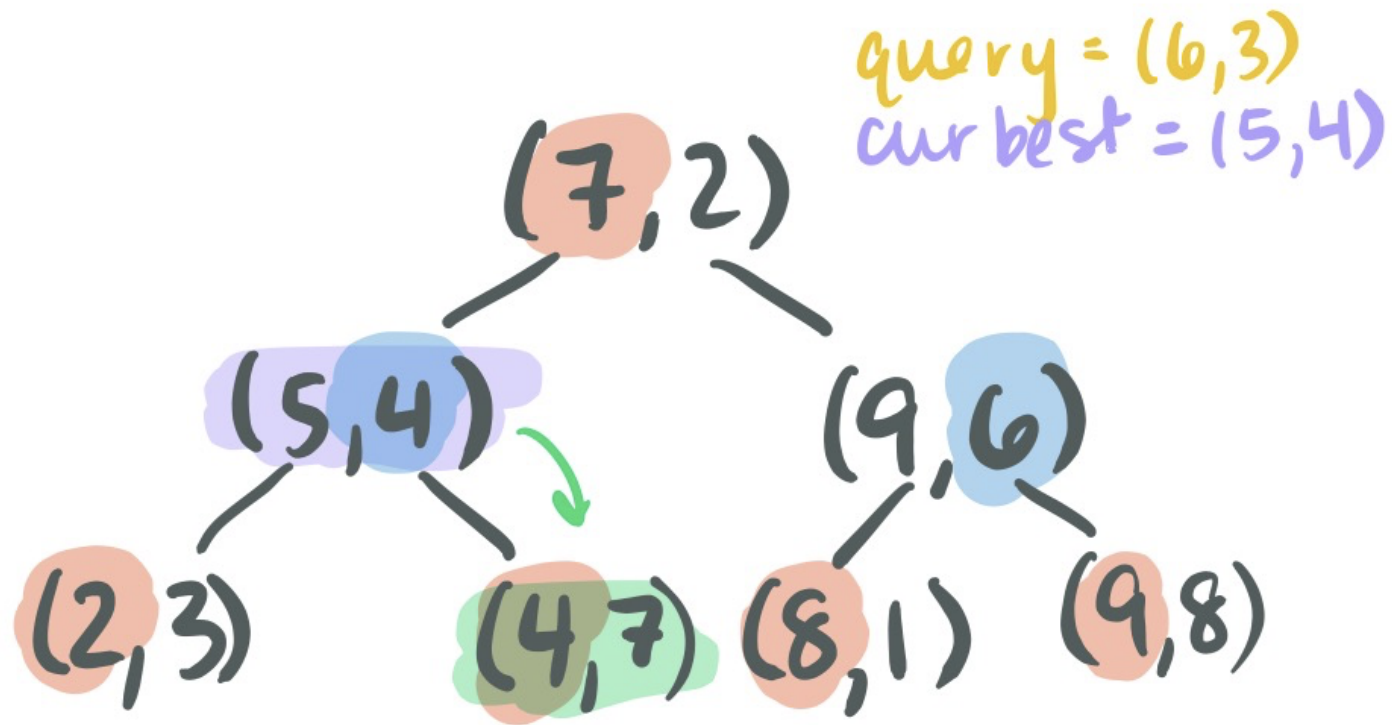


Nearest Neighbor: k-d tree

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

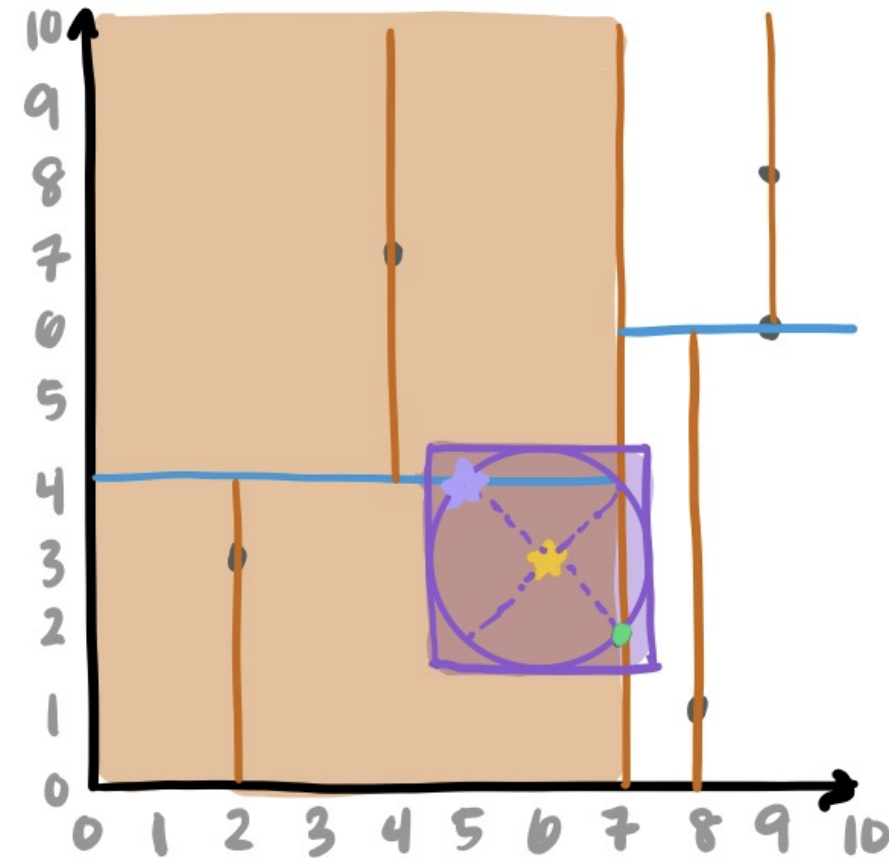
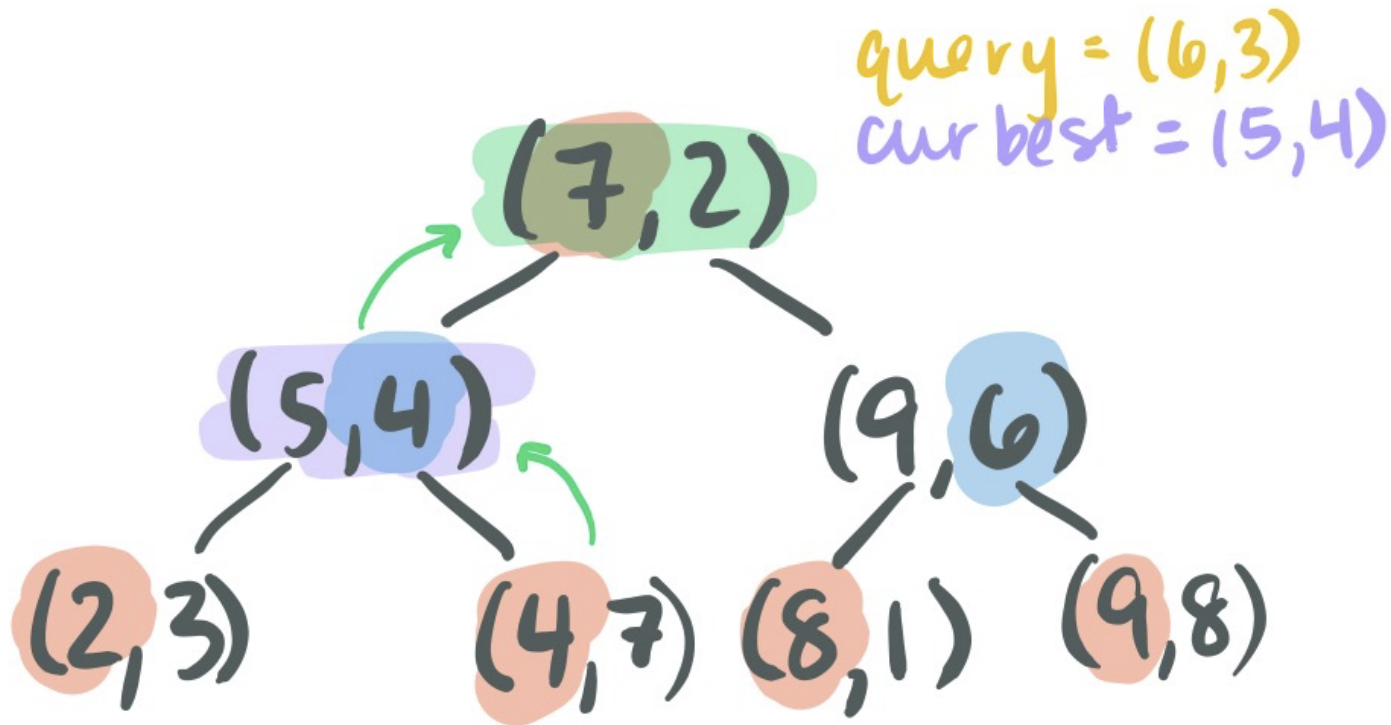


Nearest Neighbor: k-d tree



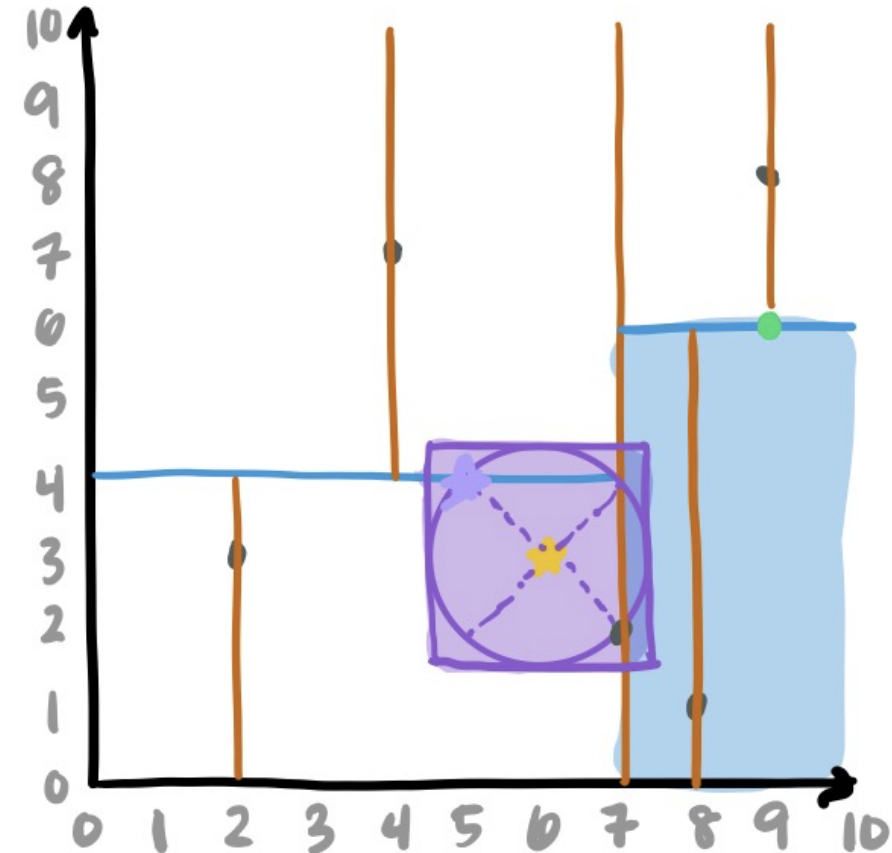
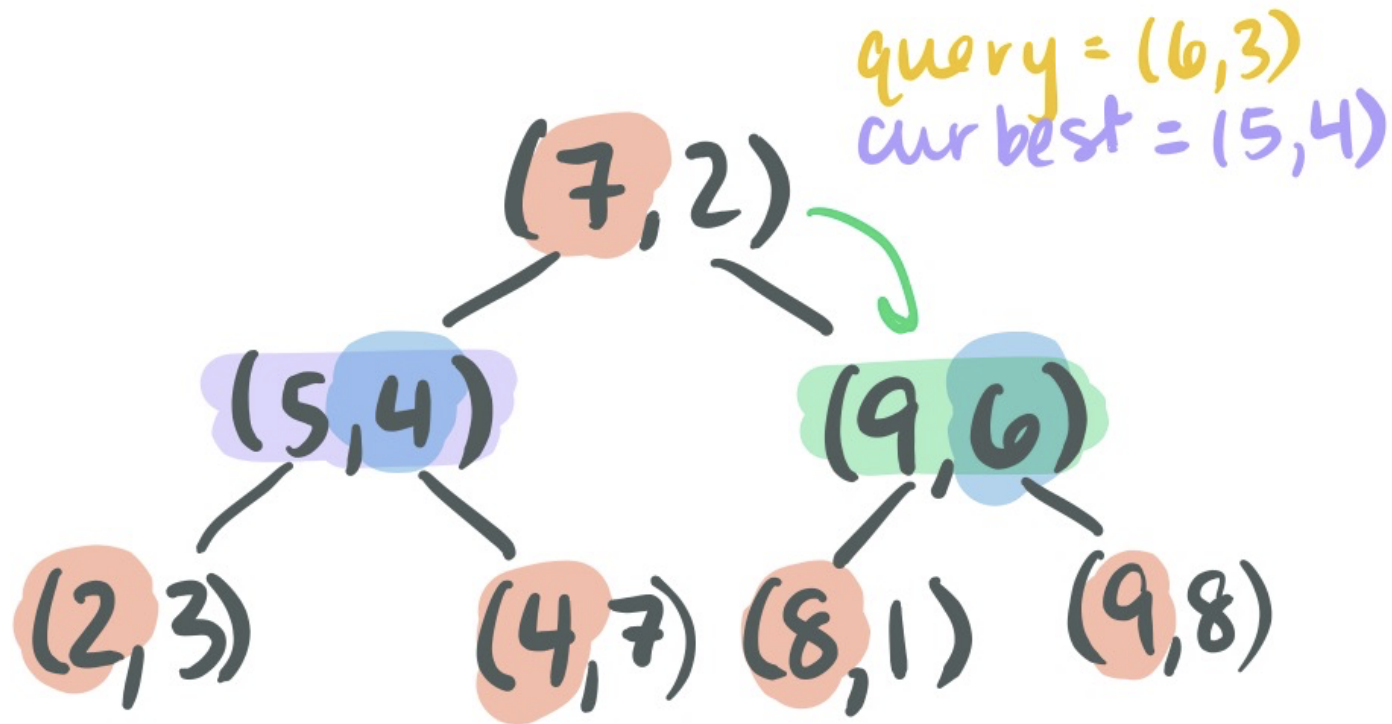
Nearest Neighbor: k-d tree

On ties, use smallerDimVal to determine which point remains curBest

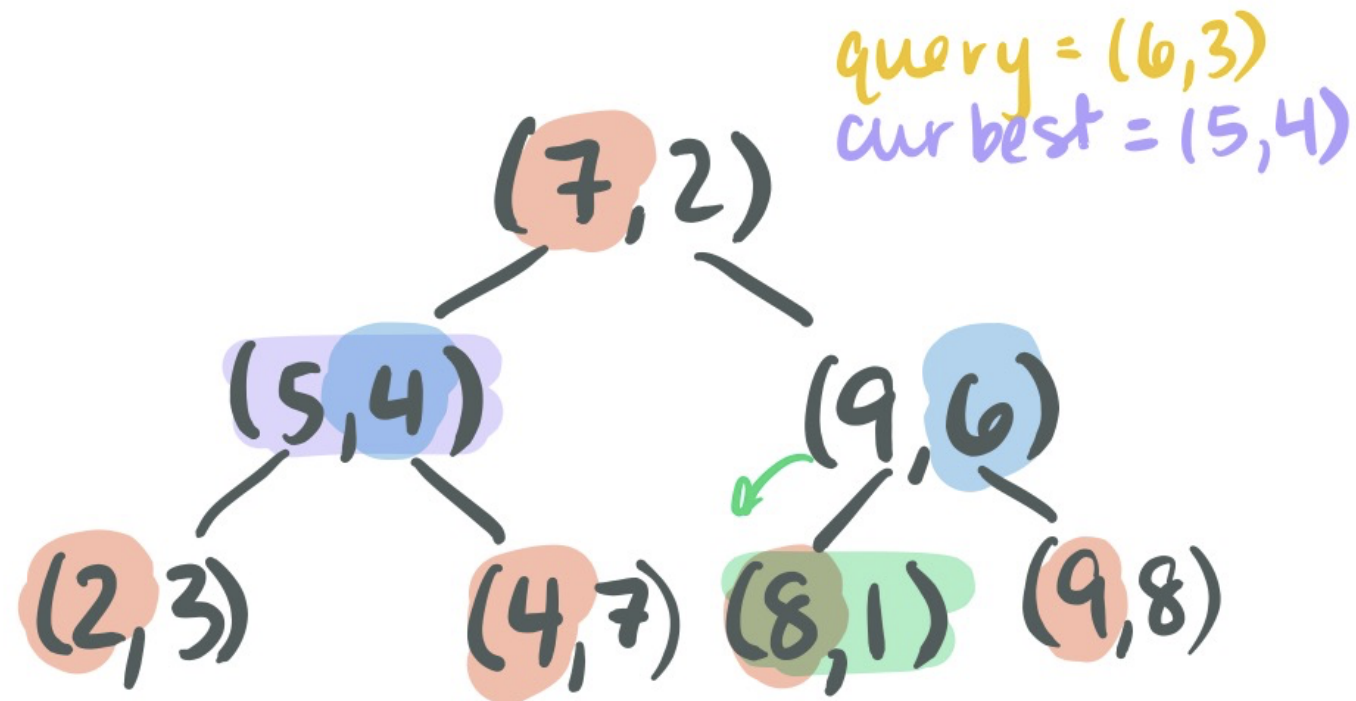


Nearest Neighbor: k-d tree

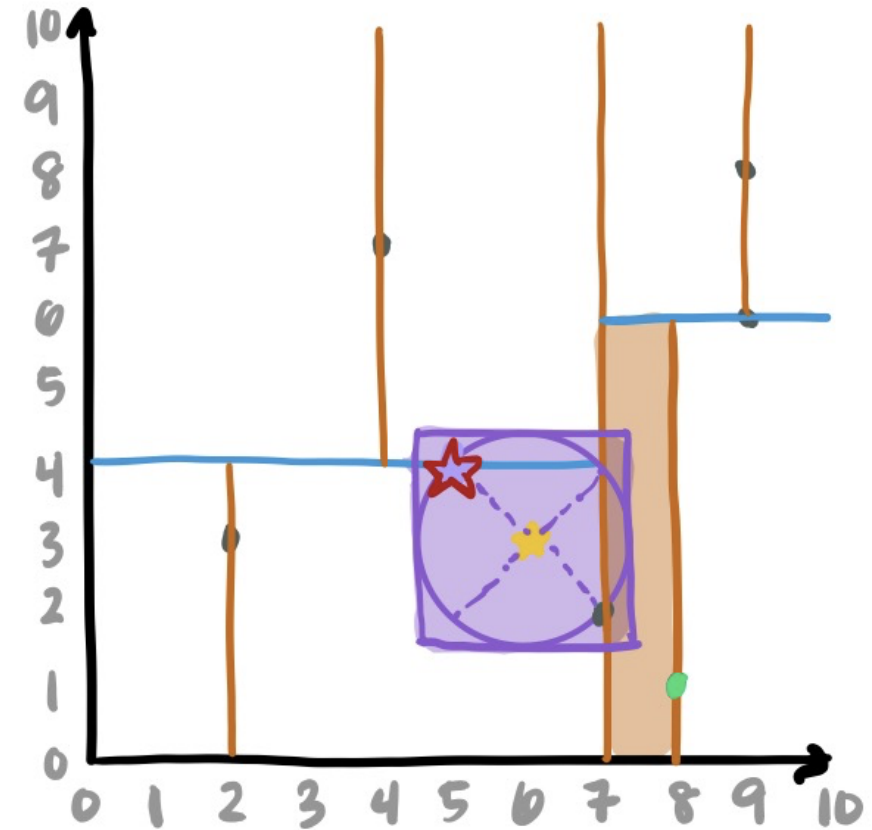
Why do we need to explore this subtree?



Nearest Neighbor: k-d tree



BEST: (5,4)



Tips and Tricks for MP_Mosaics

1. Review, understand, and use **quickselect**

```
1  template <typename RandIter, typename Comparator>
2  void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
3  {
4      /**
5       * @todo Implement this function!
6       */
7
8  }
```

2. Review, understand, and use **lambda functions**

Understanding 'randIter'

An iterator is a container giving access in different ways:

Forward

Bidirectional

Random Access

Implementing quickselect with RandIter

Random Access Iterator lets you:

Swap items using `std::swap()`

```
1  template <typename RandIter, typename Comparator>
2  void BlackBox(RandIter A, RandIter B)
3  {
4
5      std::swap(*A, *B) ;
6
7
8  }
9
```

Hint: Look at pseudo-code for quickselect!

Implementing quickselect with RandIter

Random Access Iterator lets you:

Access container indices using math operations

```
randIter A;
```

```
auto nth = *(A + n);
```

Get distance between two iterators

```
randIter A, B;
```

```
A < B;           // True if A is earlier in container than B
```

```
A - B;           // The distance between A and B
```

Implementing quickselect with RandIter

Random Access Iterator lets you:

Do most things you'd expect an array to be able to do!

The power of the **Interface!**

https://en.cppreference.com/w/cpp/iterator/random_access_iterator

Tips and Tricks for MP_Mosaics

1. Review, understand, and use **quickselect**

```
1 template <typename RandIter, typename Comparator>
2 void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
3 {
4     /**
5      * @todo Implement this function!
6      */
7
8 }
9
```

2. Review, understand, and use **lambda functions**

Functions as arguments

Consider the function from Excel
`COUNTIF(range, criteria)`

A10 ✕ ✓ fx =COUNTIF(A1:A9,"<0")			
	A	B	C
1	1		
2	102		
3	105		
4	4		
5	5		
6	27		
7	41		
8	-7		
9	999		
10	1		
11			

Functions as arguments

Countif.hpp

```
10 template <typename Iter, typename Pred>
11 int Countif(Iter begin, Iter end, Pred pred) {
12     int count = 0;
13     auto cur = begin;
14
15     while(cur != end) {
16         if(pred(*cur))
17             ++count;
18         ++cur;
19     }
20
21     return count;
22 }
```

Lambda Functions in C++

Here are several ways to write a function as an object

main.cpp

```
1 bool isNegative(int num) { return (num < 0); }
2
3 class IsNegative {
4 public:
5     bool operator() (int num) { return (num < 0); }
6 };
7
8 int main() {
9     std::vector<int> numbers = {1, 102, 105, 4, 5, 27, 41, -7, 999};
10
11     auto isnegl = [](int num) { return (num < 0); };
12     auto isnegfp = isNegative;
13     auto isnegfunctor = IsNegative();
14
15     cout << "There are " << Countif(numbers.begin(), numbers.end(), _____)
16         << " negative numbers" << std::endl;
17 }
```

Lambda Functions in C++

[Capture](Arg List){ Function Body }

Lambda Functions in C++

[Capture](Arg List){ Function Body }

Capture: Takes the value of object based on when the lambda was defined, NOT the current value of the object!

Arg List: Standard way of inputting into a function

Function Body: Code can use both capture vars and arg vars

Lambda Functions in C++



main.cpp

```
29  int big;  
30  std::cout << "How big is big? ";  
31  std::cin >> big;  
32  
33  auto isbig = [big](int num) { return (num >= big); };  
34  
35  
36  
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)  
38  << " big numbers" << std::endl;  
}
```

Lambda Functions in C++



main.cpp

```
29  int big;  
30  std::cout << "How big is big? ";  
31  std::cin >> big;  
32  
33  auto isbig = [big](int num) { return (num >= big); };  
34  
35  
36  
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)  
38  << " big numbers" << std::endl;  
}
```

Useful for mp_mosaics!

KD-Tree will split points in one dimension

When comparing, we need to remember what dimension we are in!

Tips and Tricks for MP_Mosaics

Final tips:

The mp_mosaic writeup is long. **READ IT**

The suggestions in the writeup should be followed carefully