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

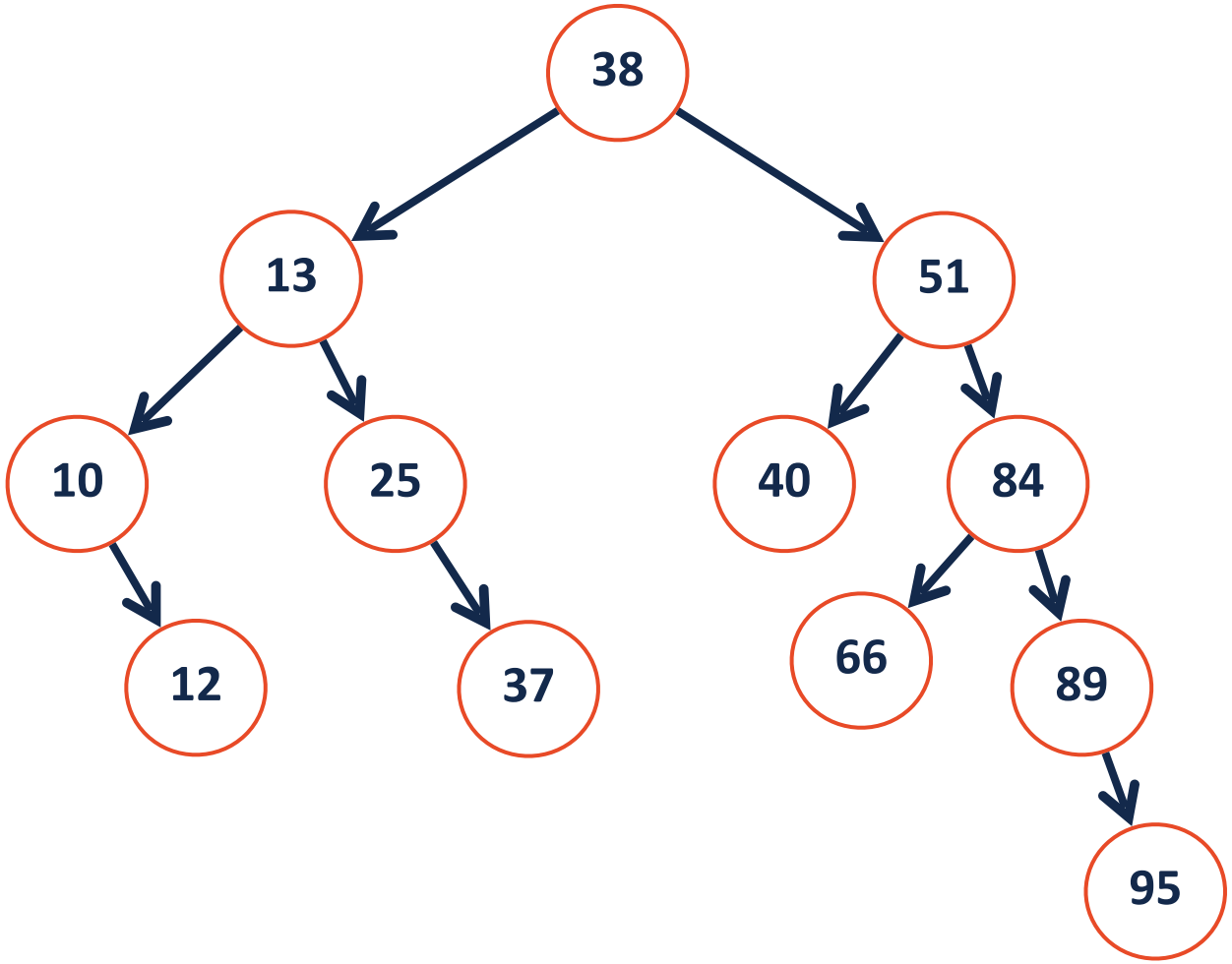
KD tree : Motivation and Creation

KD tree : Interval Search and Nearest Neighbor

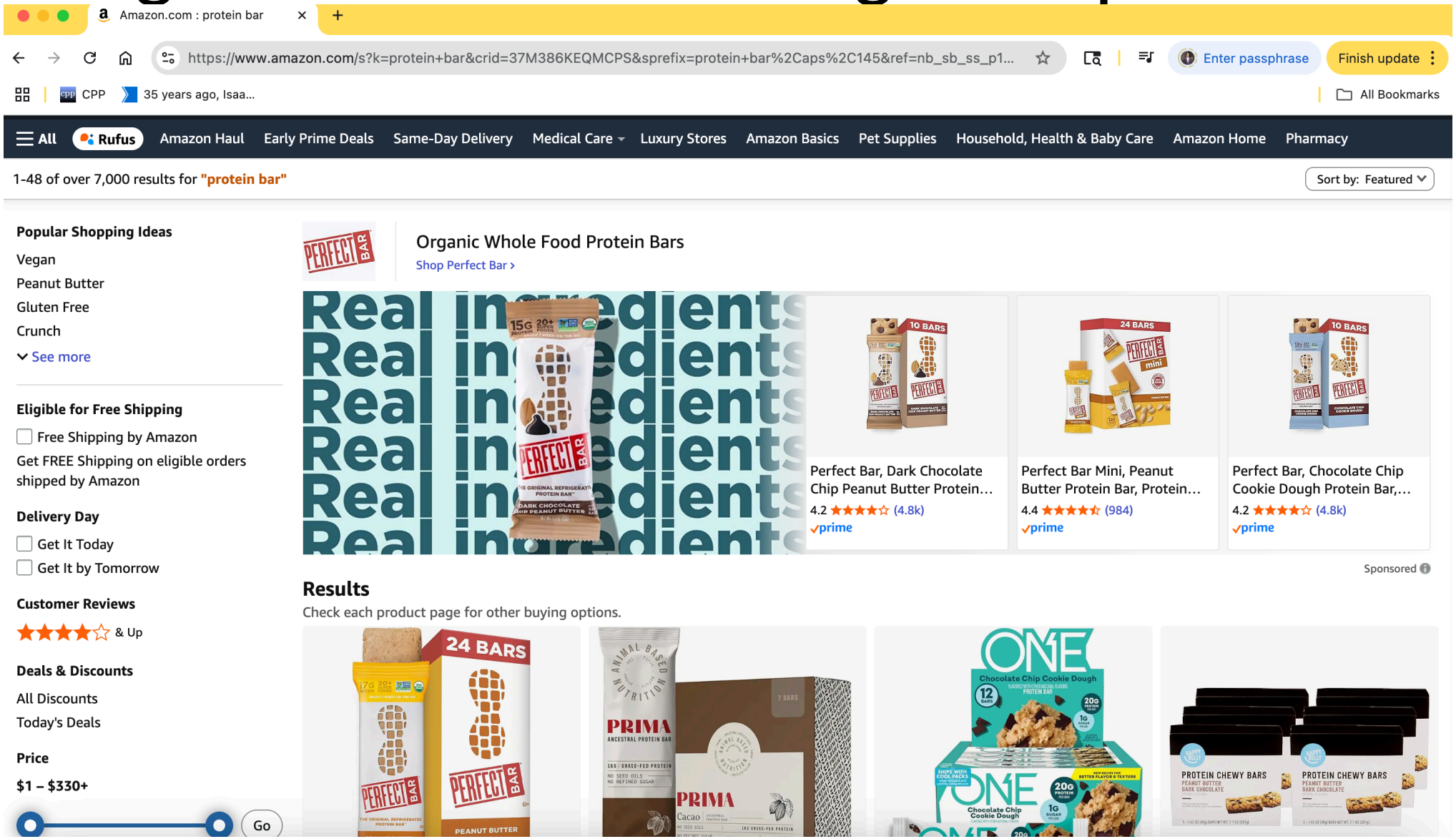
Go over C++ concepts for mp_mosaics

BST Remove (IOS)

`remove(51)`



Range Search : Motivating example



Want Protein bars that cost between 10$ and 20$

Range-based Searches

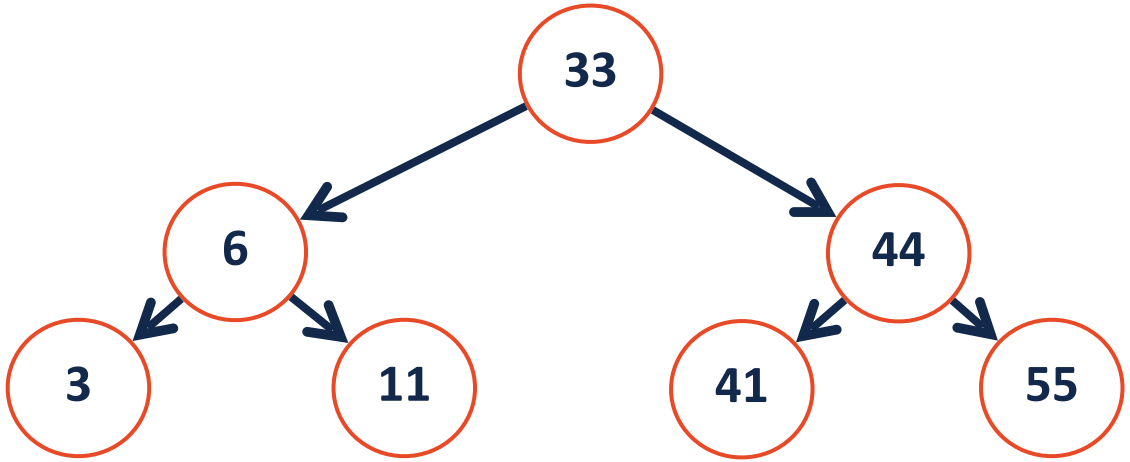
Consider a collection of points on a 1D line: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$

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: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$

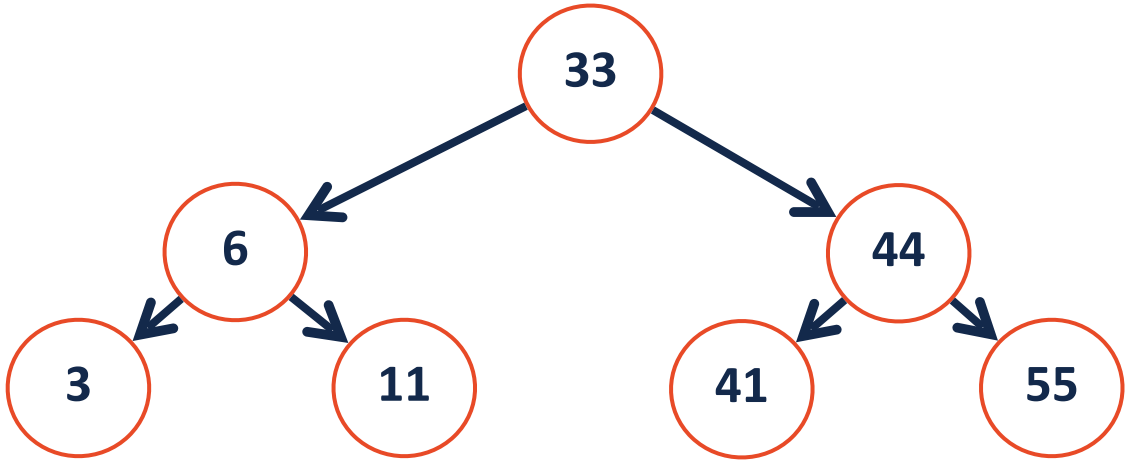
If I want to find all values between [A, B], how could I implement this?



Range-based Searches

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

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

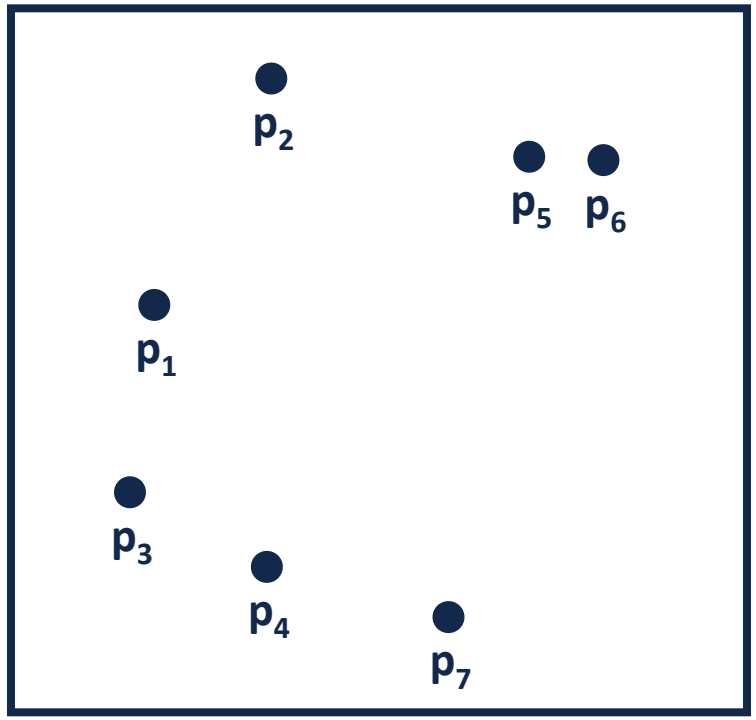



```
1
2 for(auto it = myMap.lower_bound(A); it != myMap.upper_bound(B); ++it){
3
4 // Do Stuff
5 }
```

Range-based Searches

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

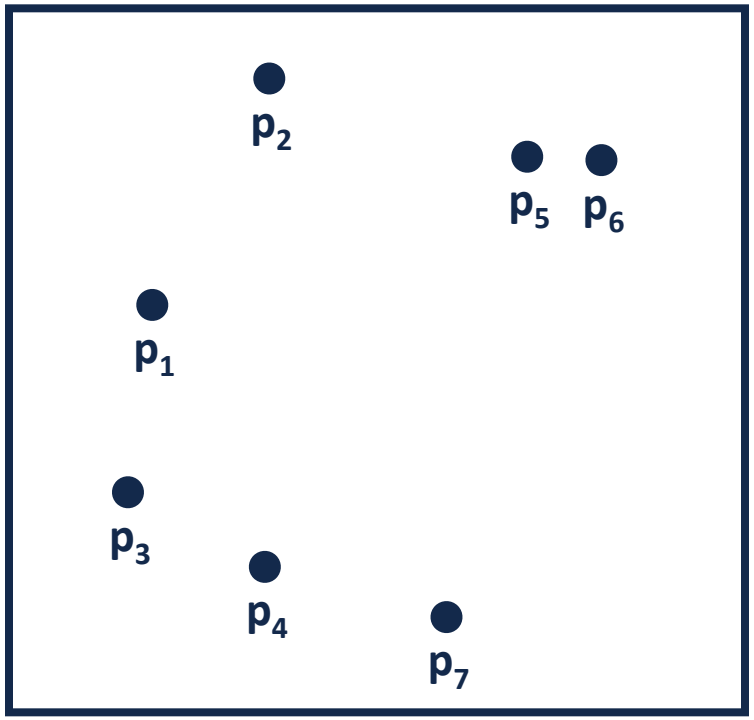
What points in rectangle [$\mathbf{(x_1, y_1)}$, $\mathbf{(x_2, y_2)}$]?



Range-based Searches

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

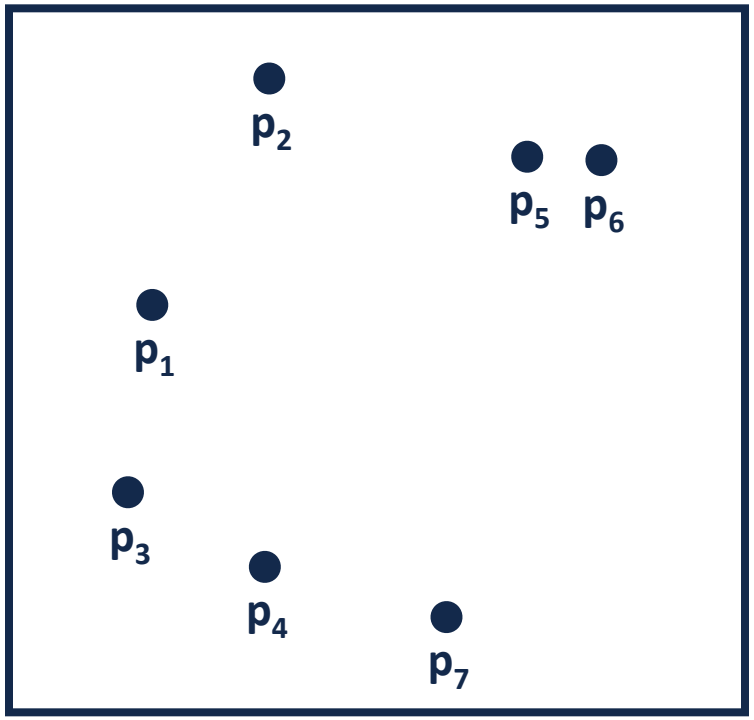
What is nearest point to $\mathbf{(x_1, y_1)}$?



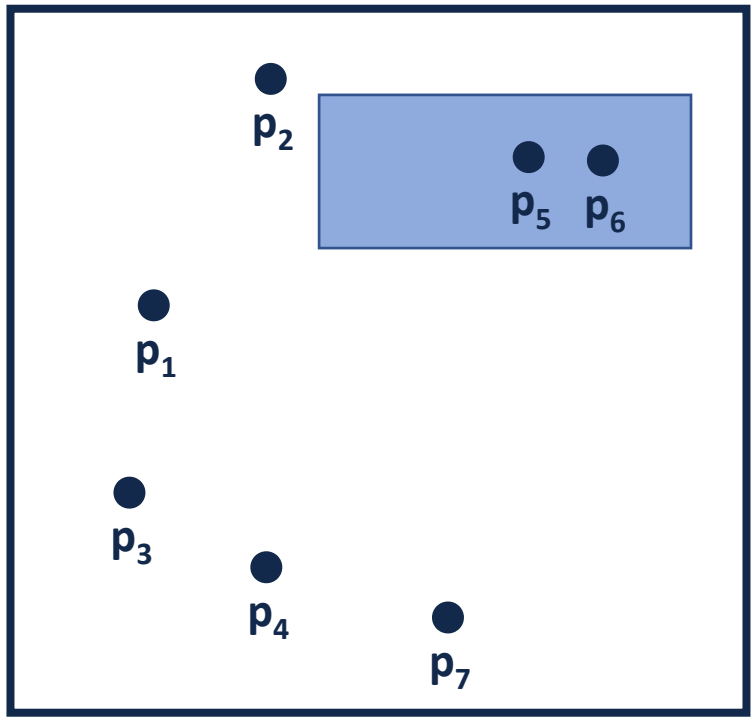
Range-based Searches

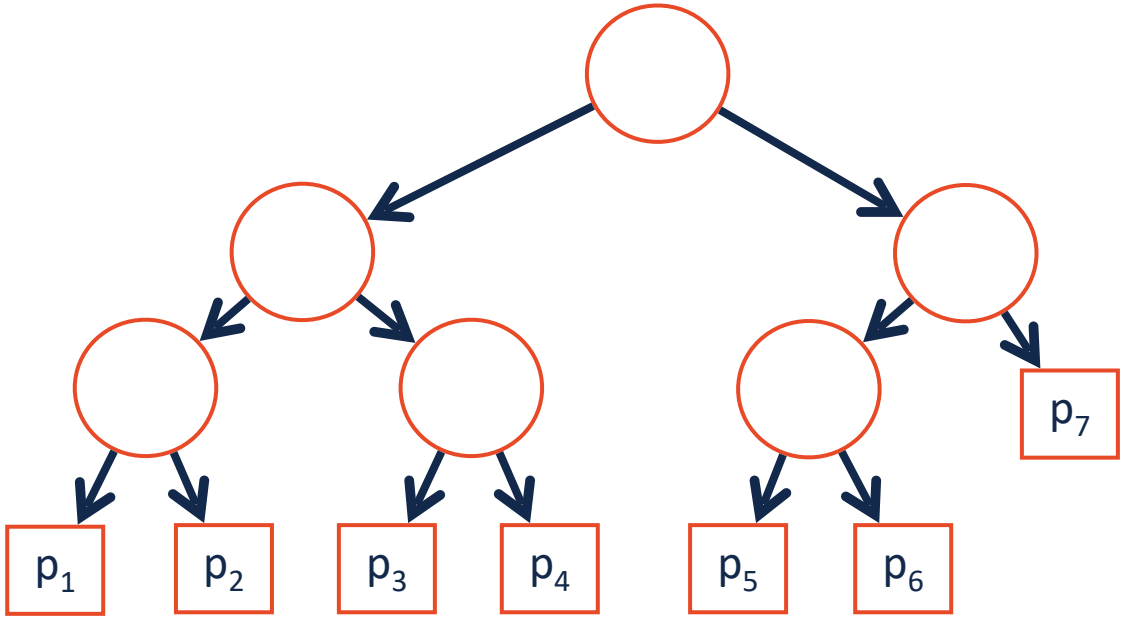
Consider points in 2D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$

Tree Construction:

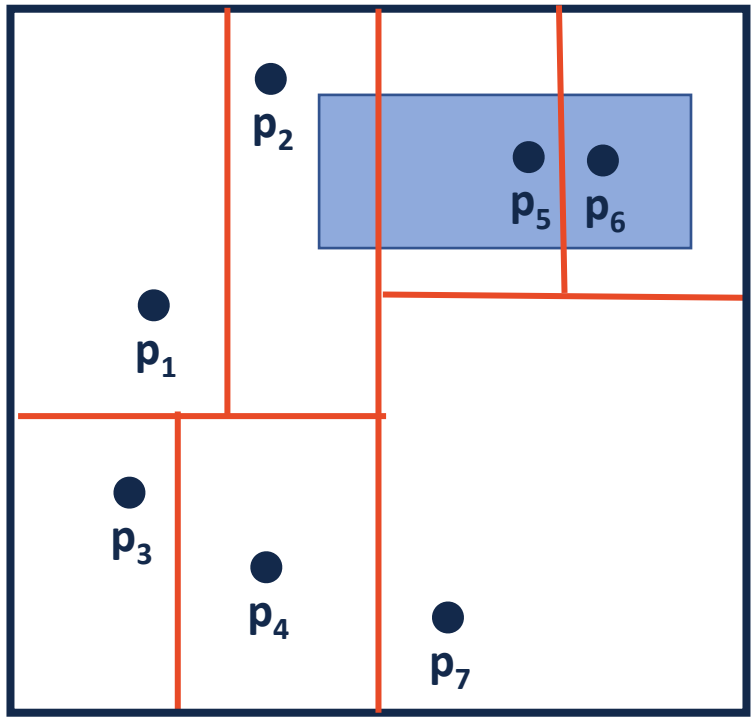


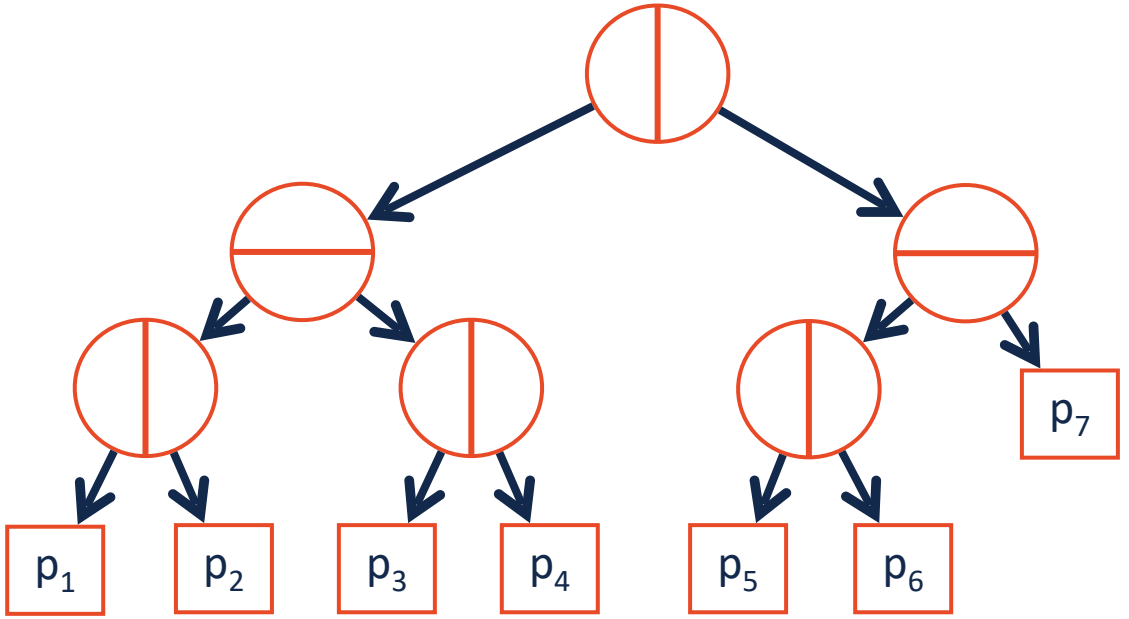
Range-based Searches





Range-based Searches

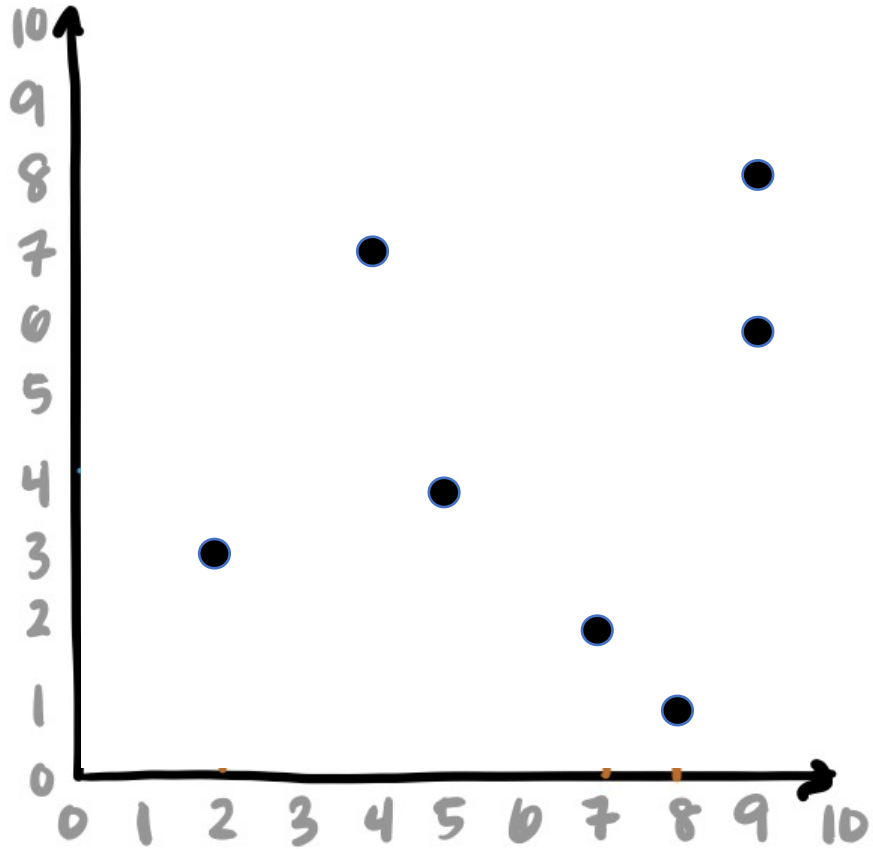




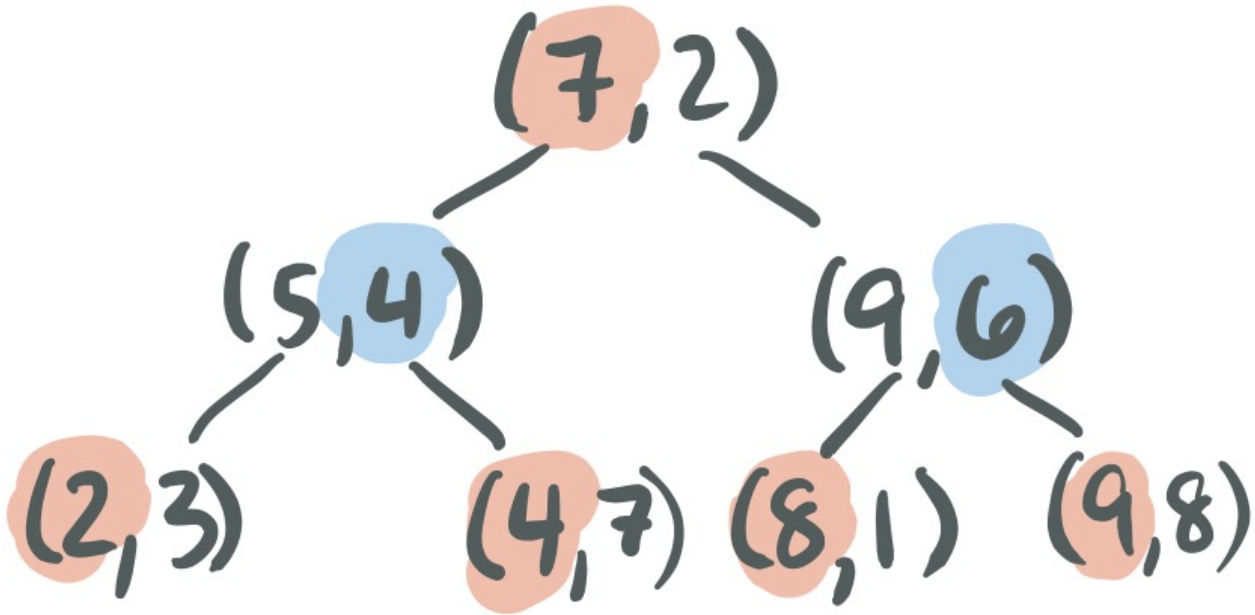
Nearest Neighbor: k-d tree

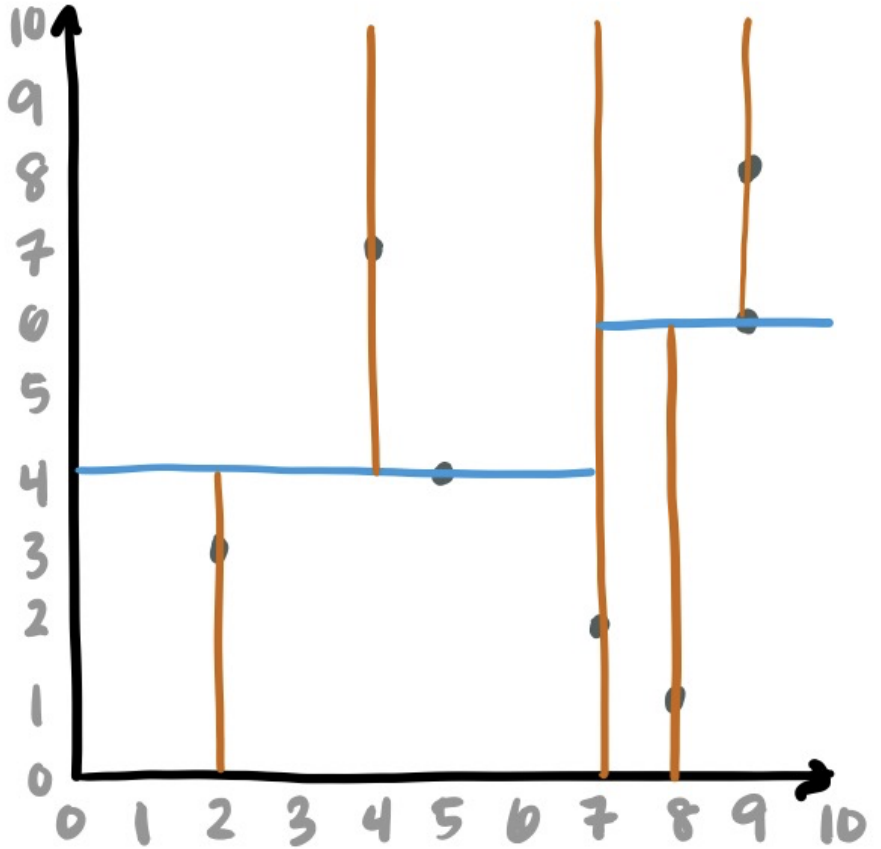
A **k-d tree** is similar but splits on points:

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



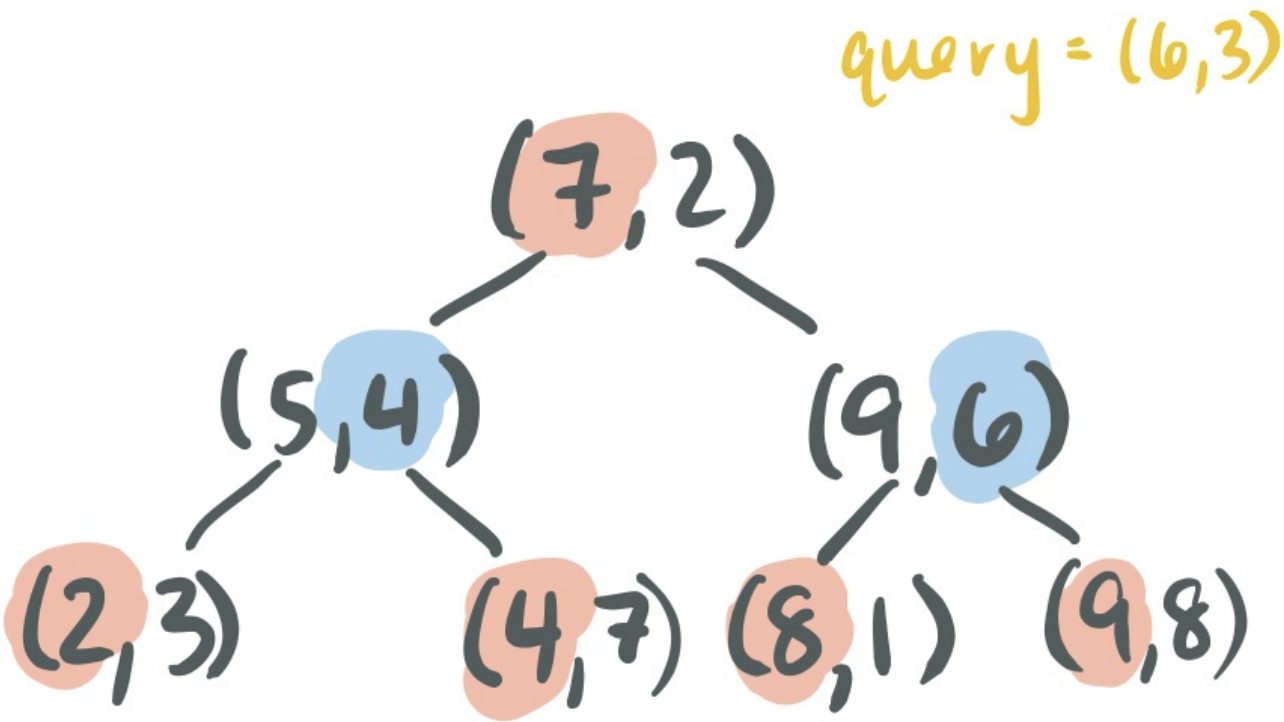
Nearest Neighbor: k-d tree

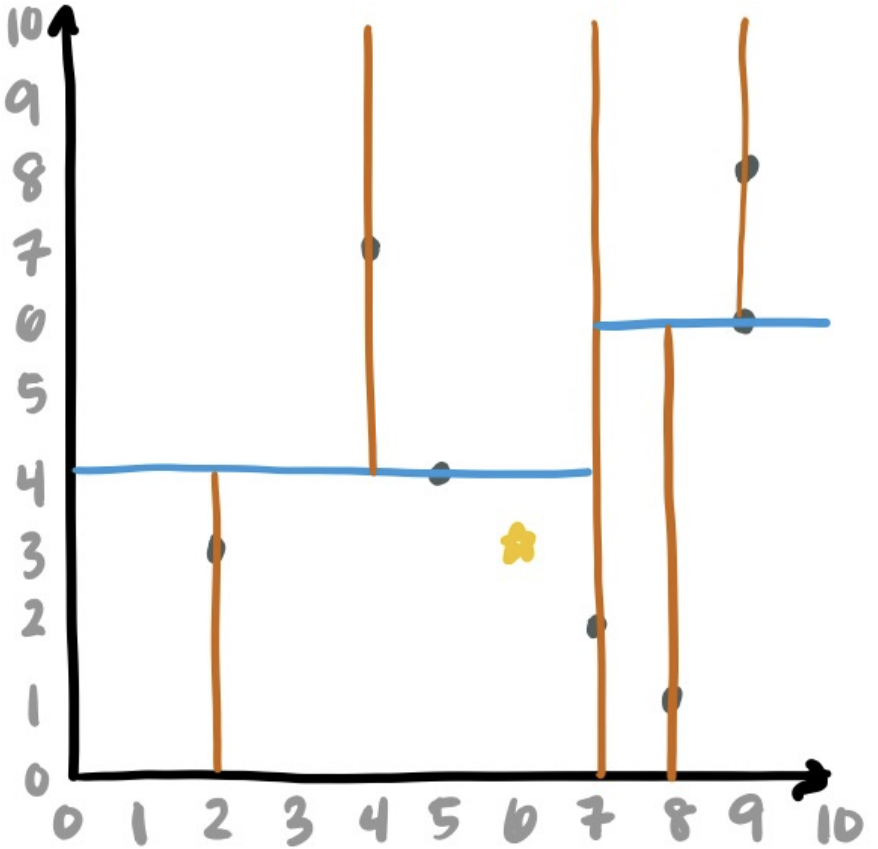




Nearest Neighbor: k-d tree

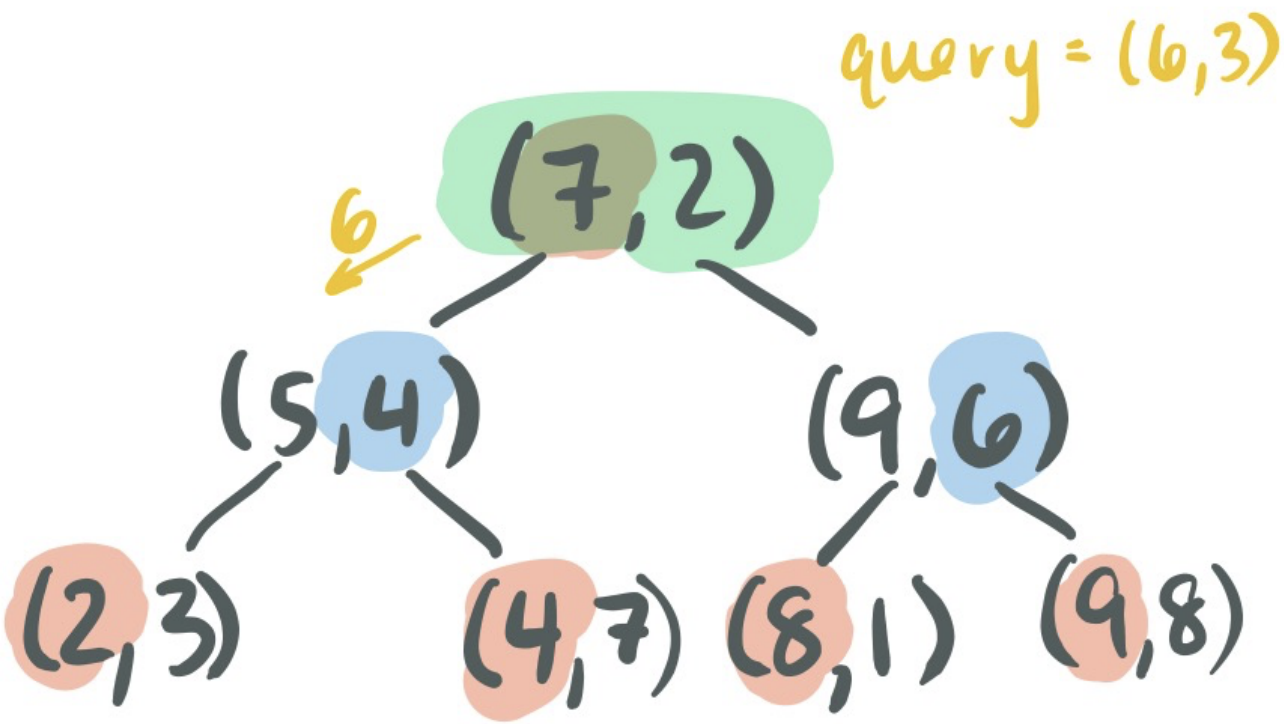
When querying a k-d tree, it acts like a BST* at first…

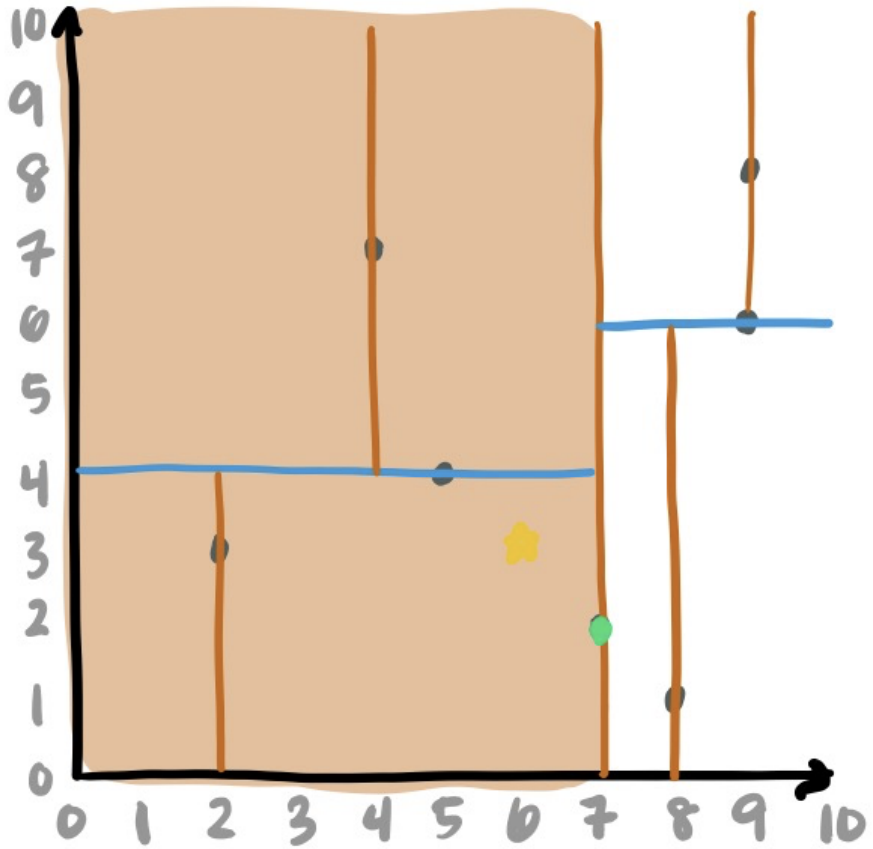




Nearest Neighbor: k-d tree

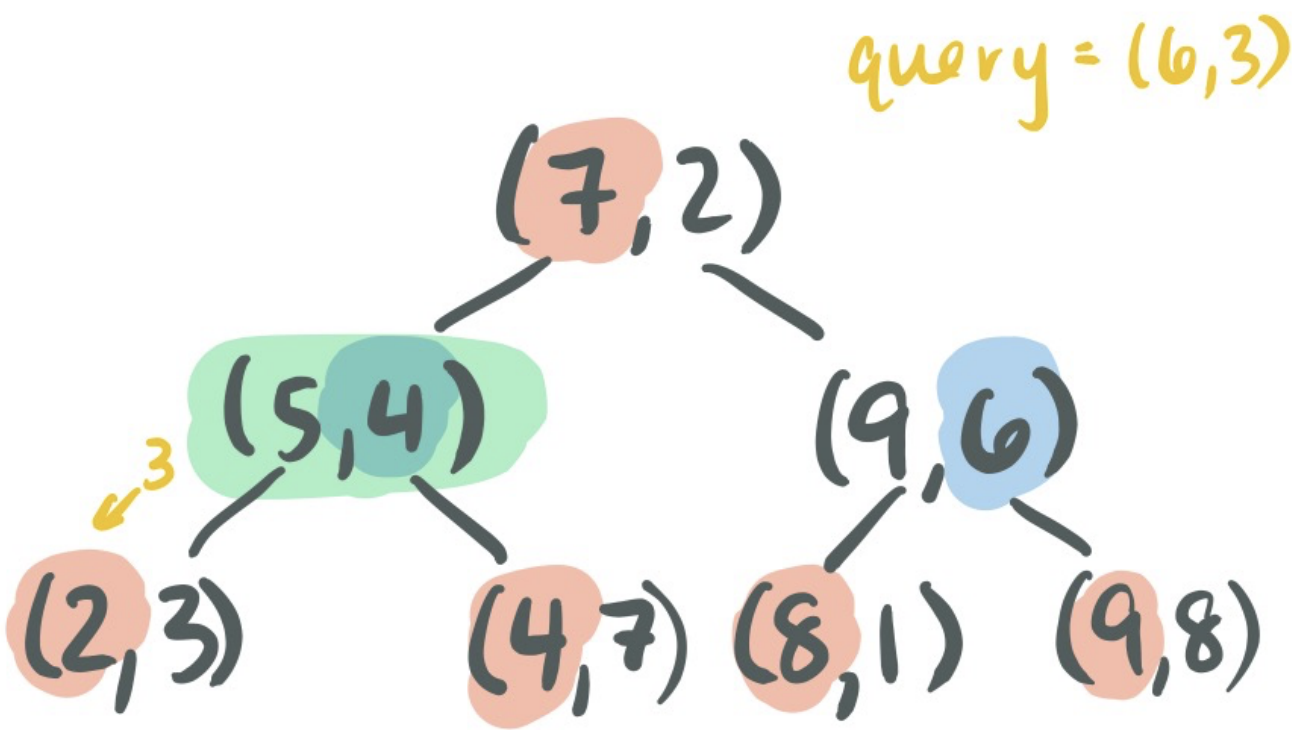
When querying a k-d tree, it acts like a BST* at first…

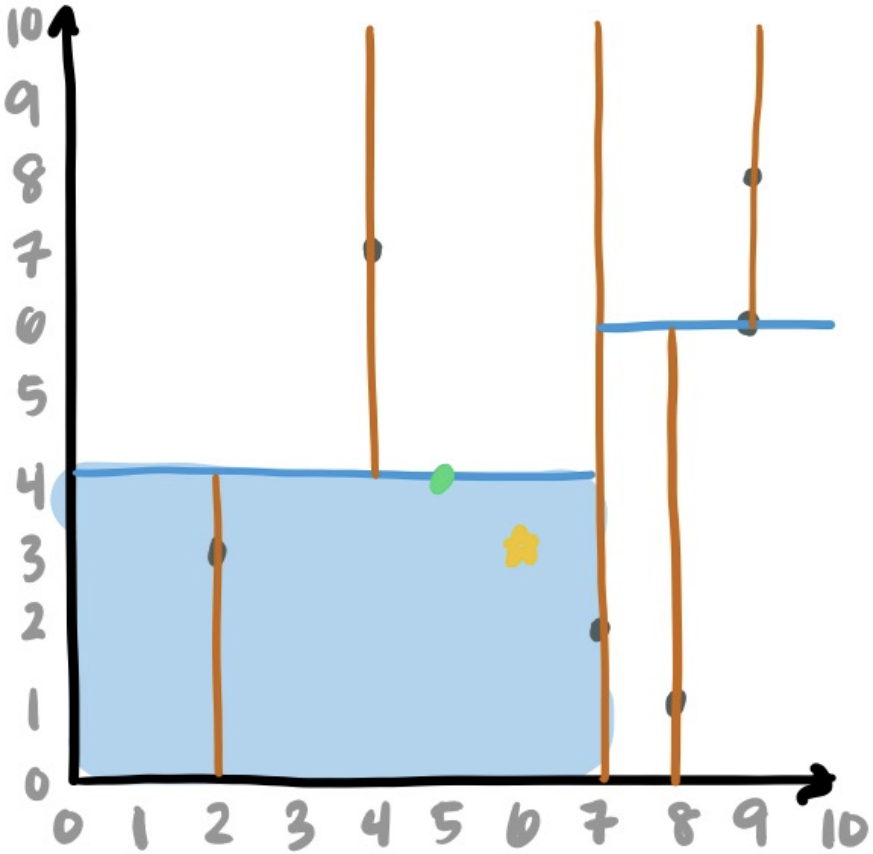




Nearest Neighbor: k-d tree

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

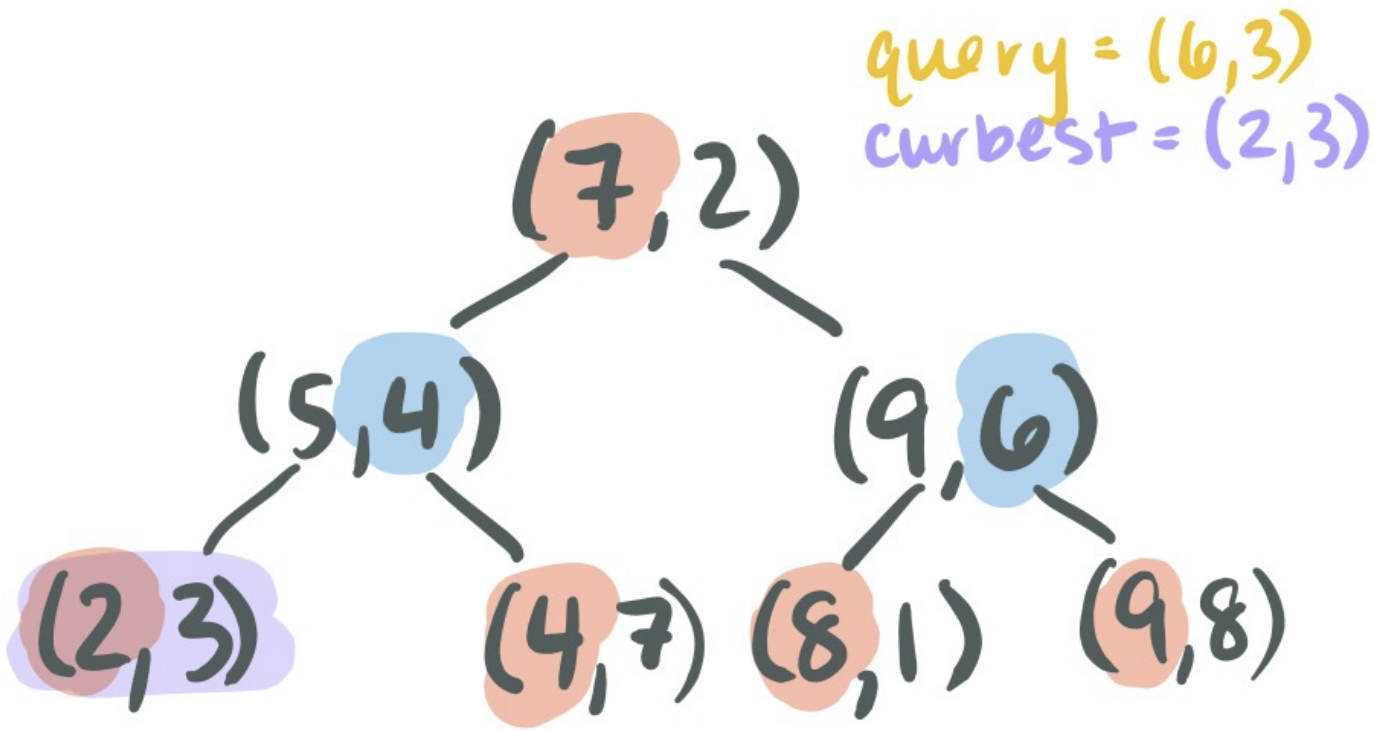


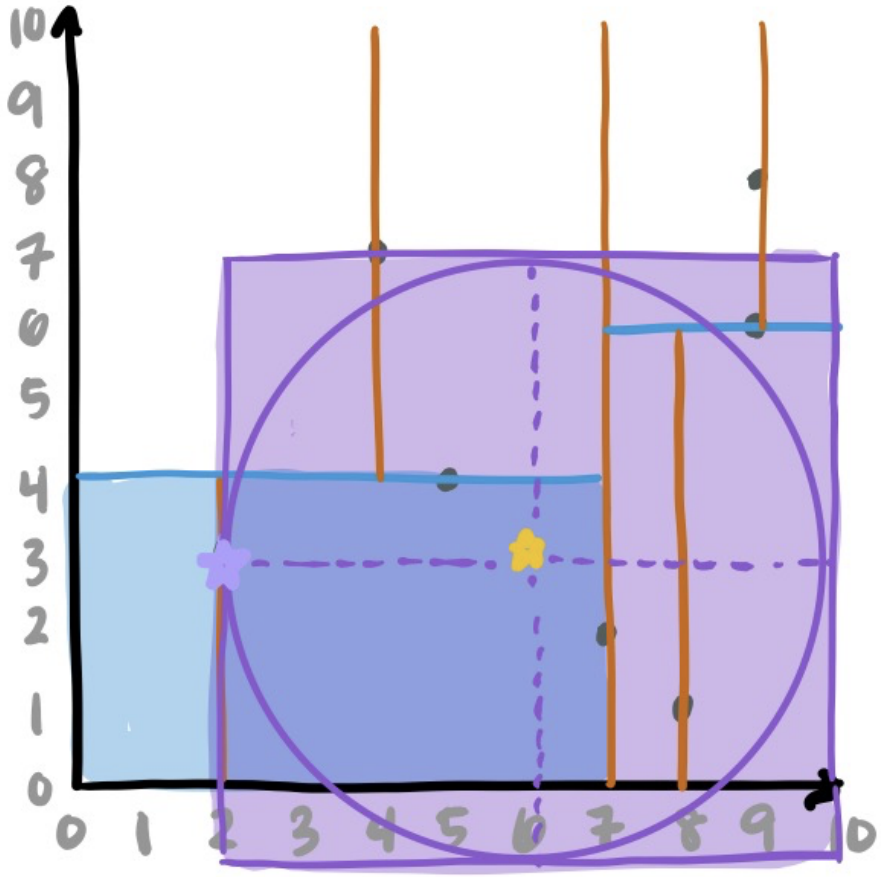


Nearest Neighbor: k-d tree

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

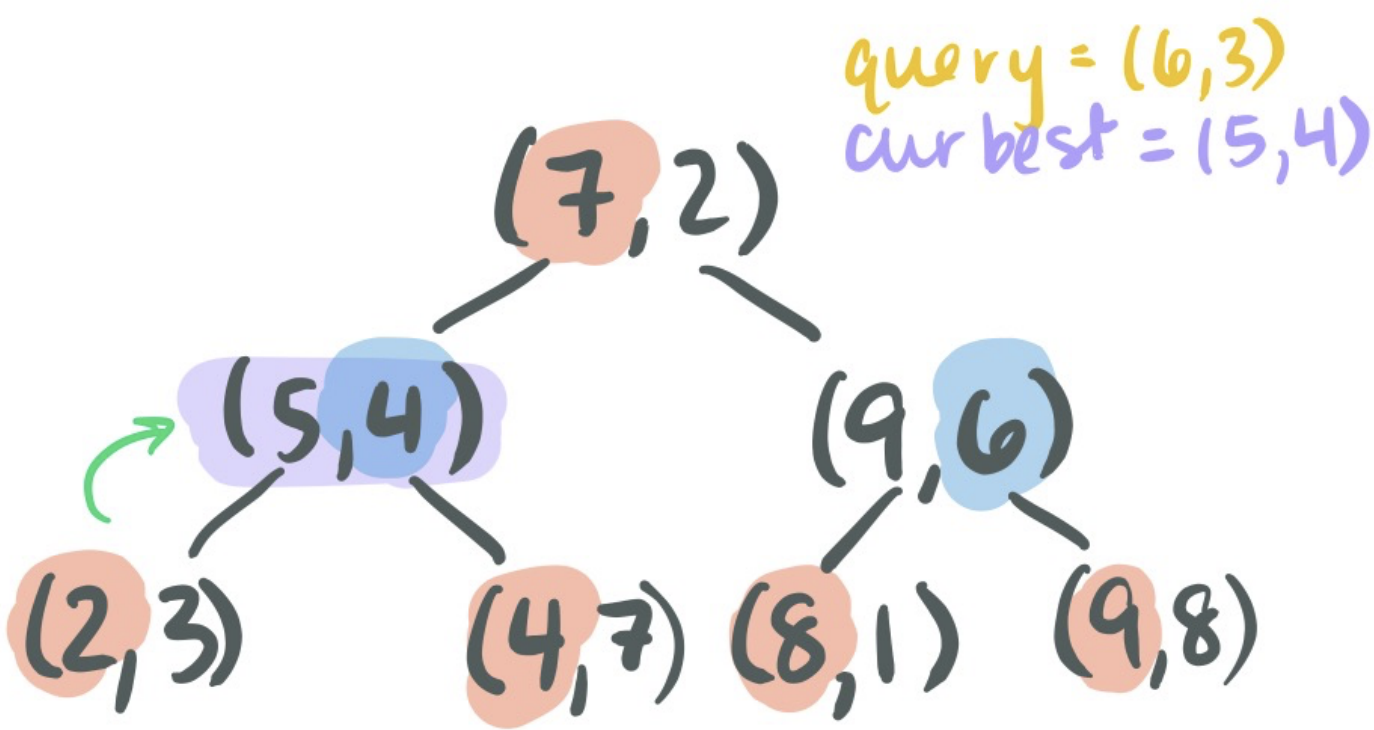
… But if we dont find exact match, have to find nearest neighbor

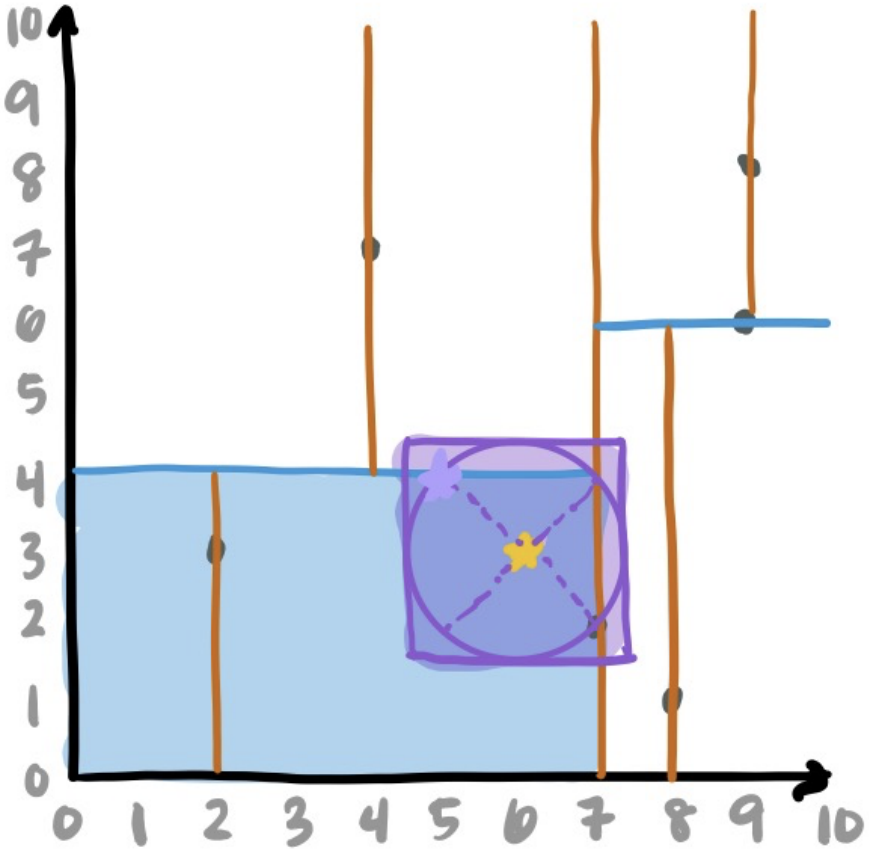




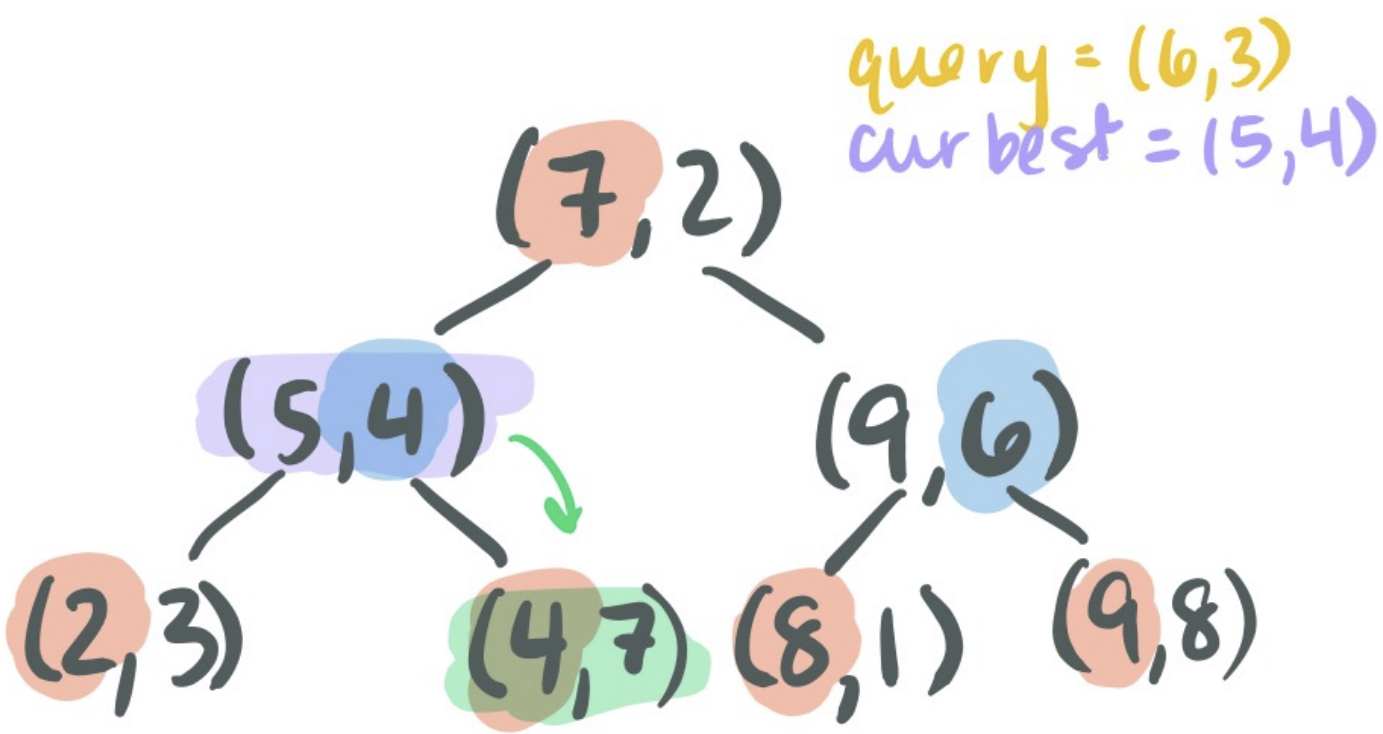
Nearest Neighbor: k-d tree

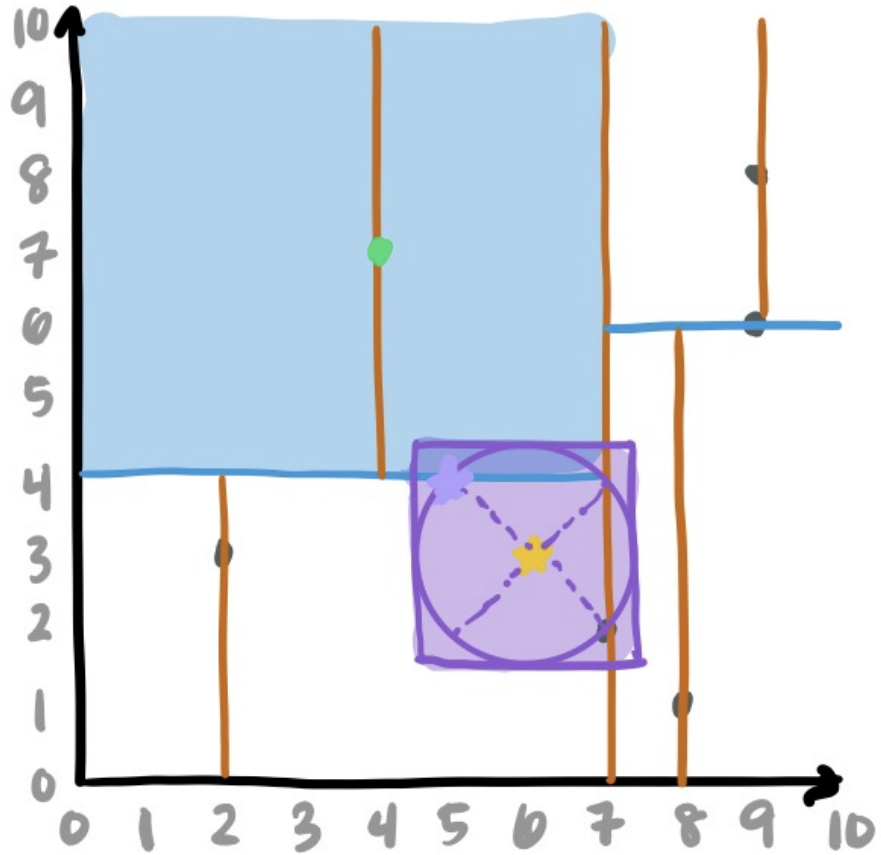
Backtracking: start recursing backwards -- store "best" possibility as you trace back





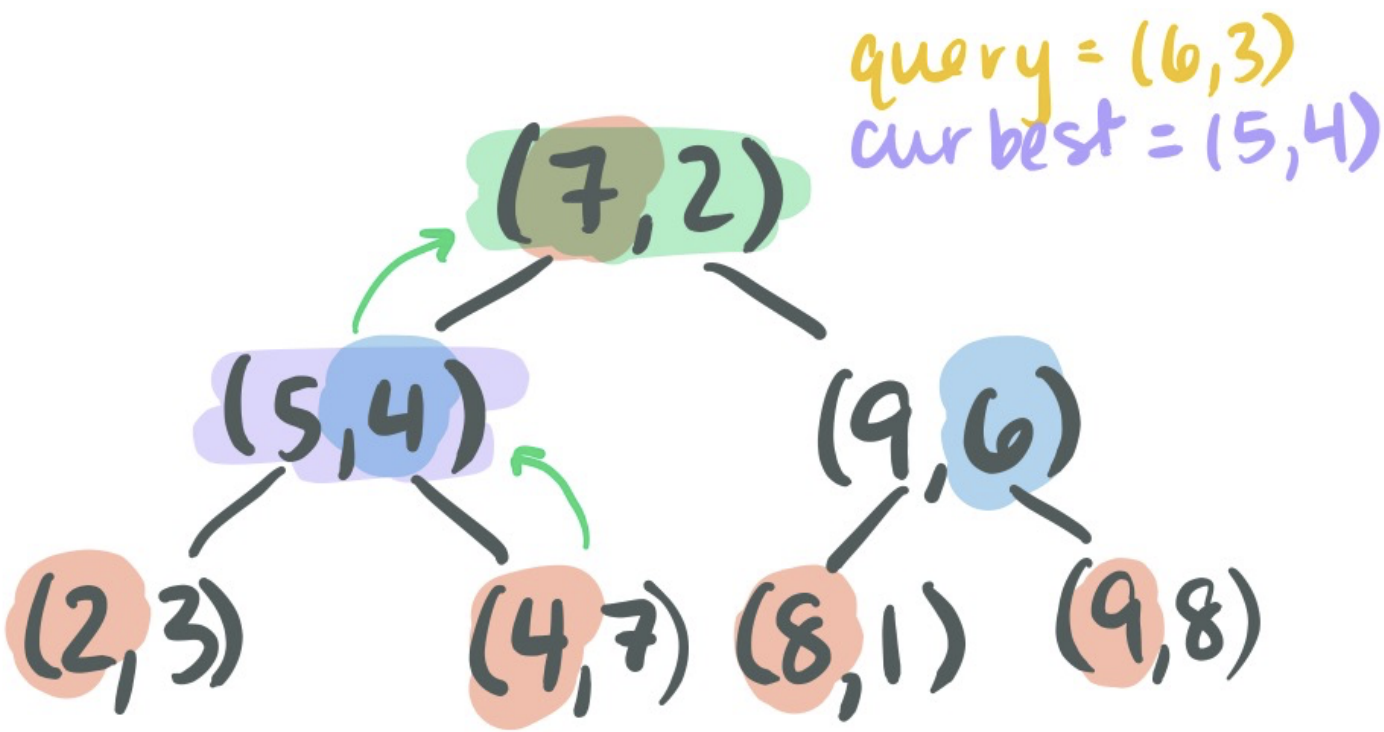
Nearest Neighbor: k-d tree

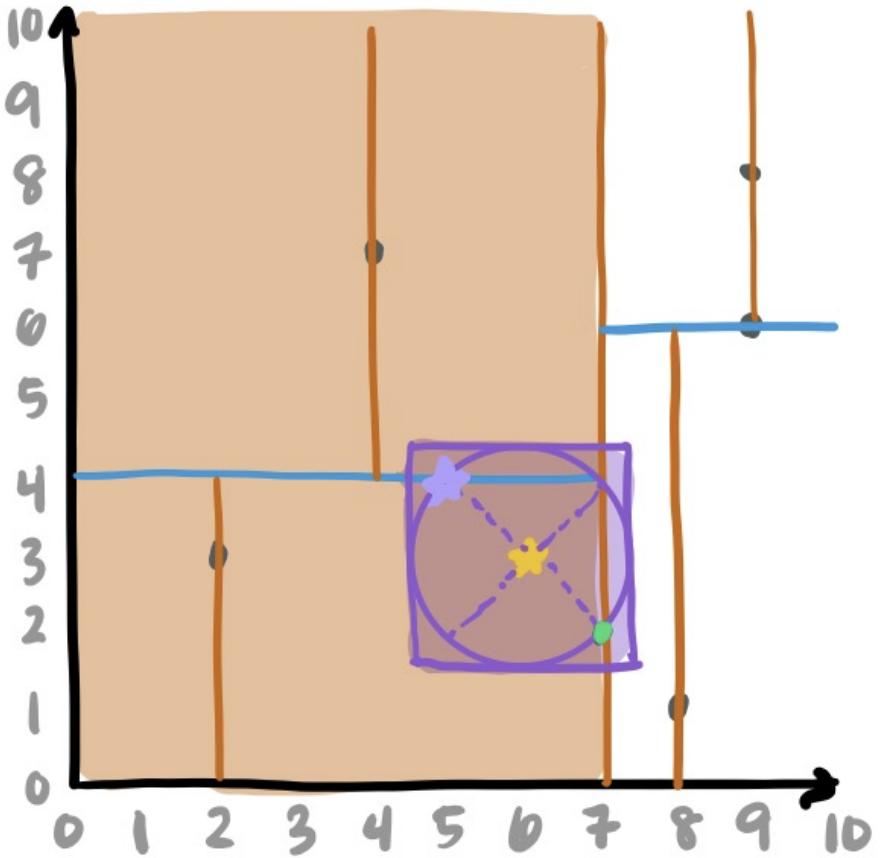




Nearest Neighbor: k-d tree

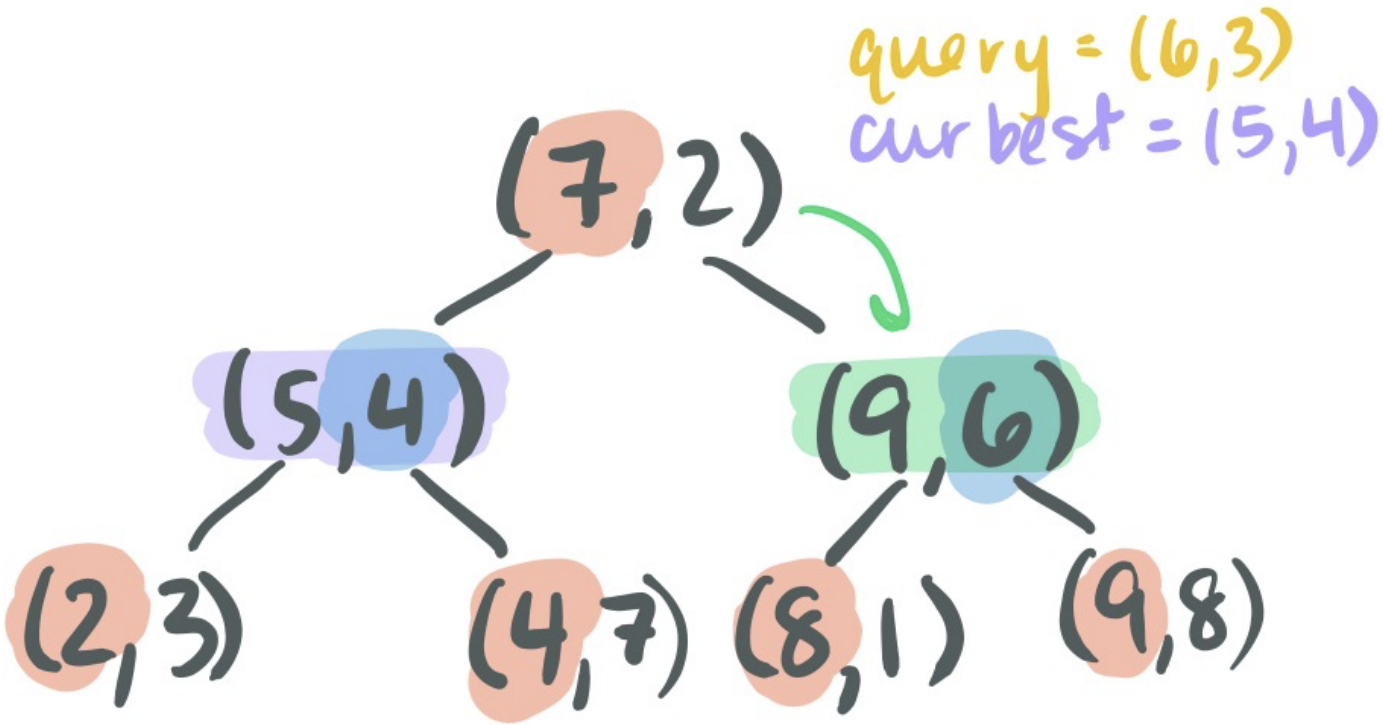
On ties, use smallerDimVal to determine which point remains curBest

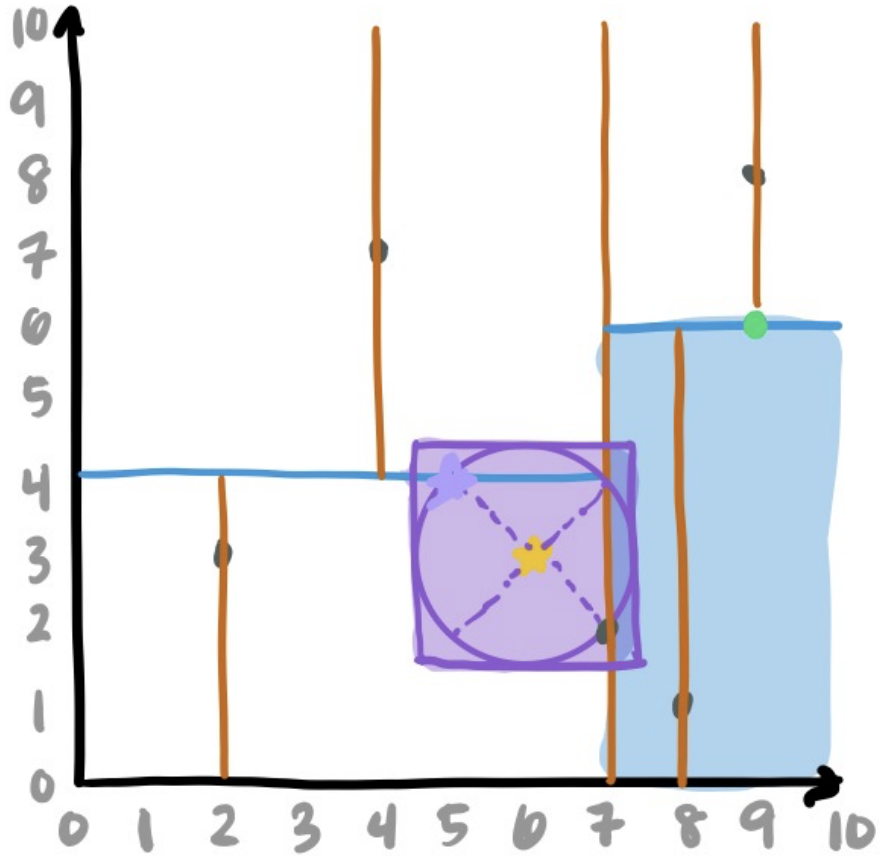




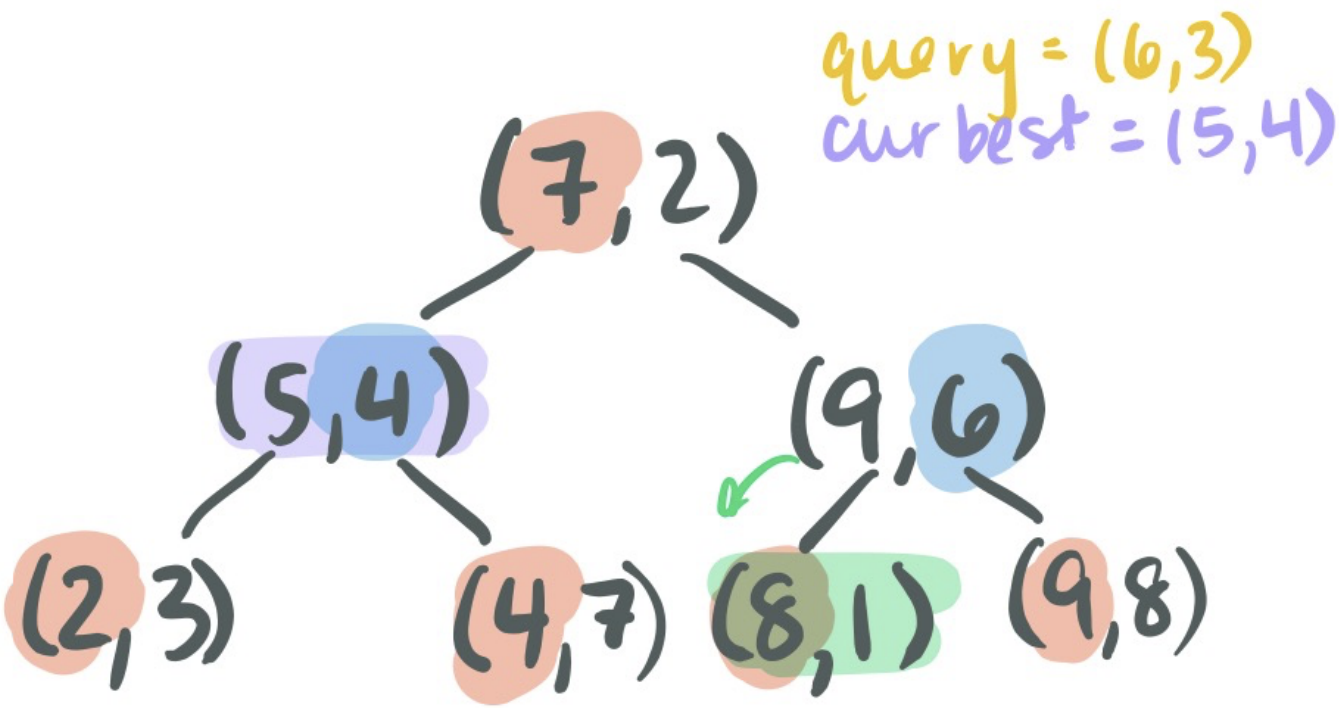
Nearest Neighbor: k-d tree

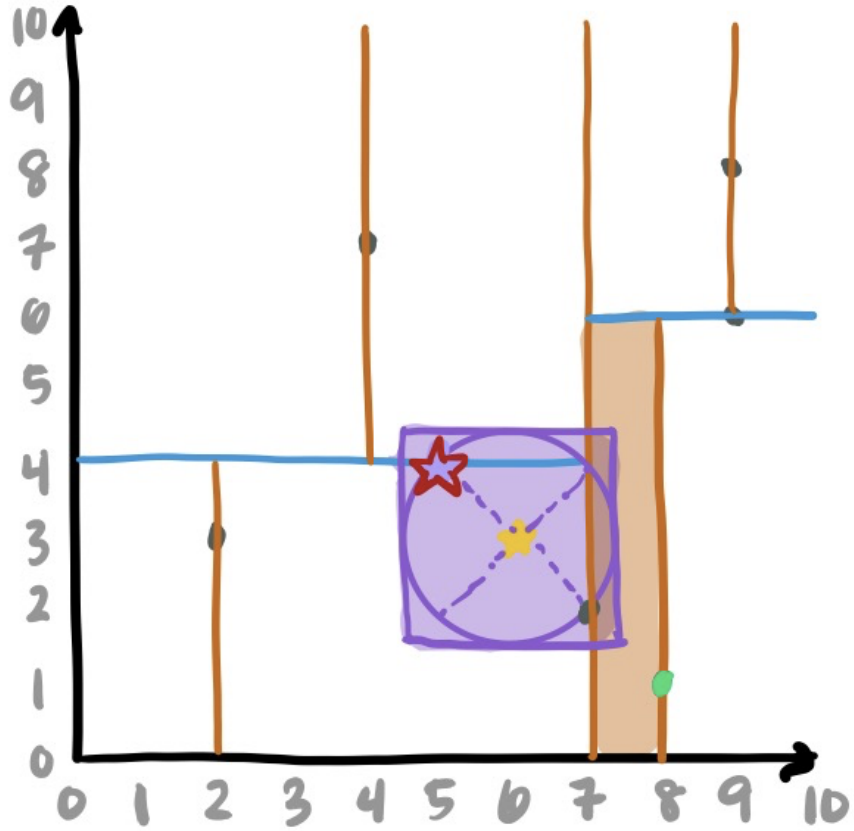
Why do we need to explore this subtree?





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    }
9
```

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

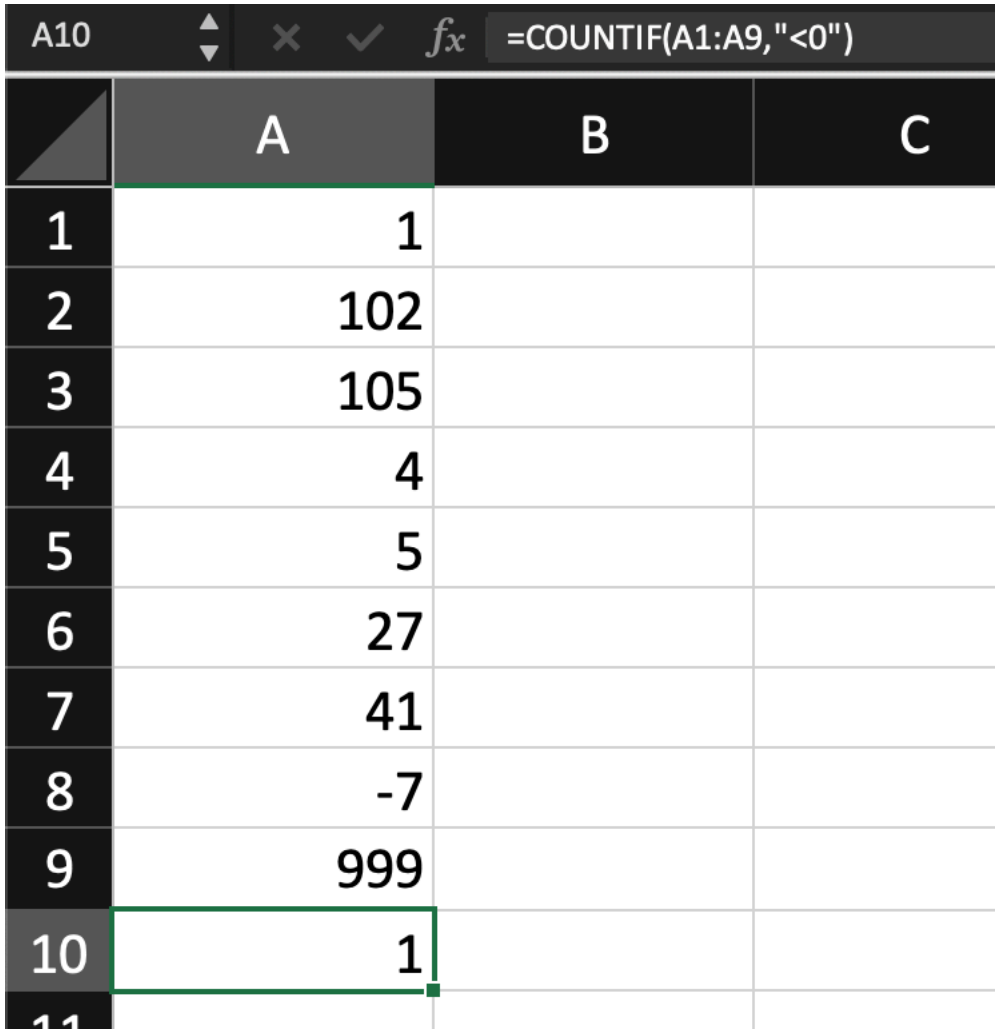
1. Review, understand, and use **quickselect**

```
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*)



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 inputing 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