Data Structures

KD Tree

CS 225
Brad Solomon

October 2, 2024

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!

Studying what aspects of class are most / least helpful

Learning Objectives

Explore the need and use of range search

Introduce the KD Tree

Go over C++ concepts for mp_mosaics

Summary of Balanced BST

AVL Trees

- Max height: 1.44 * lg(n)
- Rotations:

Summary of Balanced BST

AVL Trees

- Max height: 1.44 * lg(n)
- Rotations:

  Zero rotations on find

  One rotation on insert

  O(**h**) == O(**lg(n)**) rotations on remove

Red-Black Trees

- Max height: 2 * lg(n)
- Constant number of rotations on insert (max 2), remove (max 3).

Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```
std::map<K, V> map;

V & std::map<K, V>::operator[]( const K & )

std::map<K, V>::erase( const K & )
```

Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```
iterator std::map<K, V>::lower_bound( const K & );
```

```
iterator std::map<K, V>::upper_bound( const K & );
```

Summary of Balanced BST

Pros:

O(log N) for insert, find, remove

Optimal range queries in 1D

Cons:

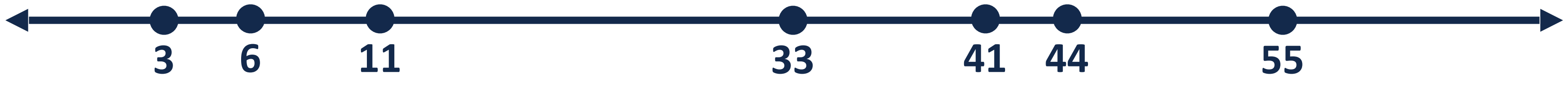
O(log N) isn’t that great

Large in-memory requirement

Range-based Searches

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

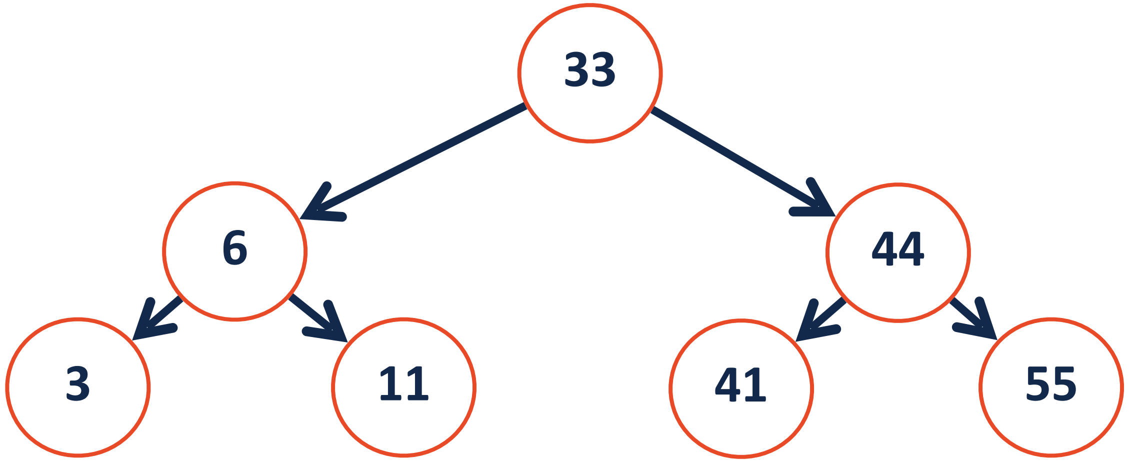
If I want to find all values between [A, B], how could I implement this?



Range-based Searches

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

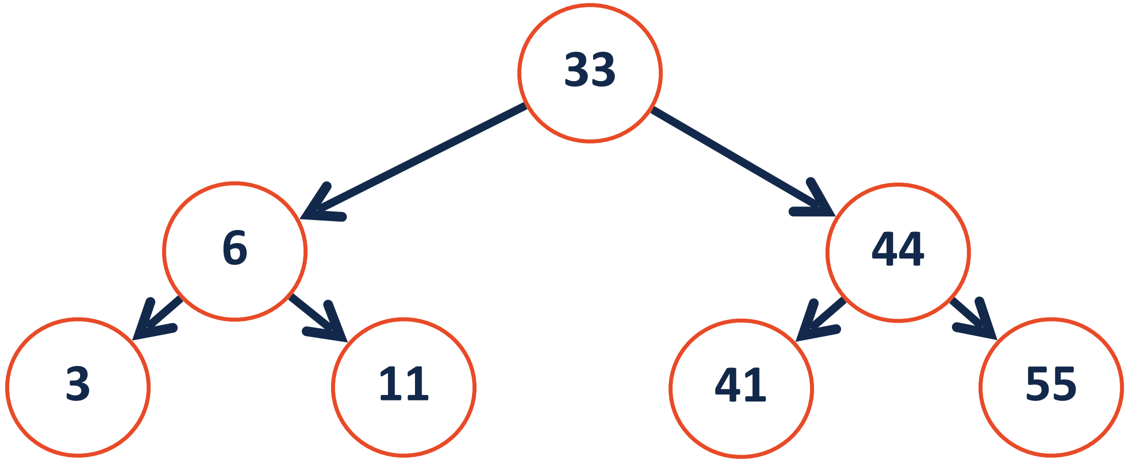
If I want to find all values between [A, B], how could I implement this?



Range-based Searches

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

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

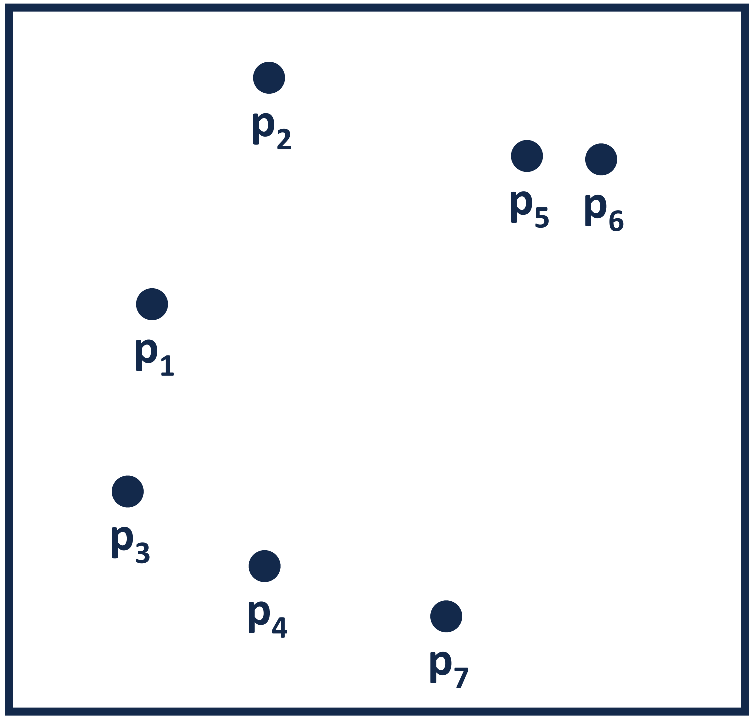


```

for(auto it = myMap.lower_bound(A); it != myMap.upper_bound(B); ++it){

// Do Stuff
}
```

Range-based Searches

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

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

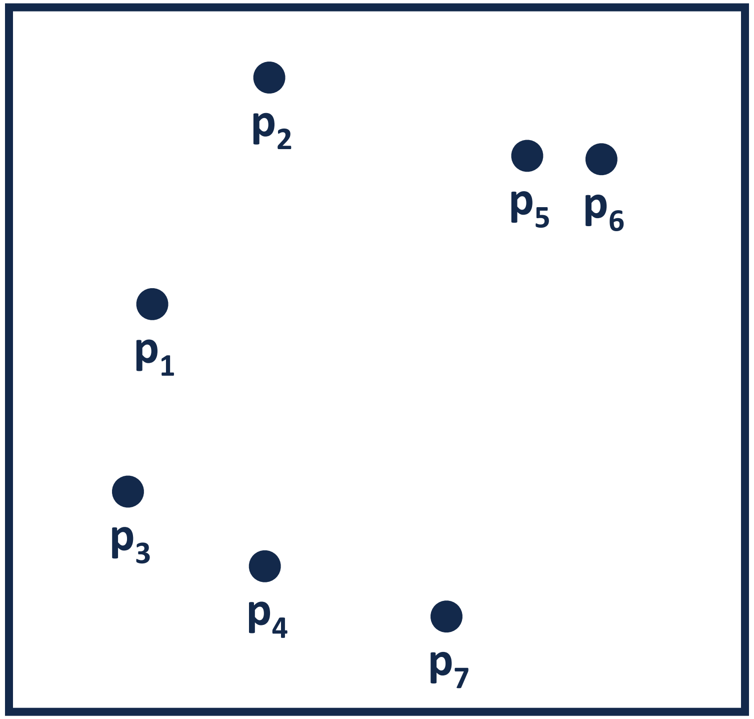


What is nearest point to $\mathbf{(x_1, y_1)}$?

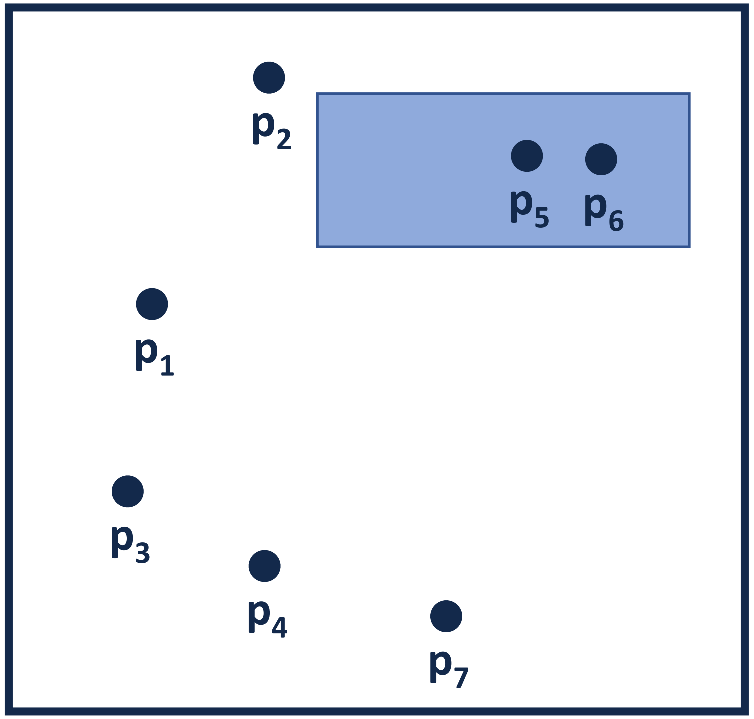
Range-based Searches

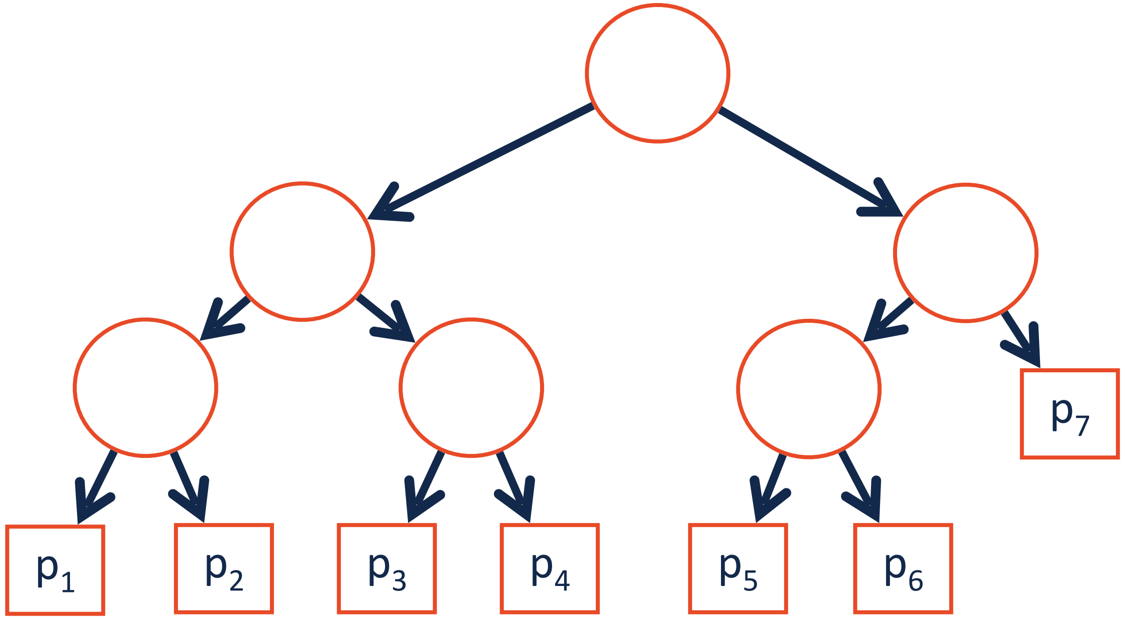
Consider points in 2D: $\mathbf{p} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$

Tree Construction:

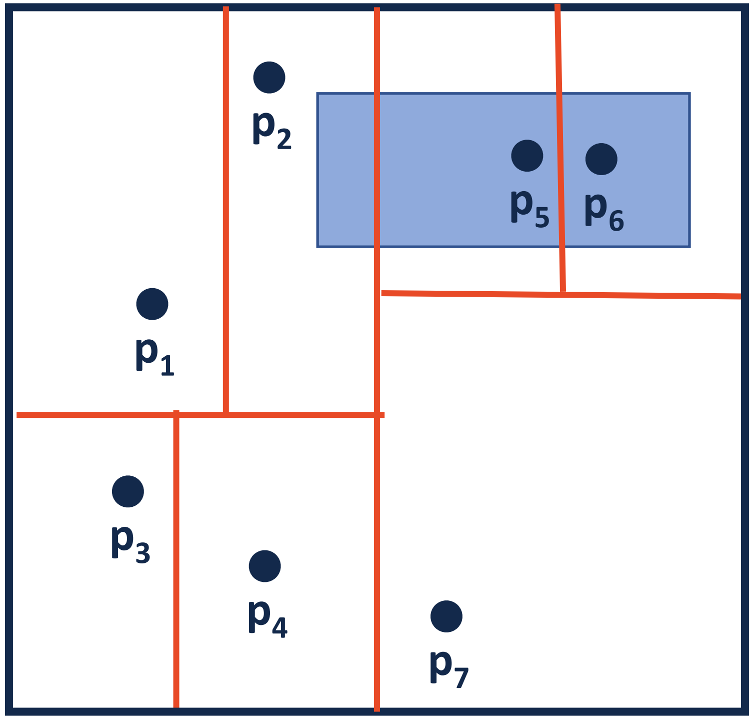


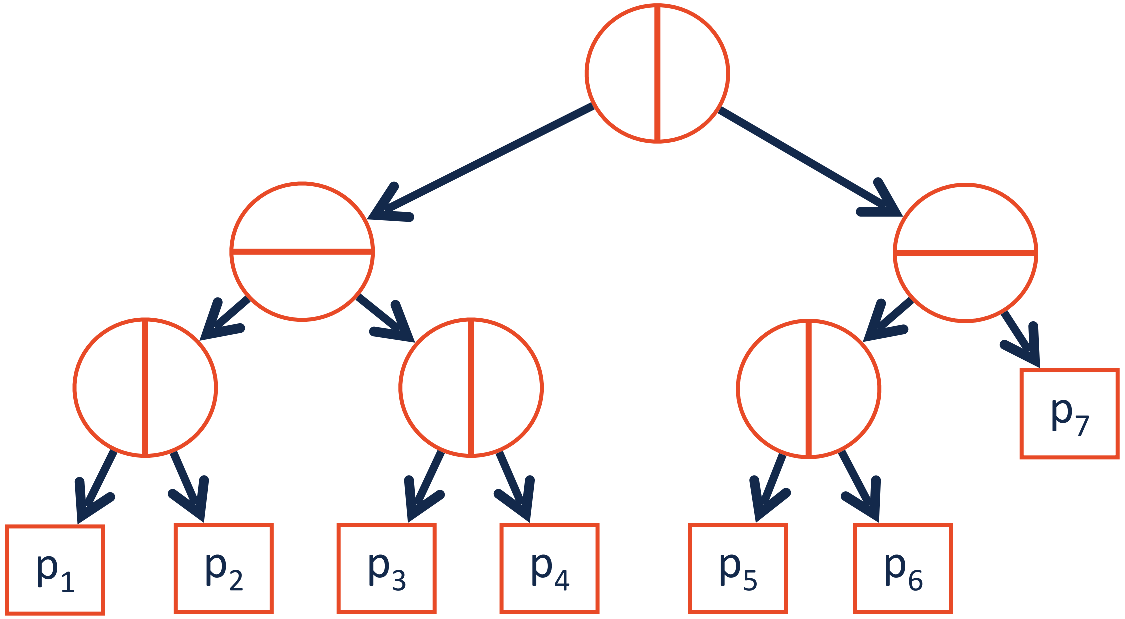
Range-based Searches





Range-based Searches

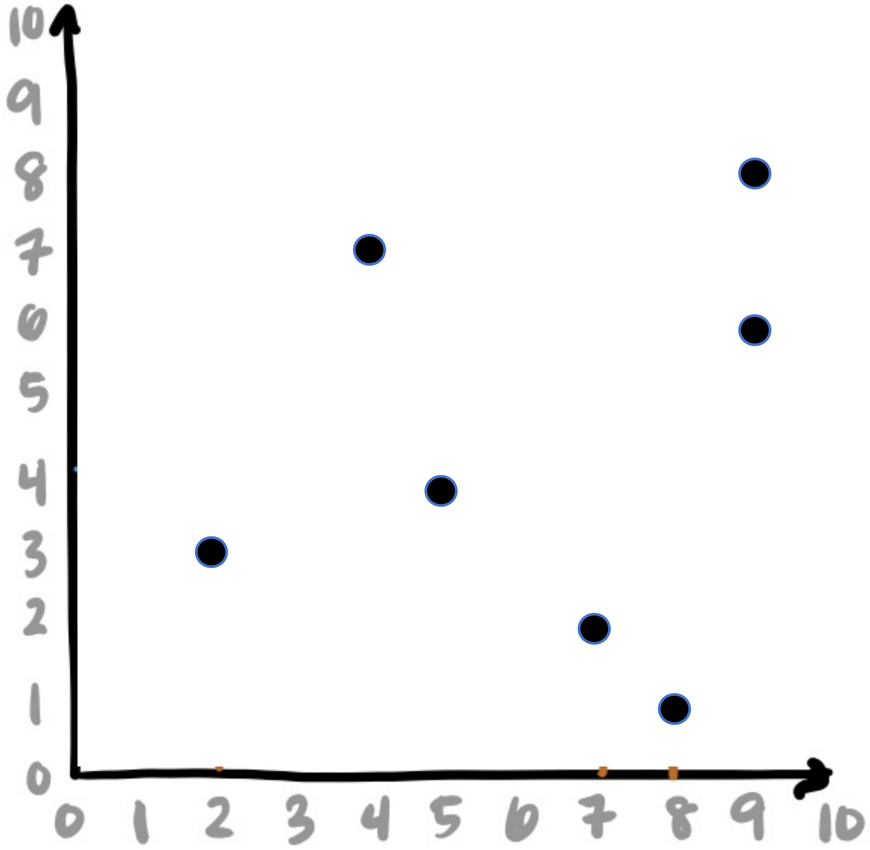




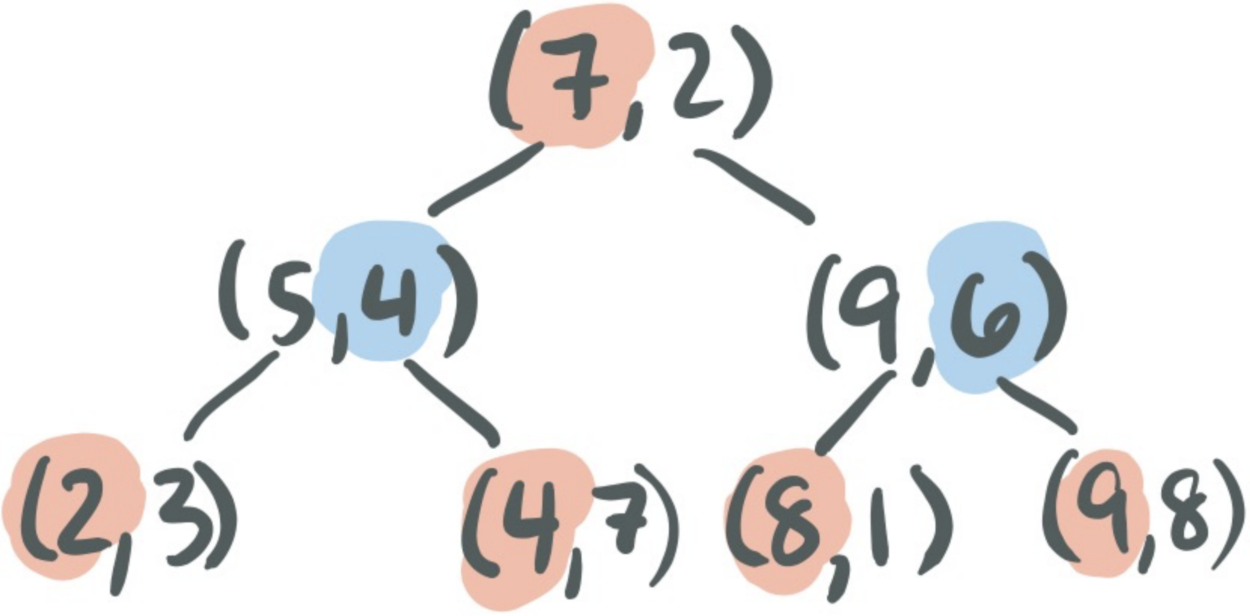
Nearest Neighbor: k-d tree

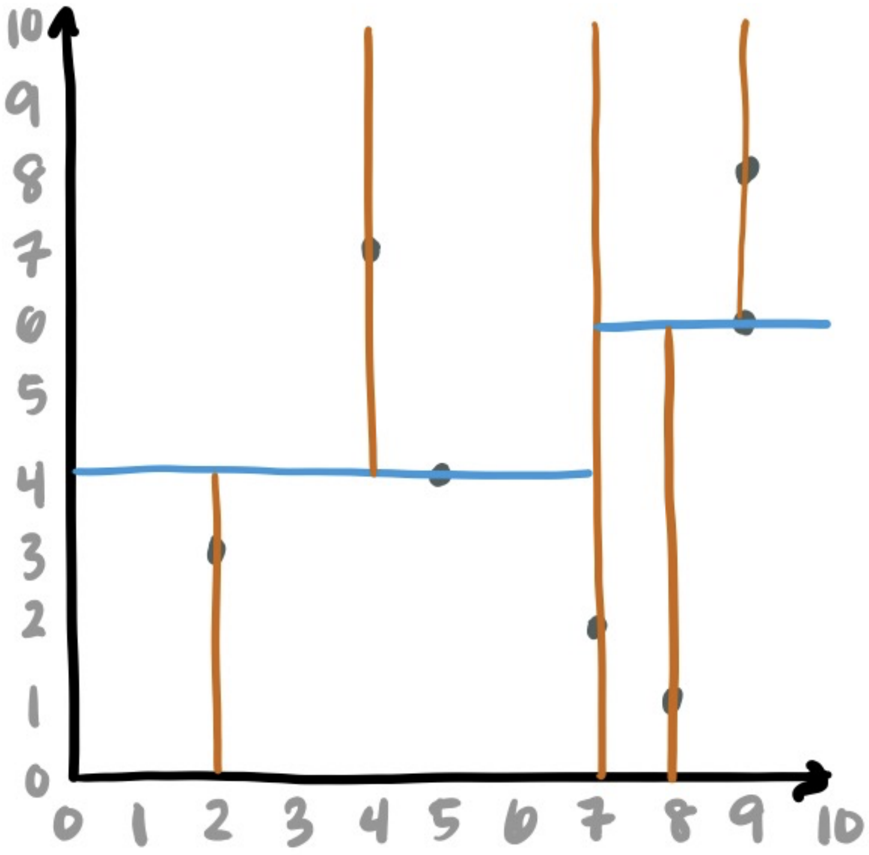
A **k-d tree** is similar but splits on points:

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



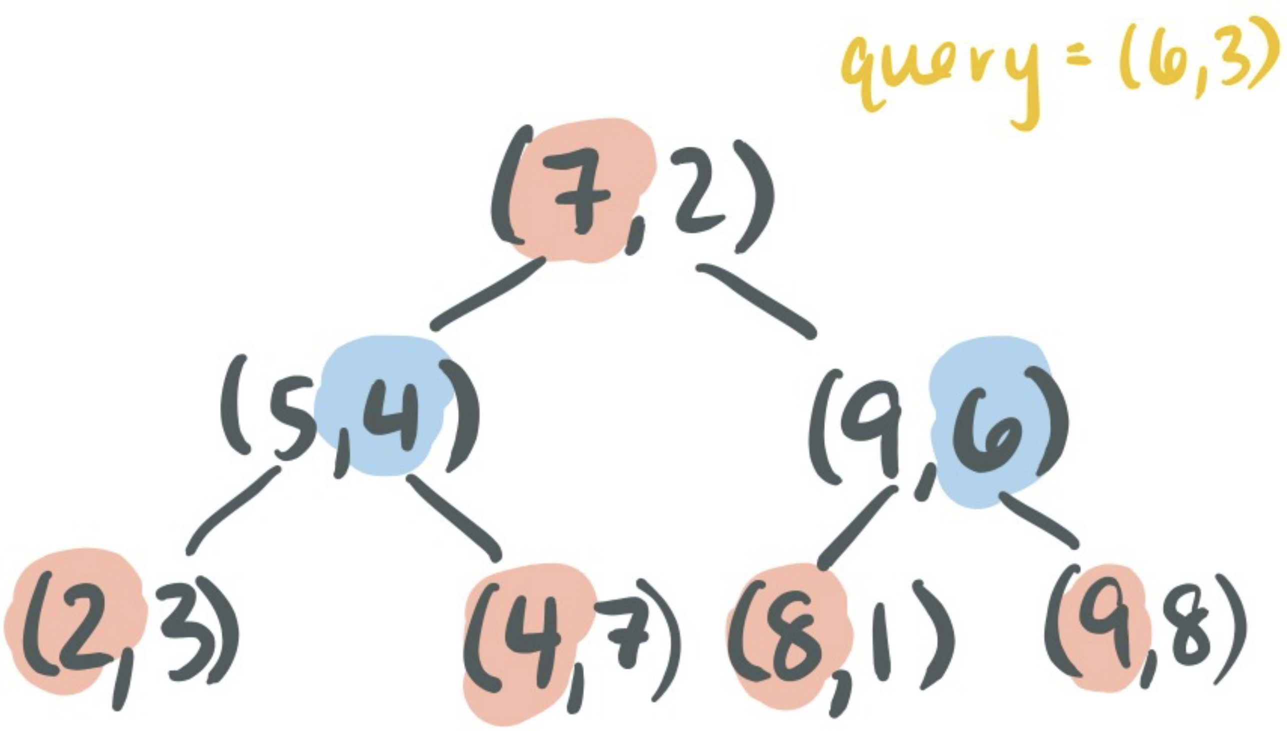
Nearest Neighbor: k-d tree

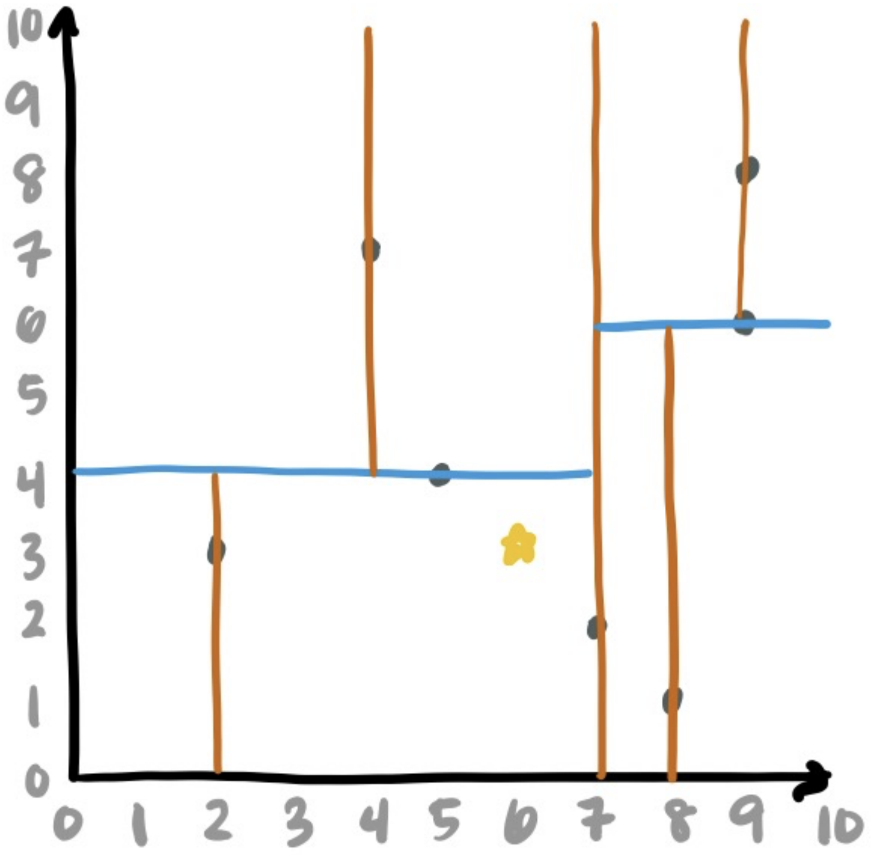




Nearest Neighbor: k-d tree

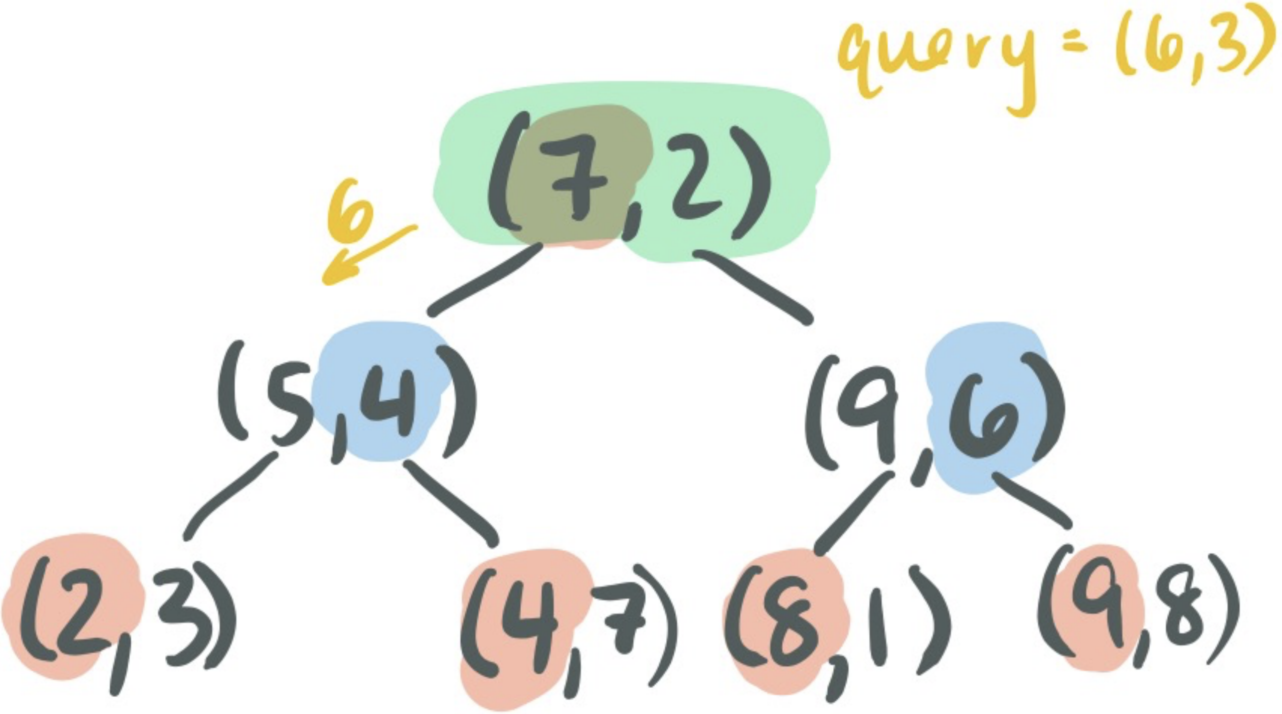
When querying a k-d tree, it acts like a BST* at first…

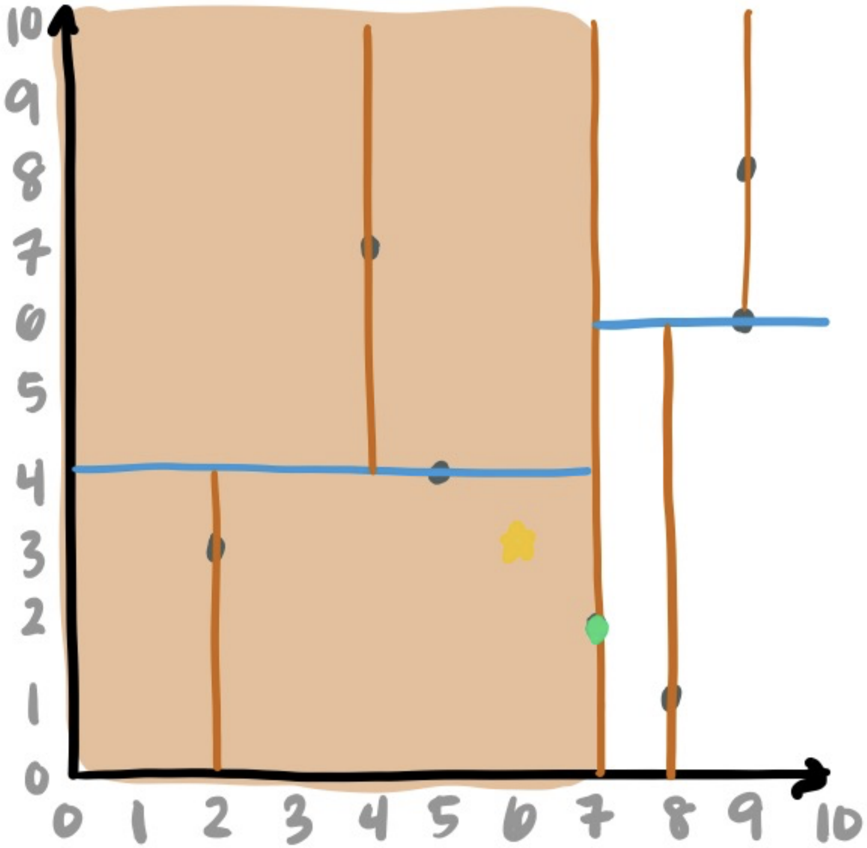




Nearest Neighbor: k-d tree

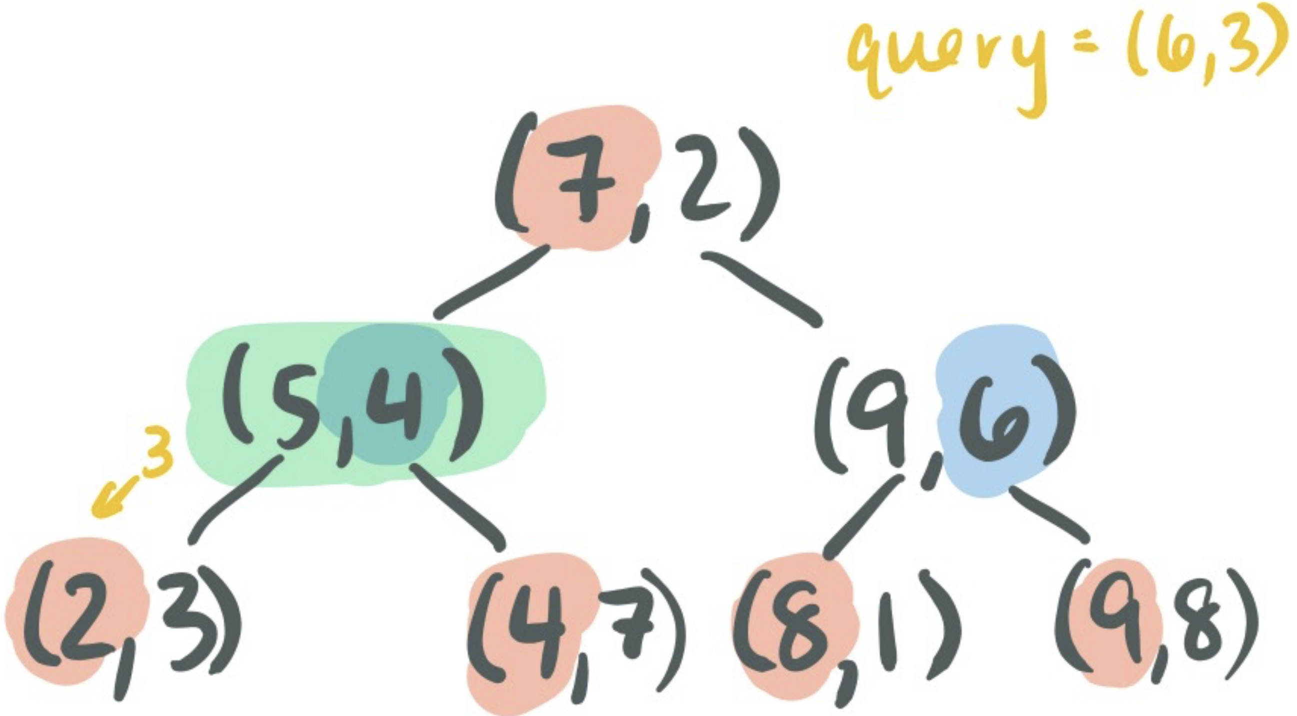
When querying a k-d tree, it acts like a BST* at first…

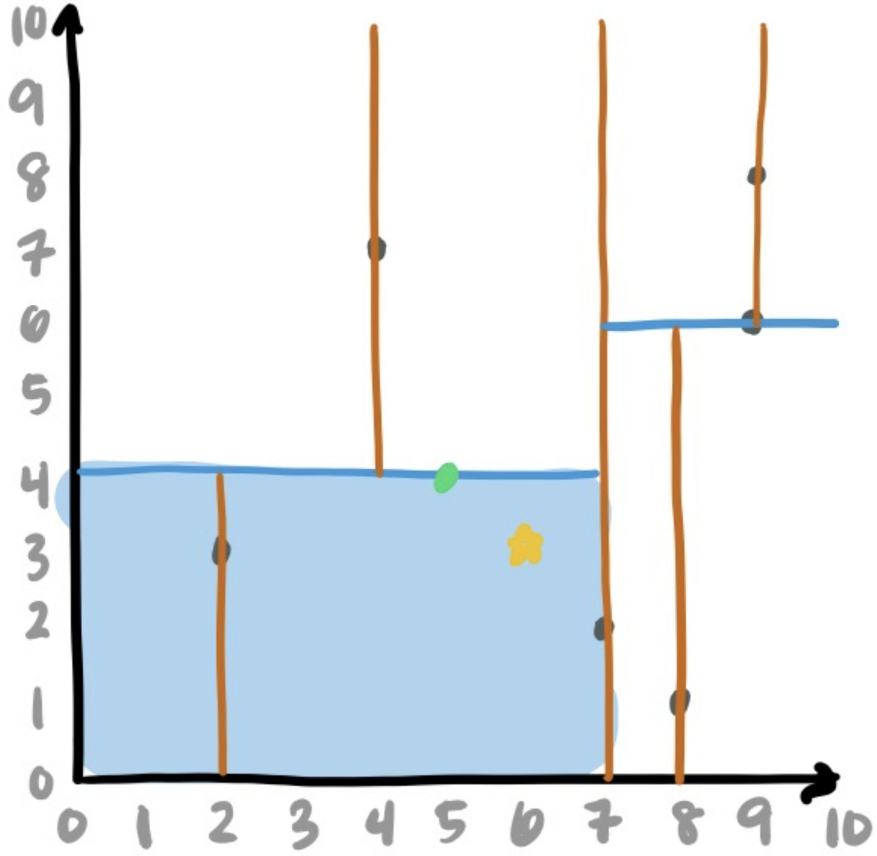




Nearest Neighbor: k-d tree

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

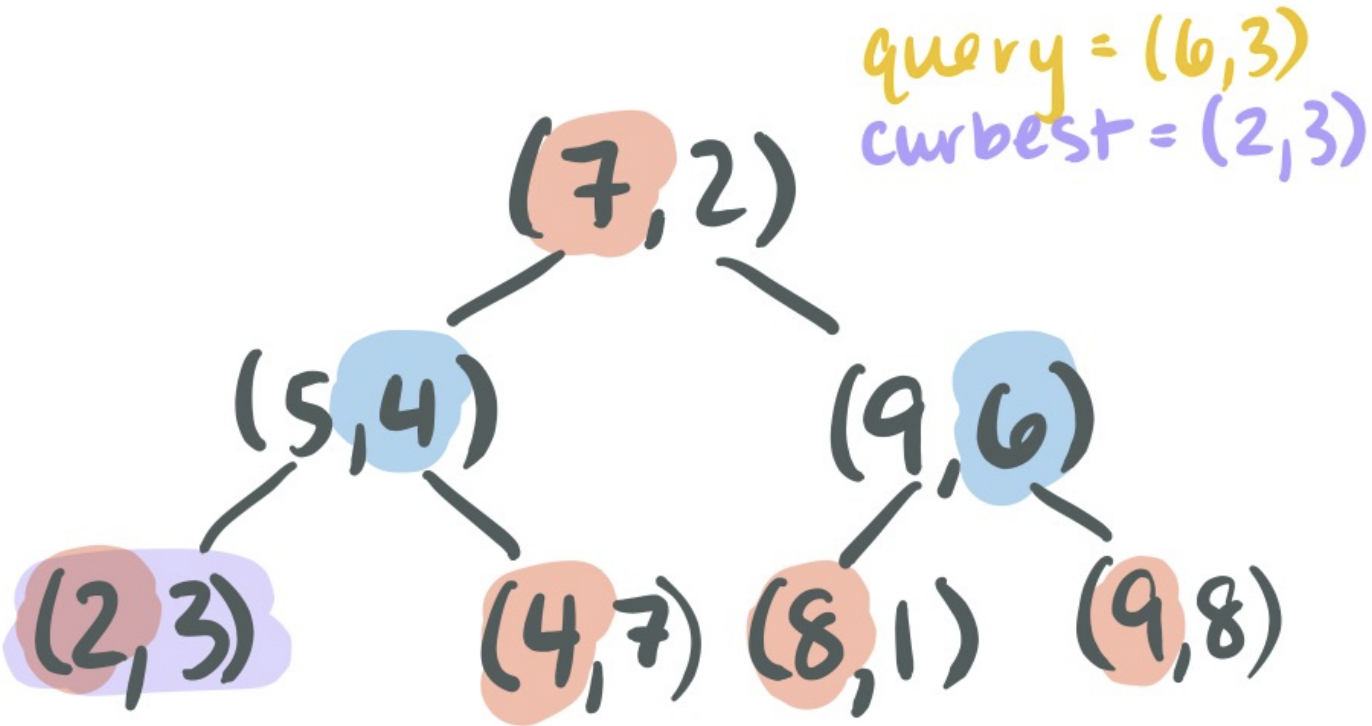


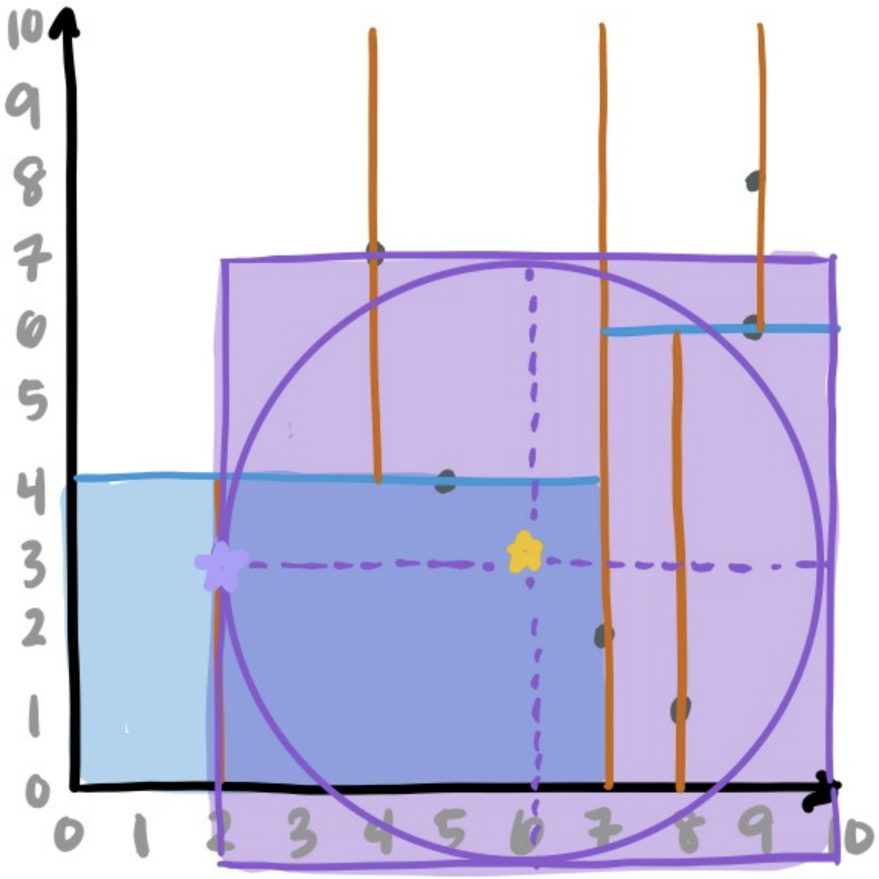


Nearest Neighbor: k-d tree

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

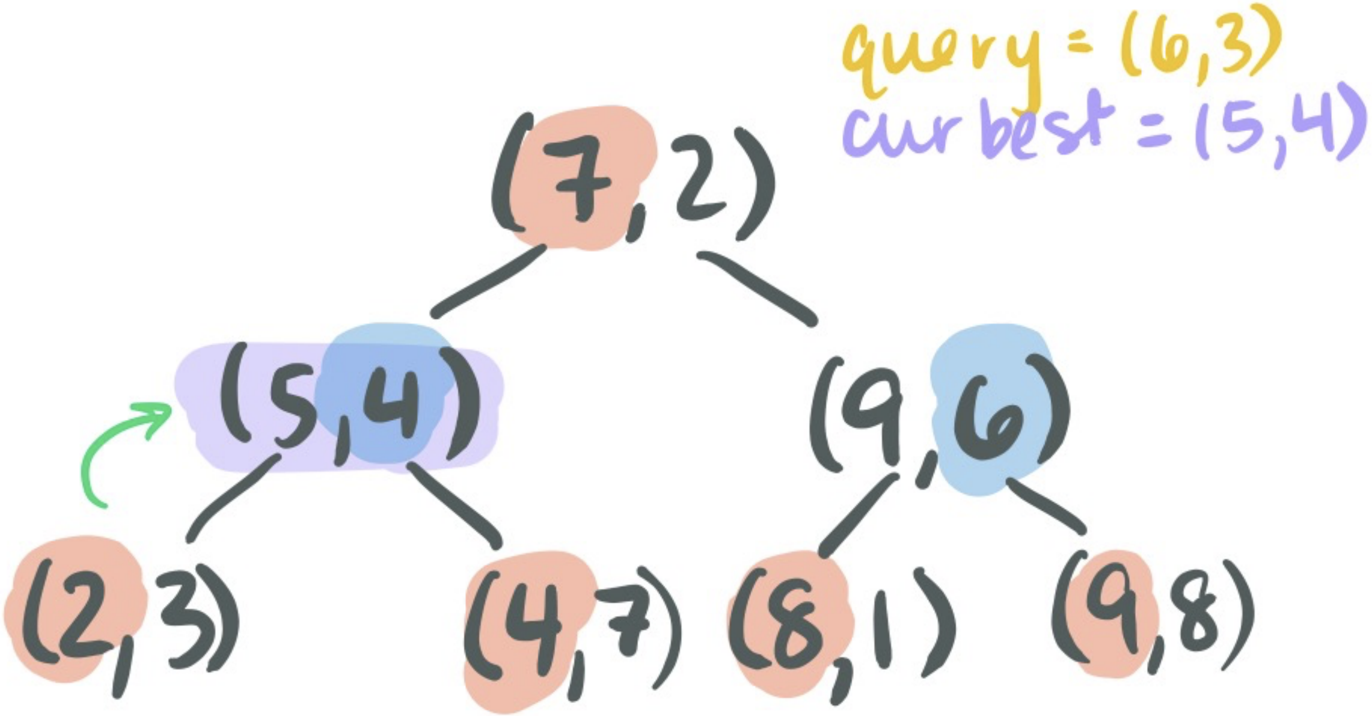
… But if we dont find exact match, have to find nearest neighbor

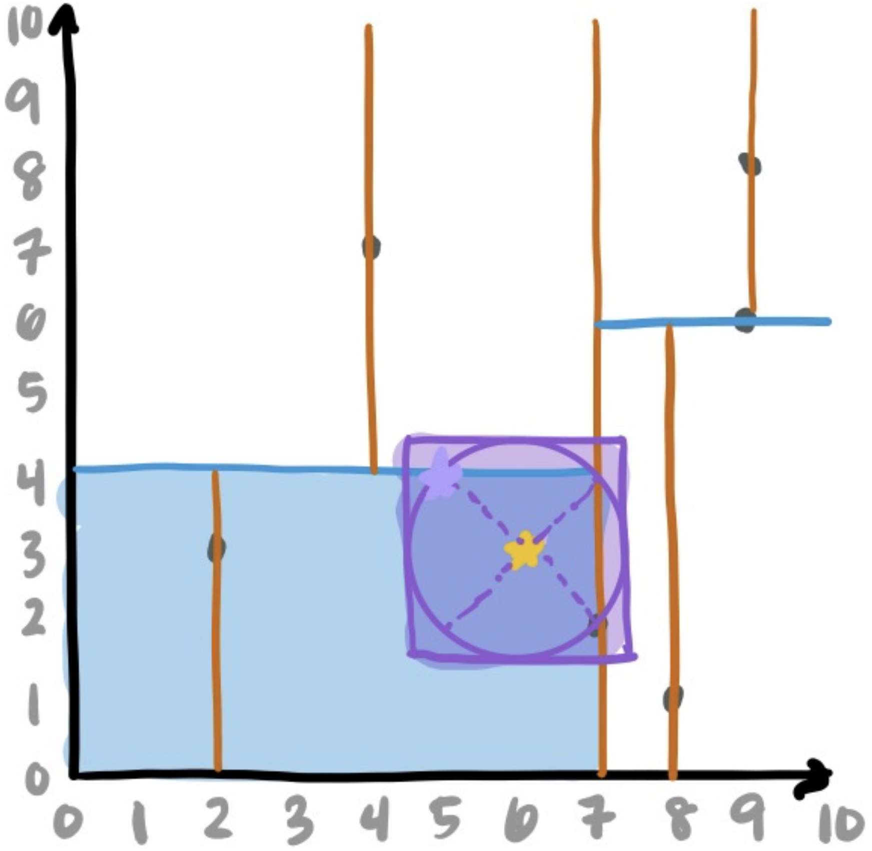




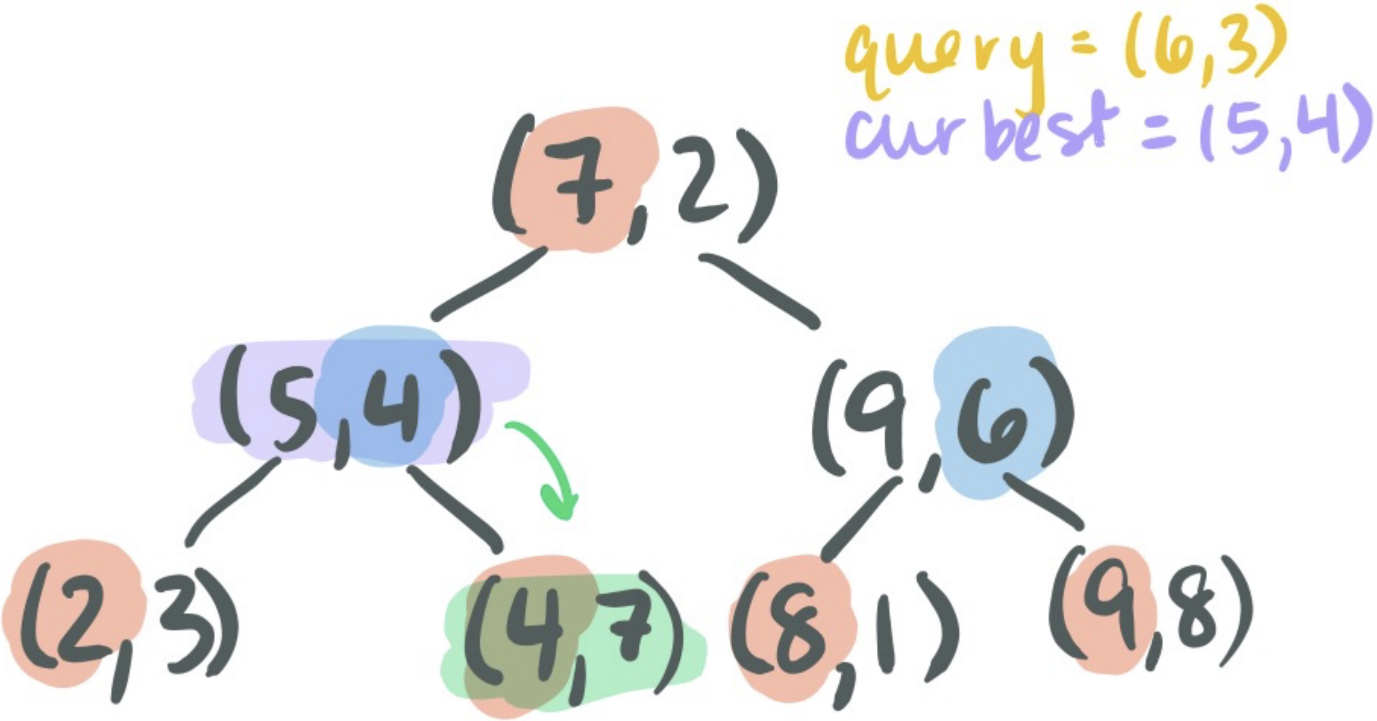
Nearest Neighbor: k-d tree

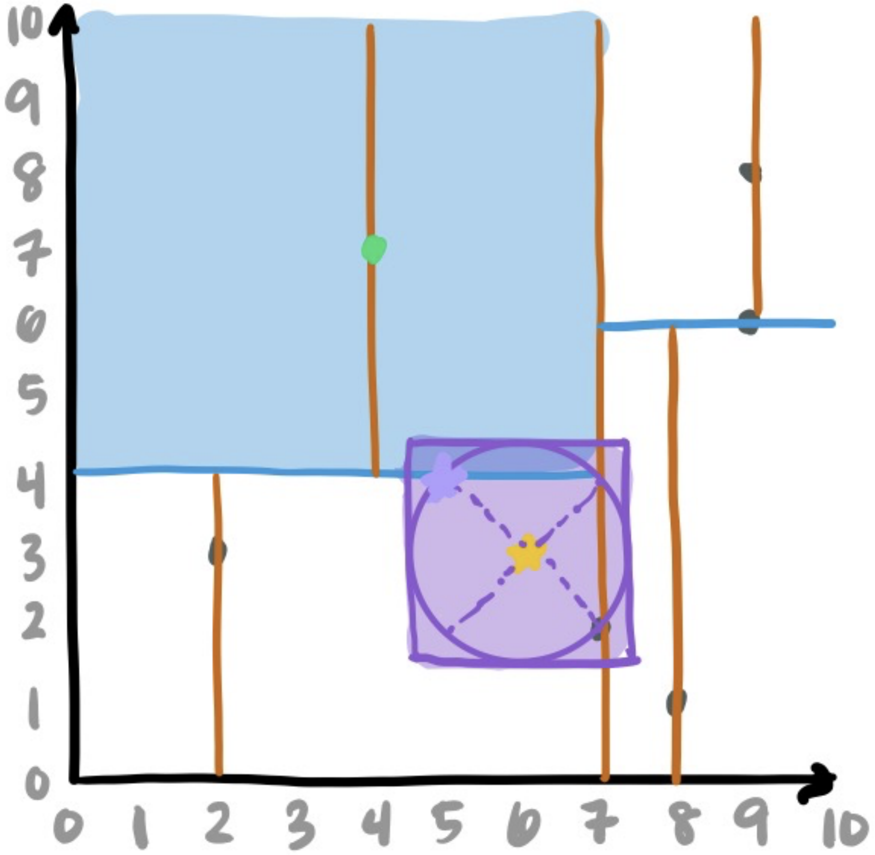
Backtracking: start recursing backwards -- store “best” possibility as you trace back





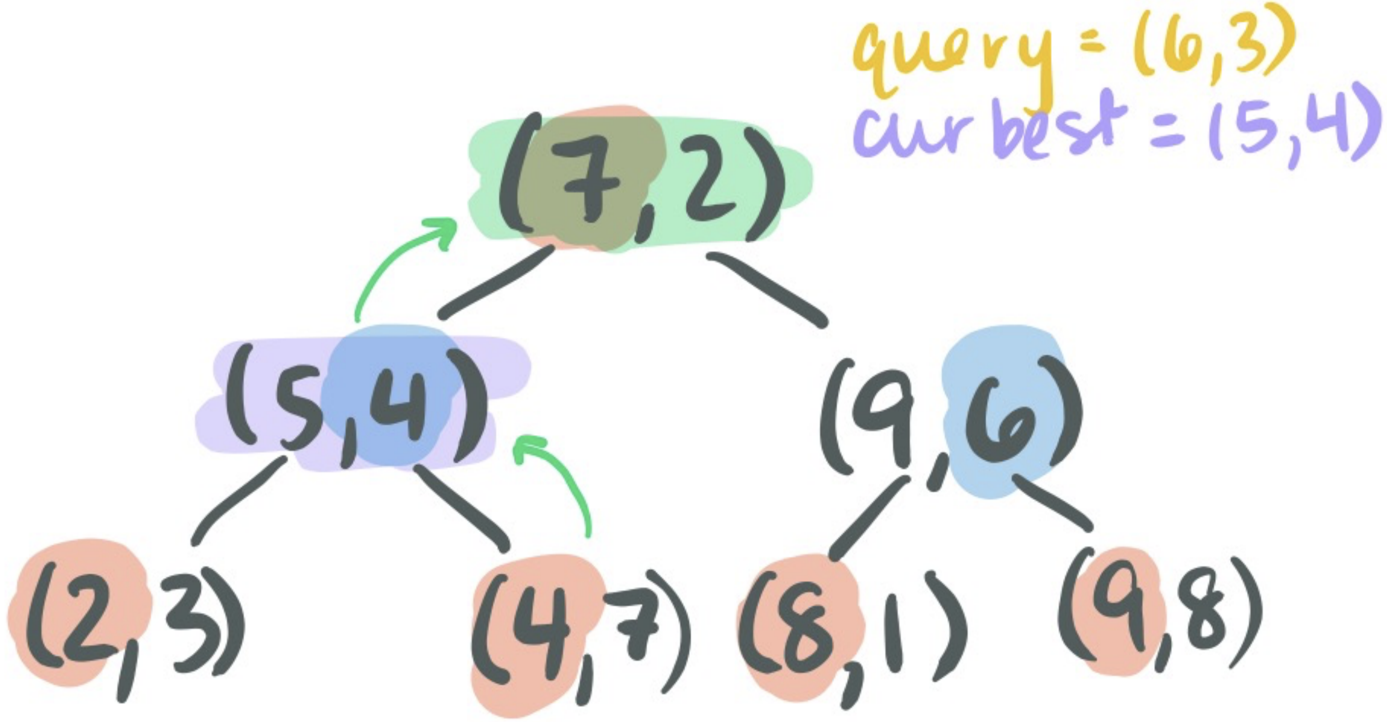
Nearest Neighbor: k-d tree

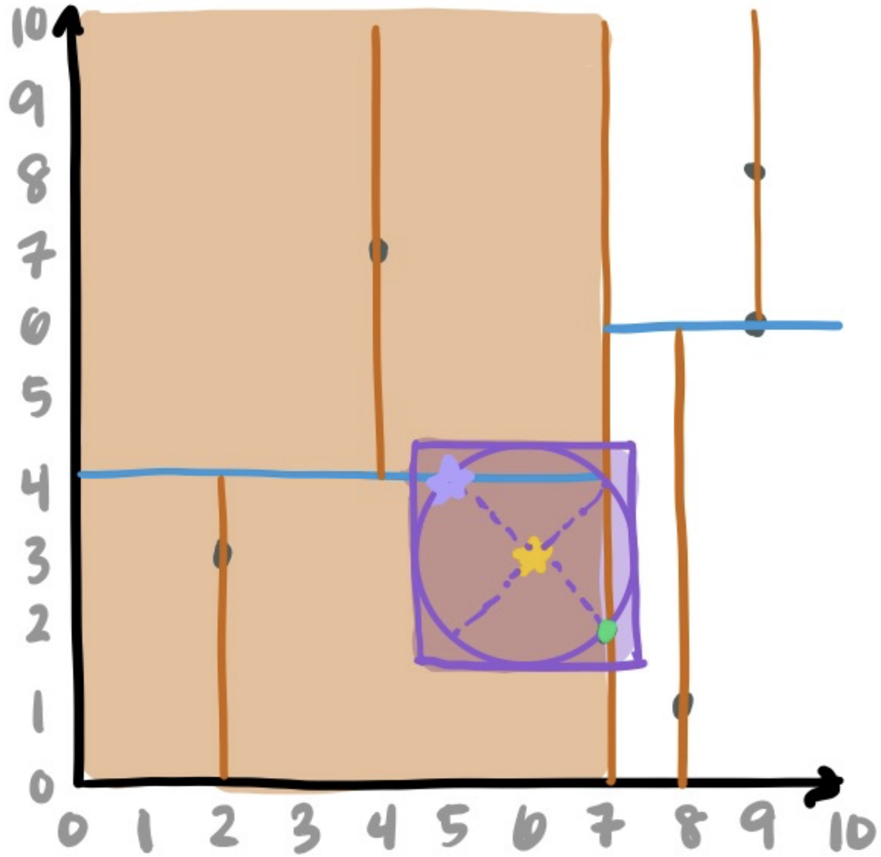




Nearest Neighbor: k-d tree

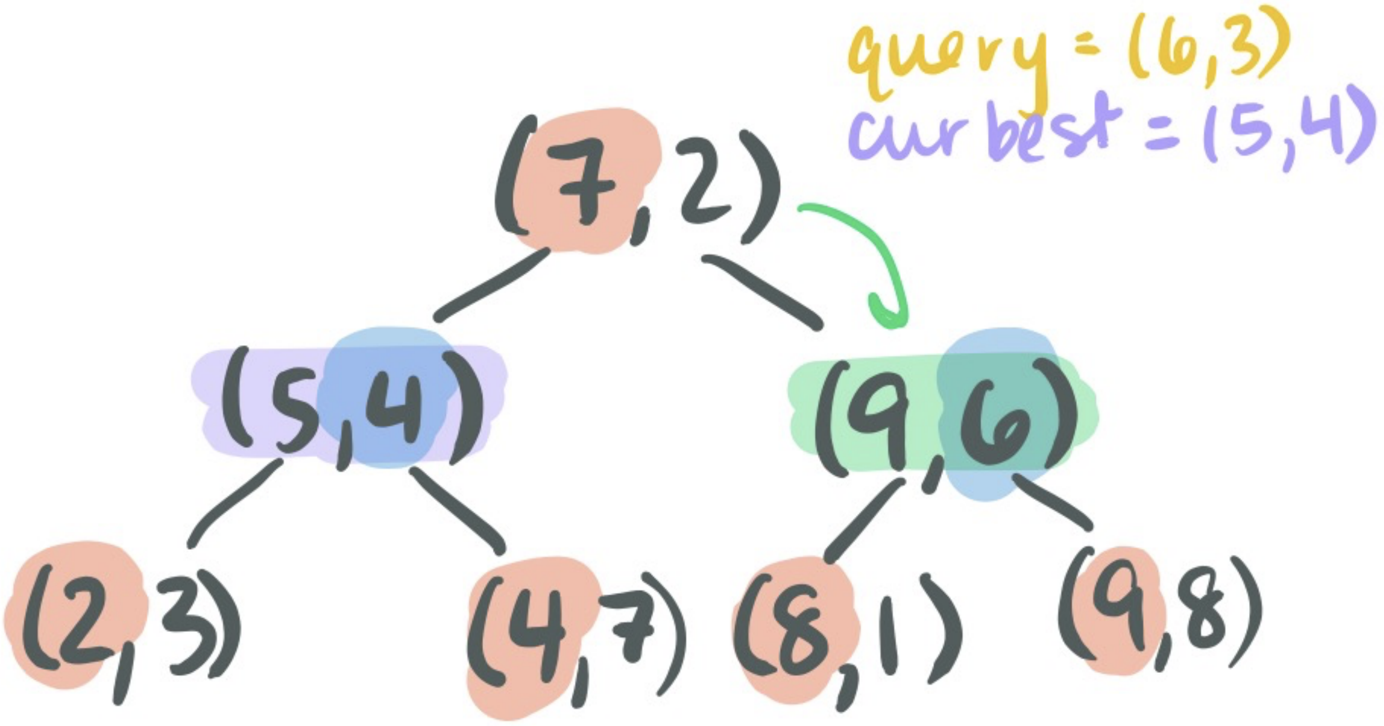
On ties, use smallerDimVal to determine which point remains curBest

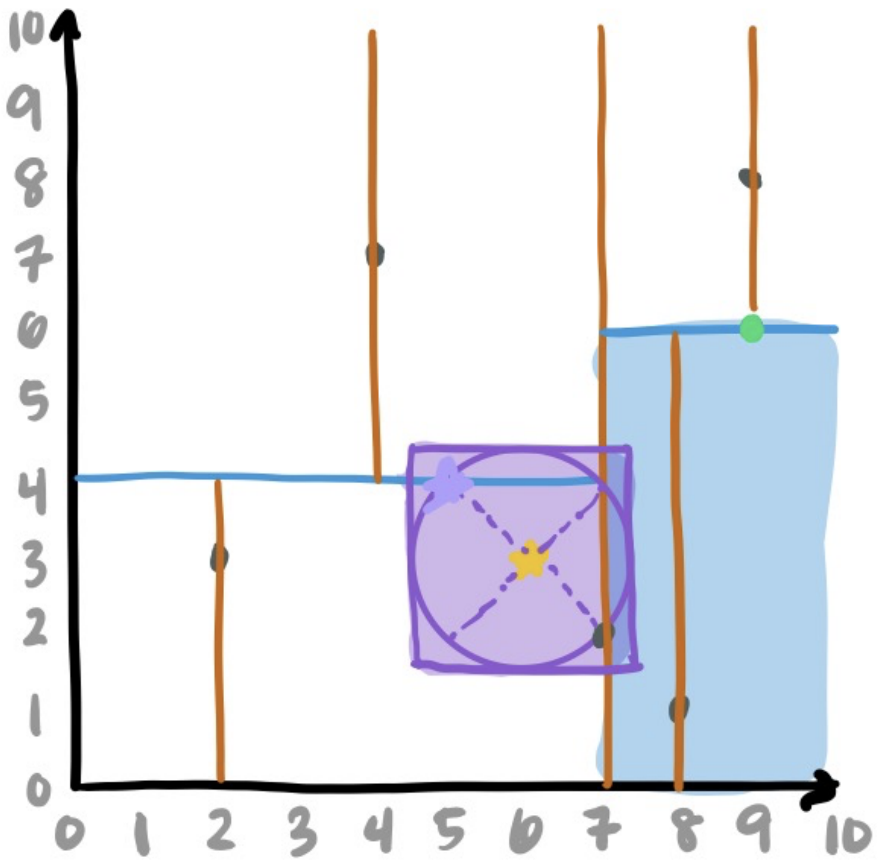




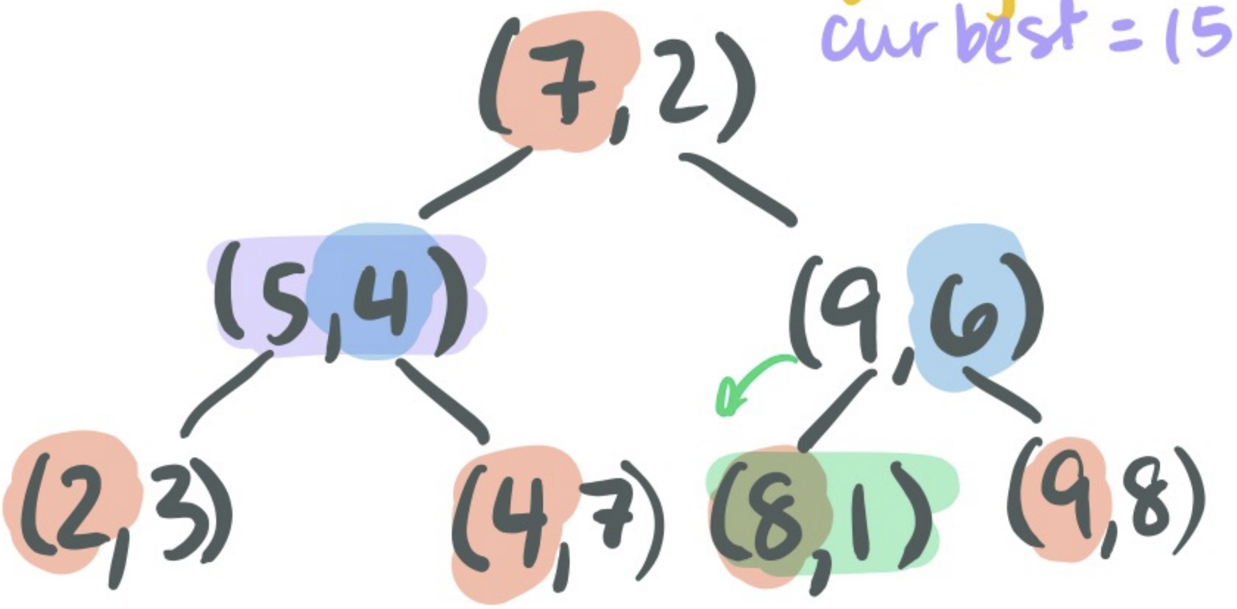
Nearest Neighbor: k-d tree

Why do we need to explore this subtree?



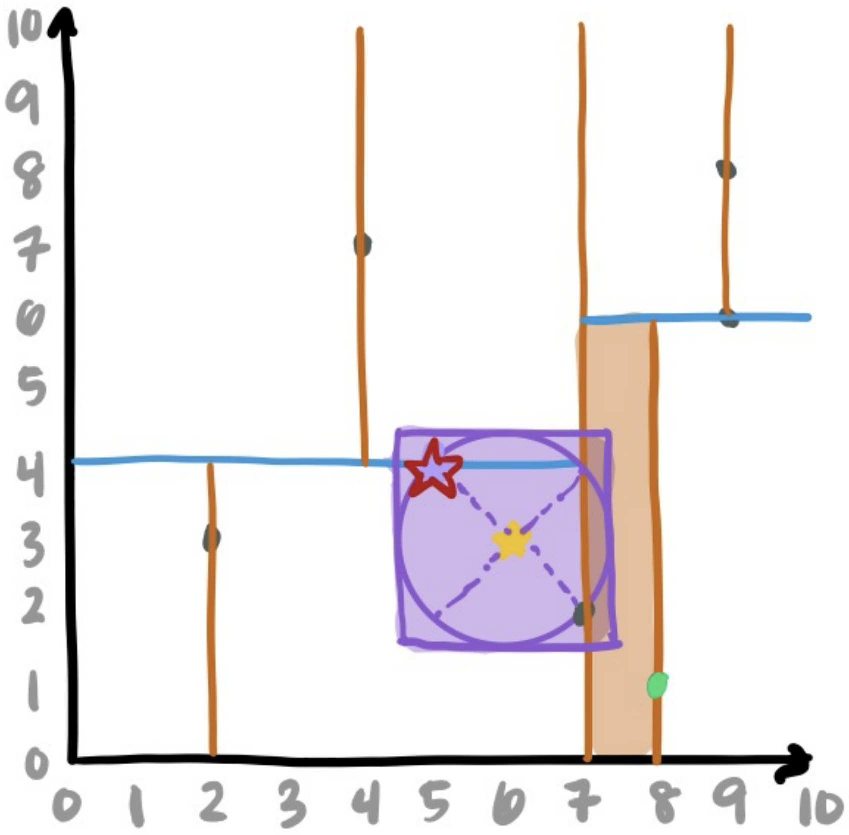


Nearest Neighbor: k-d tree



query = (6,3)

cur best = (5,4)



BEST : (5,4)

Tips and Tricks for MP_Mosaics

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

```
template <typename RandIter, typename Comparator>
void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
{
    /**
     * @todo Implement this function!
     */

  }

```

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

[Capture](Arg List){ Function Body}

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()

```
template <typename RandIter, typename Comparator>
void BlackBox(RandIter A, RandIter B)
{

   std::swap(*A, *B);


}
```

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**

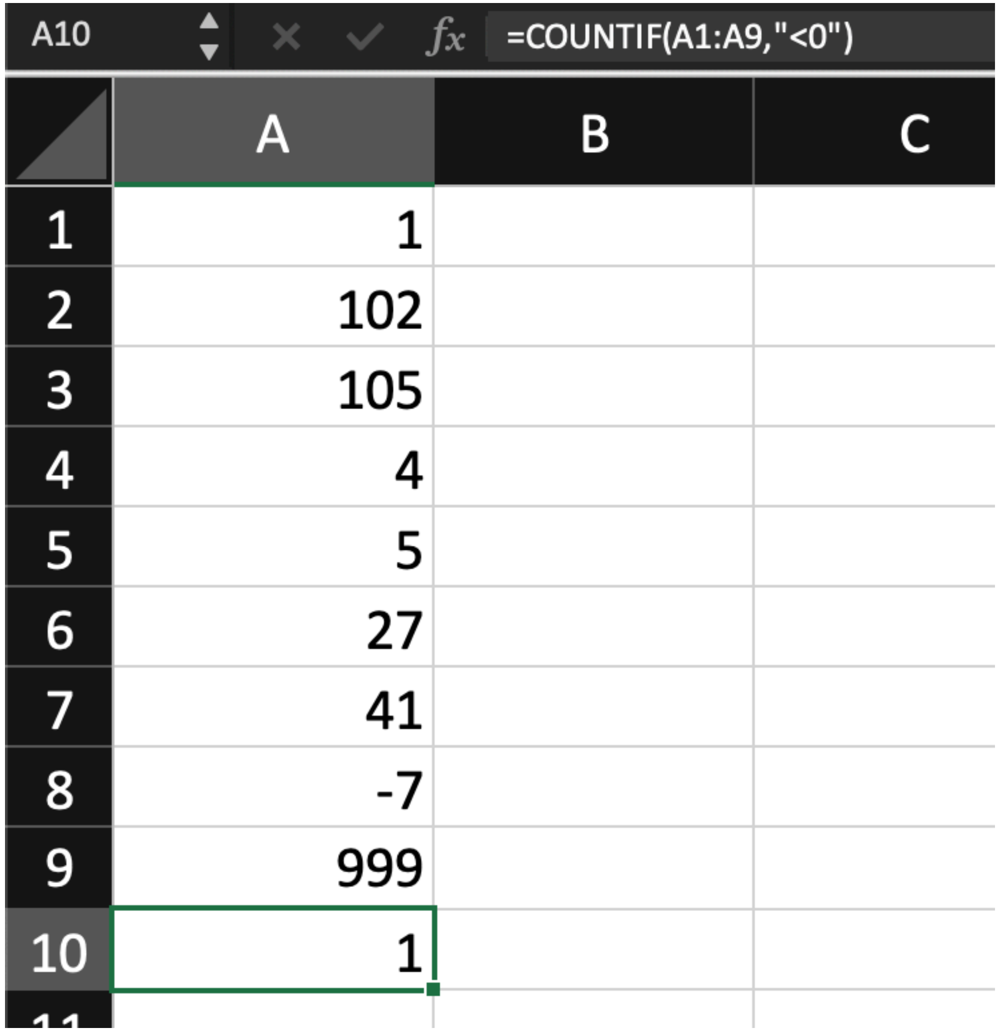
```
template <typename RandIter, typename Comparator>
void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
{
    /**
     * @todo Implement this function!
     */

  }

```

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

[Capture](Arg List){ Function Body}

Functions as arguments

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



A10

=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		

Functions as arguments

Countif.hpp

```
template <typename Iter, typename Pred>
int Countif(Iter begin, Iter end, Pred pred) {
  int count = 0;
  auto cur = begin;

  while(cur != end) {
    if(pred(*cur))
      ++count;
    ++cur;
  }

  return count;
}
```

Lambda Functions in C++

Here are several ways to write a function as an object

main.cpp

```
bool isNegative(int num) { return (num < 0); }

class IsNegative {
public:
    bool operator() (int num) { return (num < 0); }
};

int main() {
  std::vector<int> numbers = {1, 102, 105, 4, 5, 27, 41, -7, 999};

  auto isnegl = [](int num) { return (num < 0); };
  auto isnegfp = isNegative;
  auto isnegfunctor = IsNegative();

  cout << "There are " << Countif(numbers.begin(), numbers.end(), _______)
    << " negative numbers" << std::endl;

```

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

```
  int big;
  std::cout << "How big is big? ";
  std::cin >> big;


  auto isbig = [big](int num) { return (num >= big); };


  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)
    << " big numbers" << std::endl;
}
```

Lambda Functions in C++

main.cpp

```cpp
  int big;
  std::cout << "How big is big? ";
  std::cin >> big;


  auto isbig = [big](int num) { return (num >= big); };


  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)
    << " 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