Data Structures

KD Tree 2

CS 225

Harsha Tirumala

September 29, 2025

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Announcements

Exam 2 this week - 10/01 to 10/03

Review of practice exam 2 - notes + video

MP lists due today

Learning Objectives

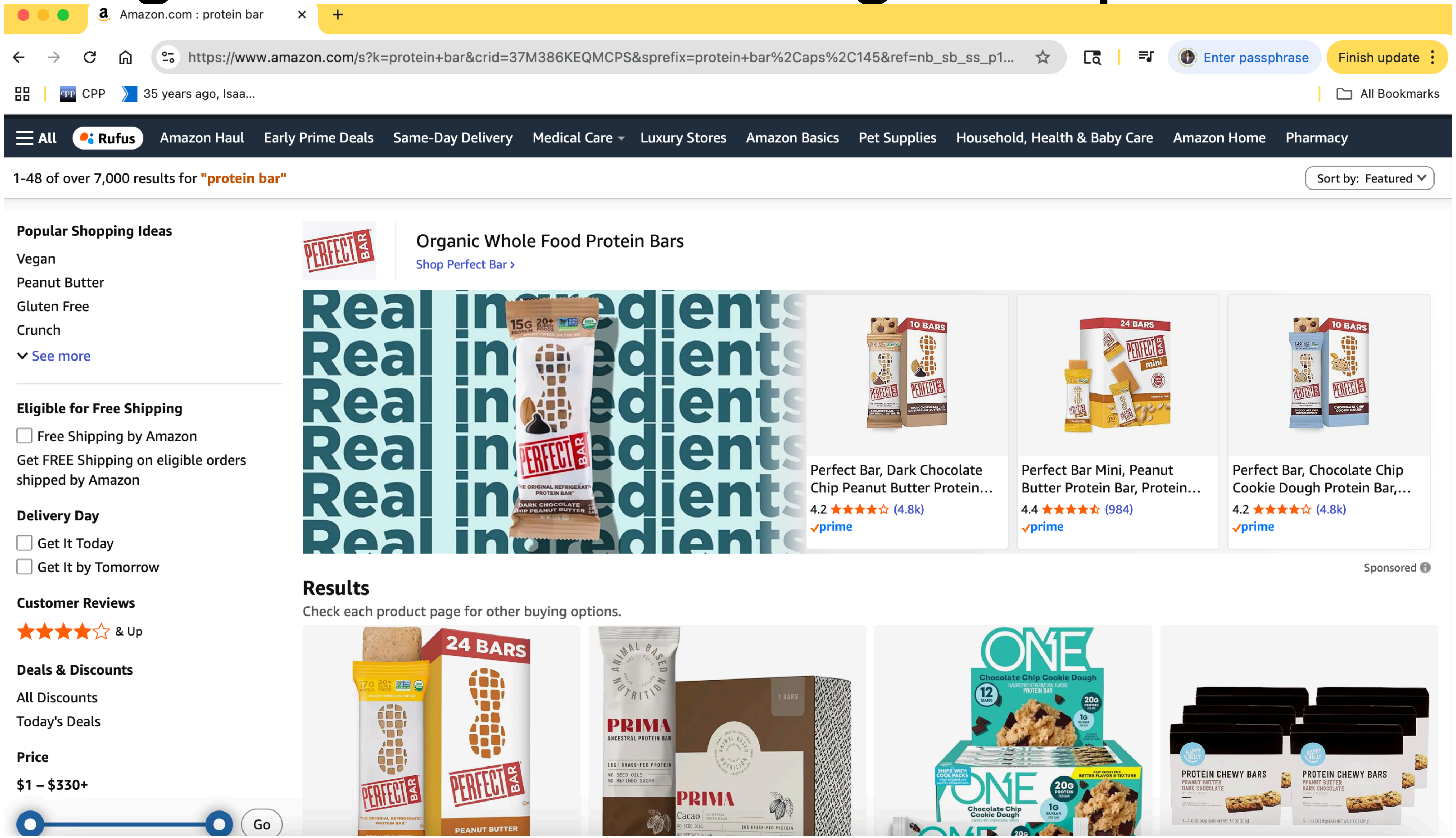
KD tree : Motivation and Creation

KD tree : Interval Search and Nearest Neighbor

KD tree : Pros and Cons

Go over C++ concepts for mp_mosaics **(probably shared as a separate video later)**

Range Search : Motivating example

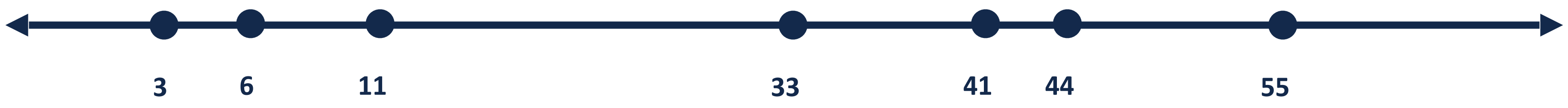


Want Protein bars that cost between 10$ and 20$ as well as protein between 5g and 15g

Range-based Searches

Consider a collection of points on a 1D line: p = {p1, p2, …, pn}

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



Range-based Searches : Brute Force

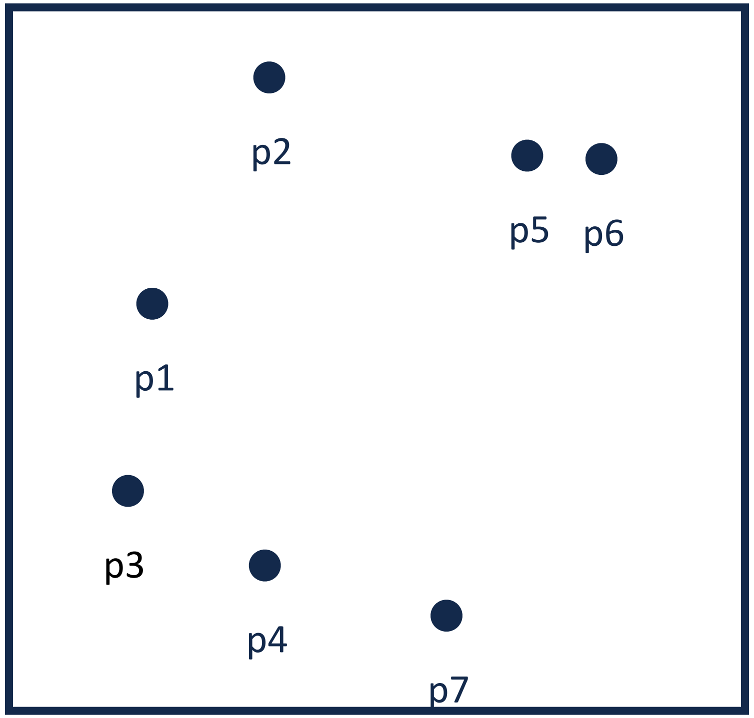
Consider points in 2D: **p = {p1, p2, ..., pn}**

What points in rectangle [(x1, y1), (x2, y2)]?

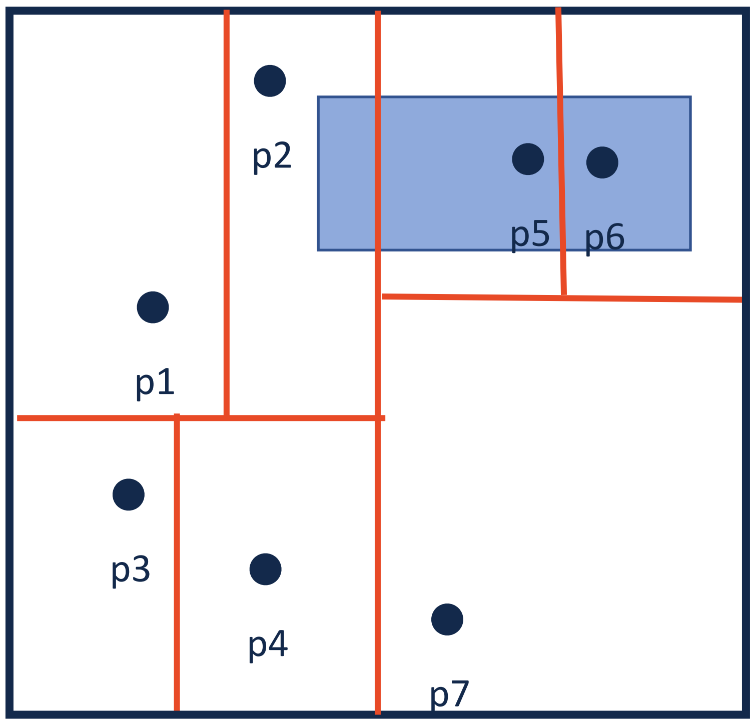
1. Brute Force : Check each point for validity

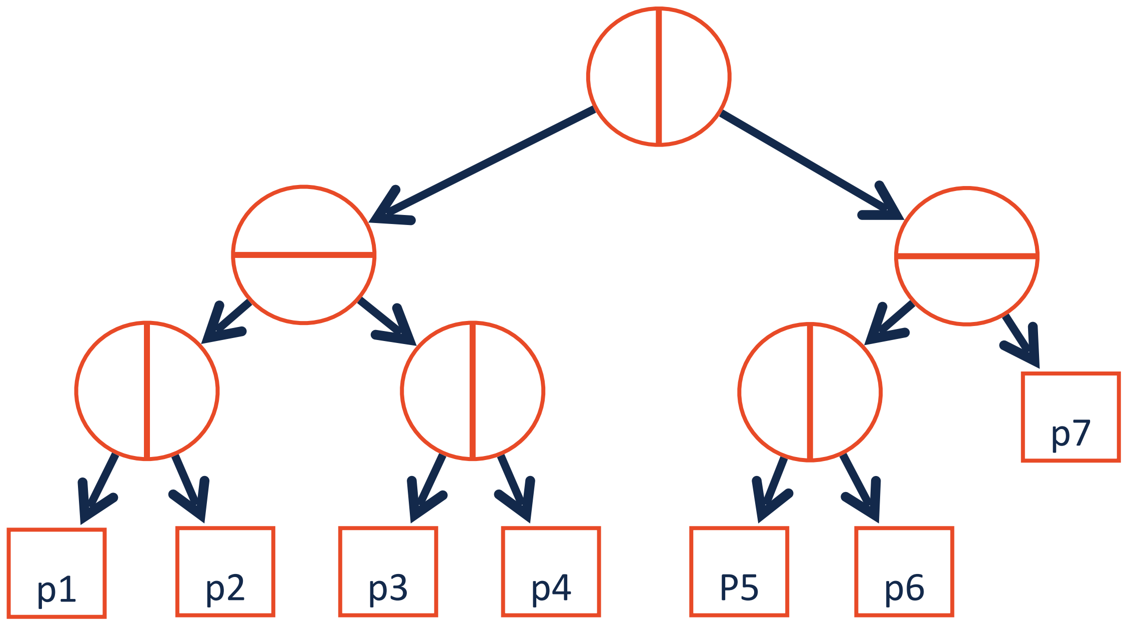
(x1 <= x <= x2 && y1 <= y <= y2)

For k dimensional data : **O(kn)**



Range-based Searches (bisecting planes)





k-d tree : Example

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

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

Step 1 - Split on x-median (7,2)

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

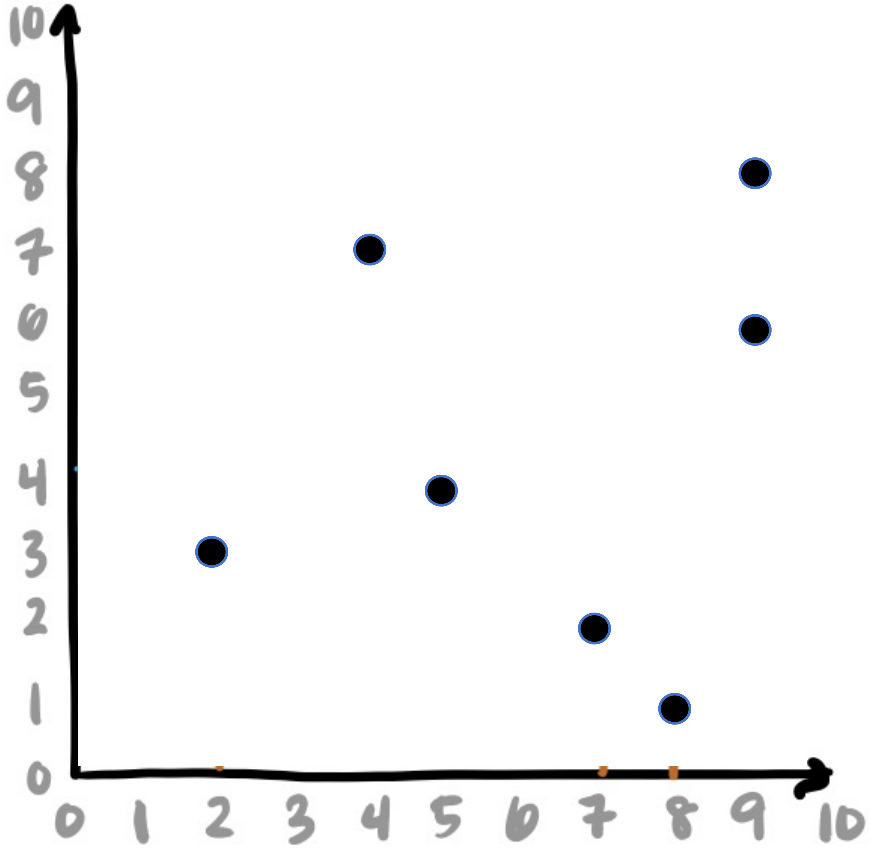
Step 2 - Split on y-median

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

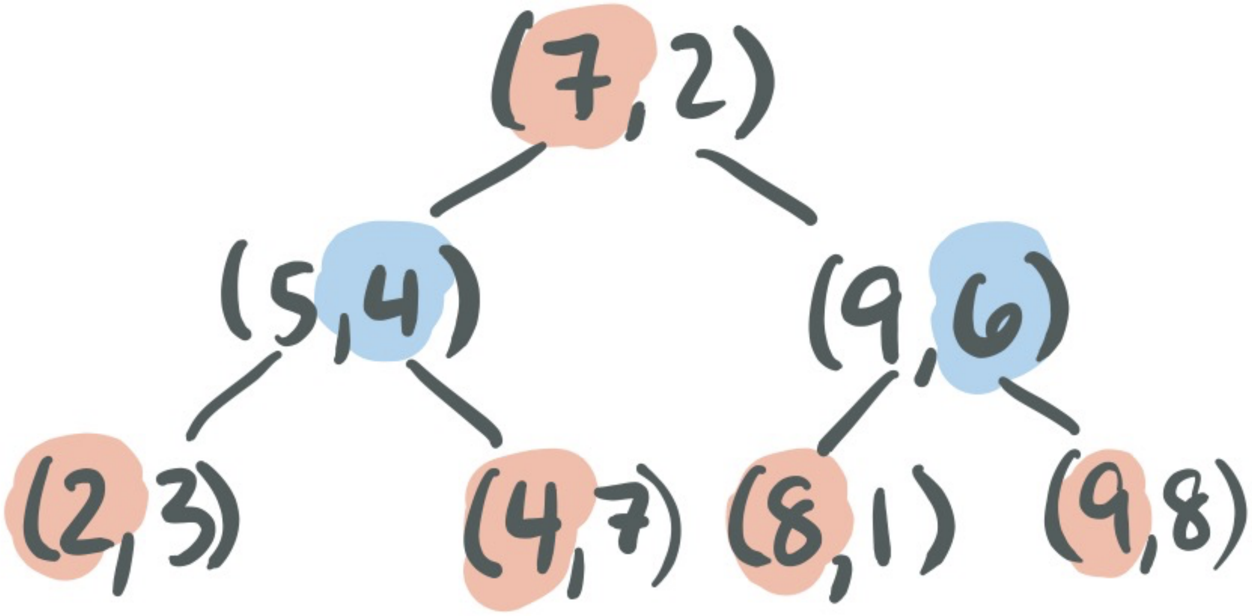
(2,3)   (4,7)   (8,1)   (9,8)

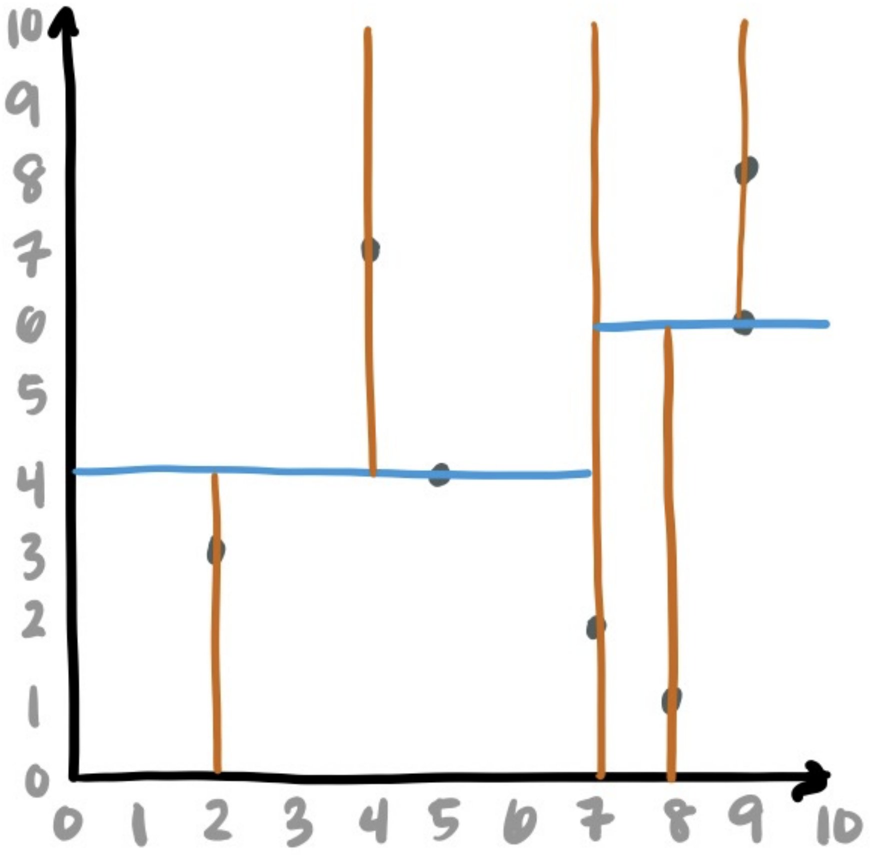
Step 3 - Split on x-median

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



k-d tree : Build

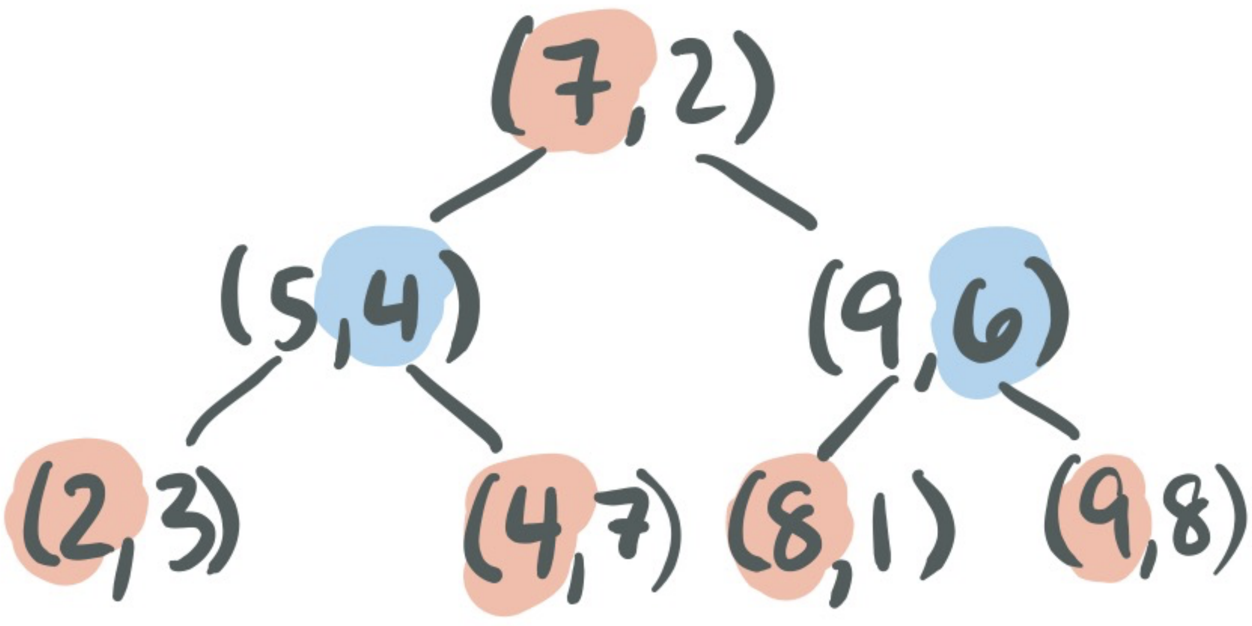


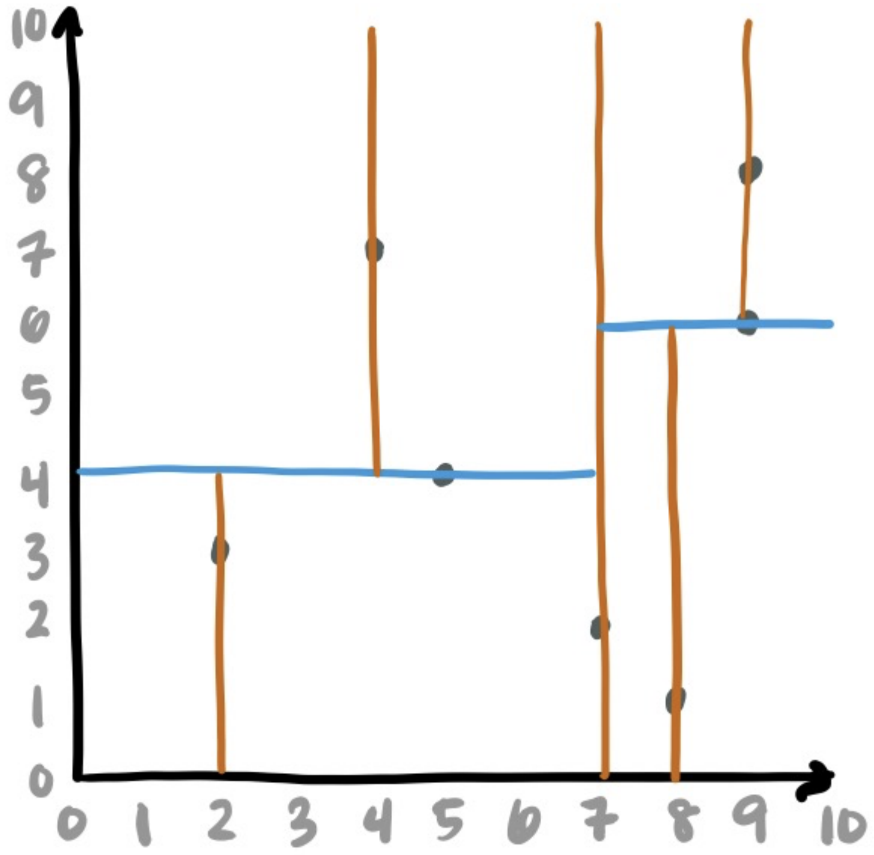


k-d tree : Properties

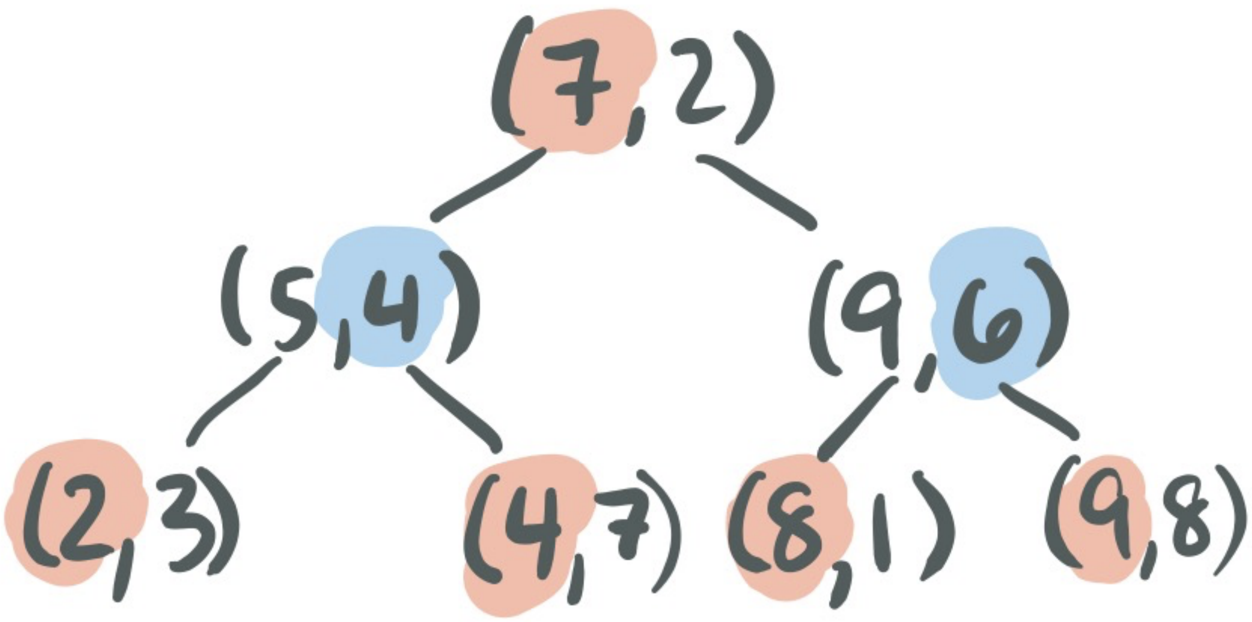
Height of a kd -tree on n points : O(log n)

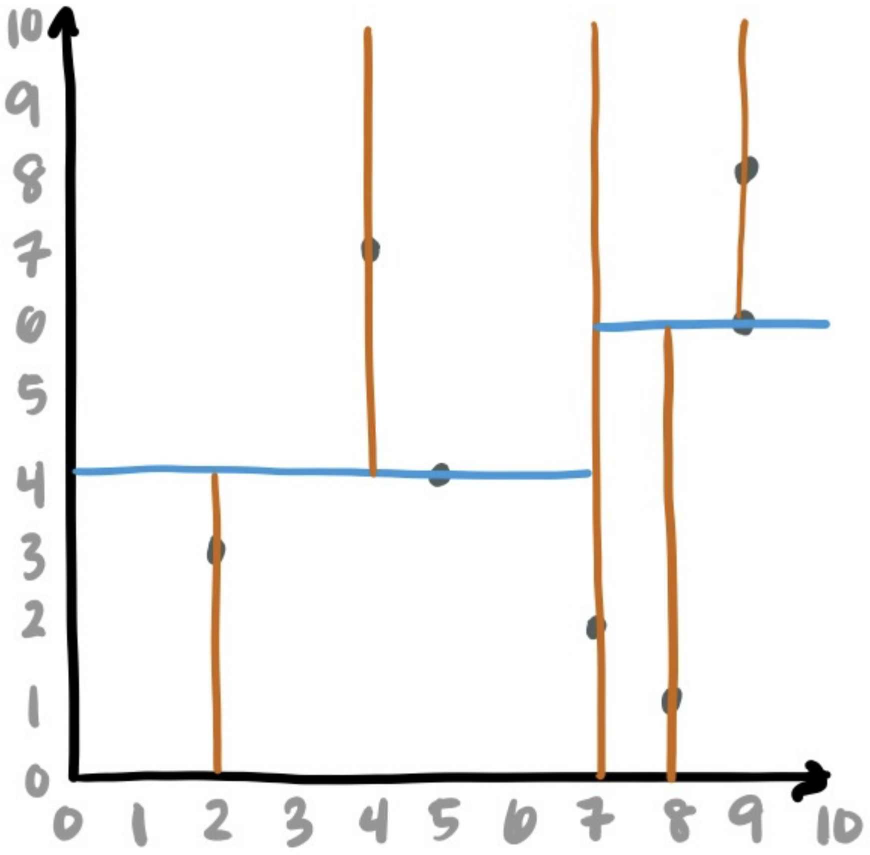
Time complexity of building a kd -tree on n points : O(n log n)





k-d tree : Range Search

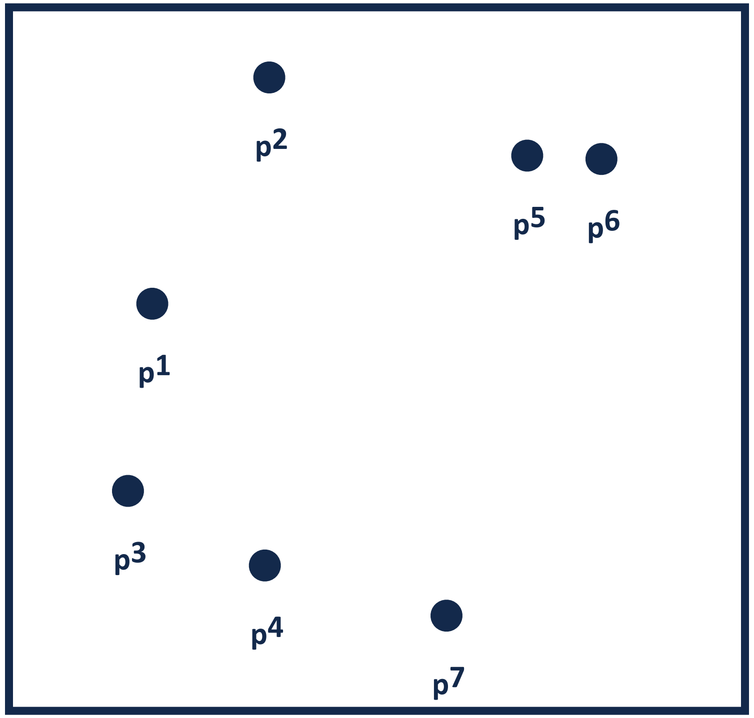




Nearest Neighbor search

Consider points in 2D: p = {p1, p2, …, pn}

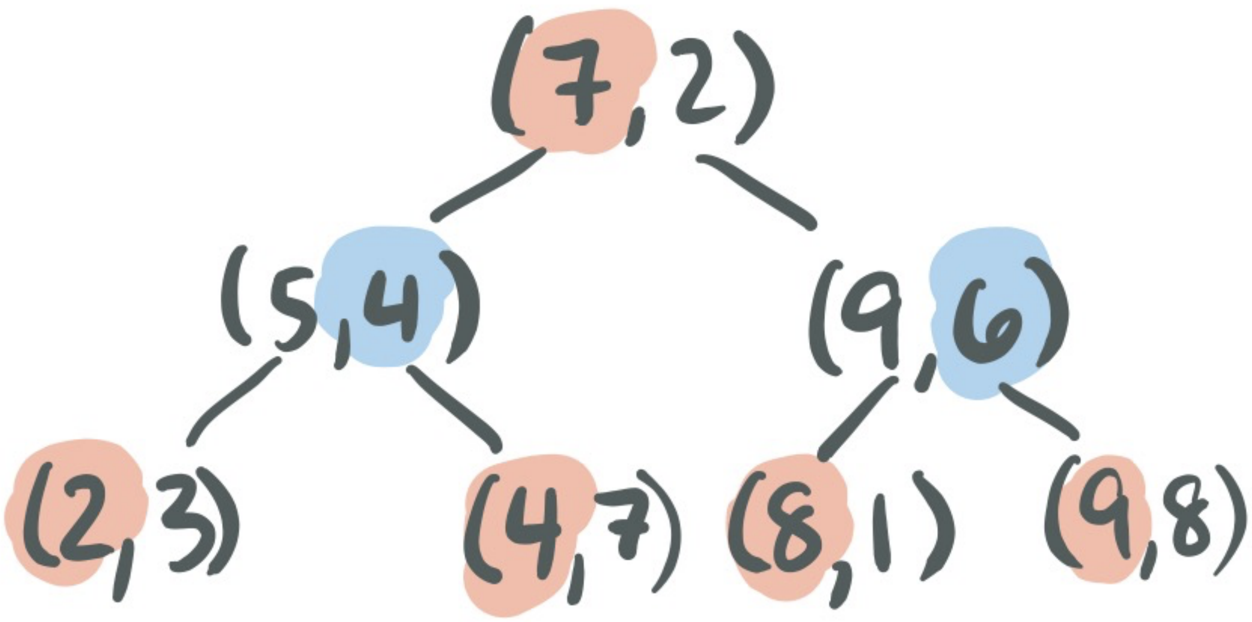
What is nearest point to (x1, y1)?

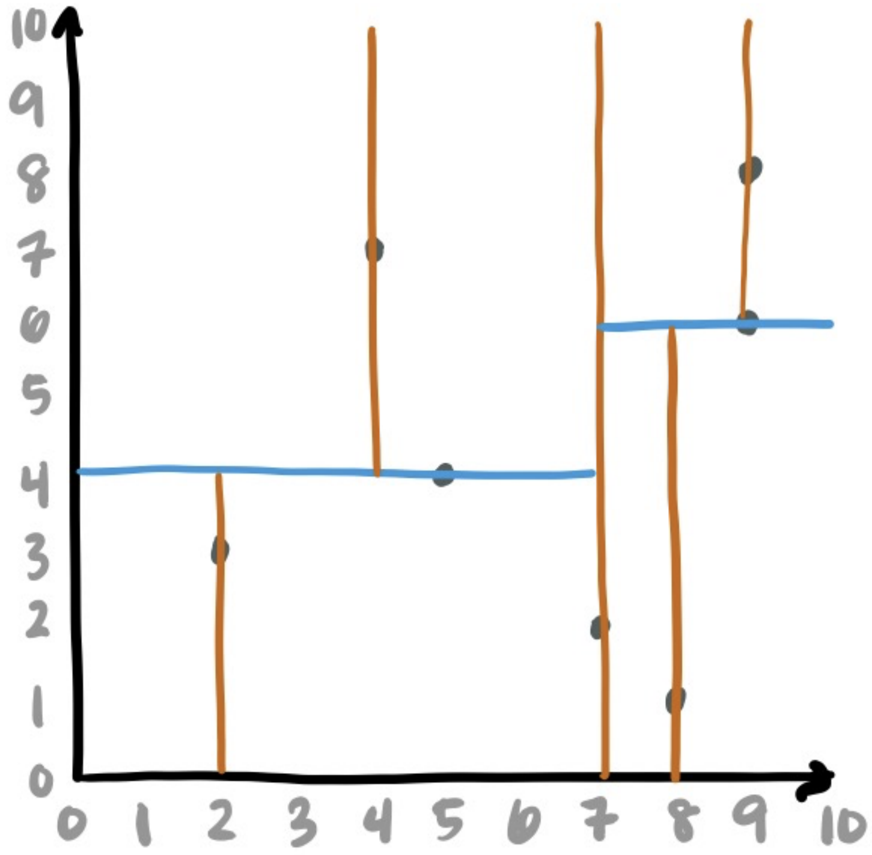


Brute Force : Query distance of each point pi from (x1, y1) and pick closest

Time Complexity : O(kn)

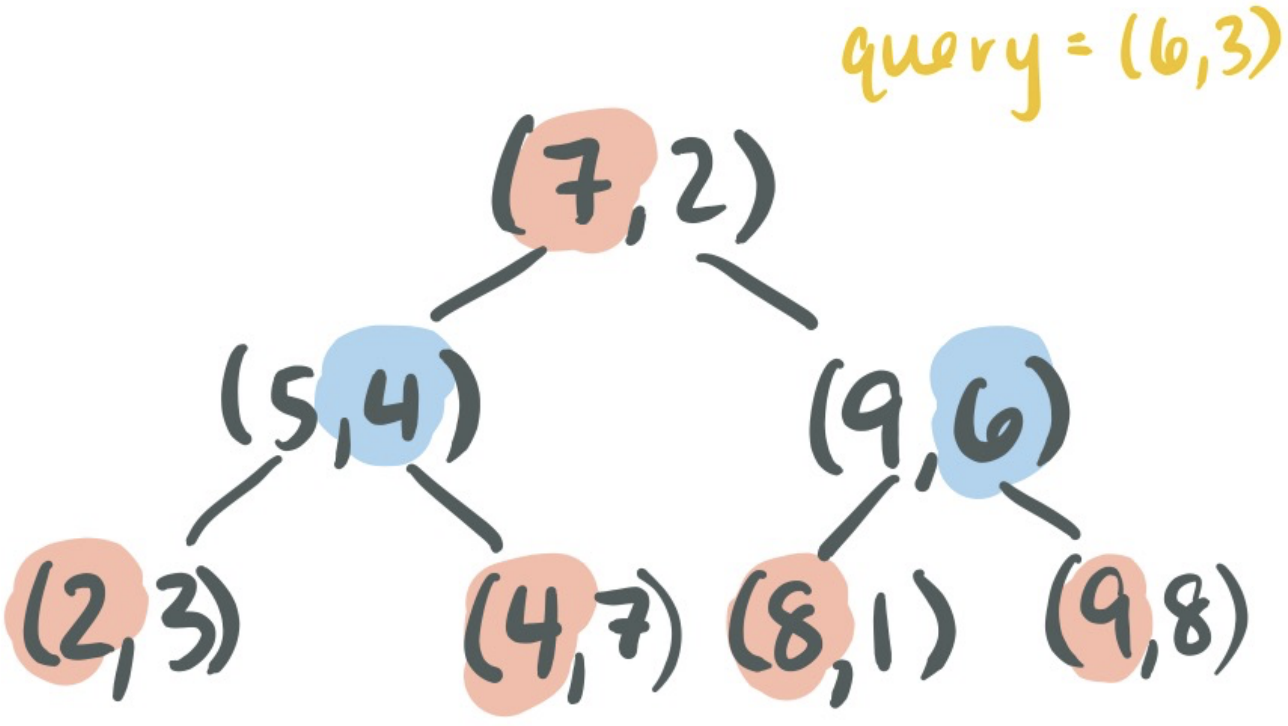
Nearest Neighbor: k-d tree

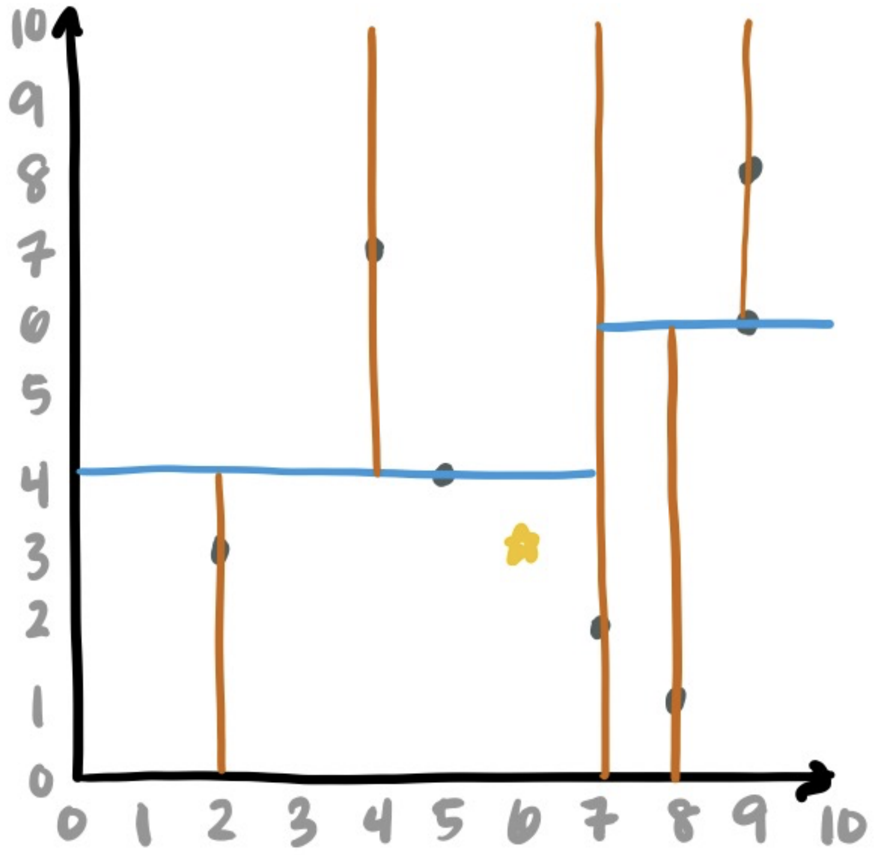




Nearest Neighbor: k-d tree

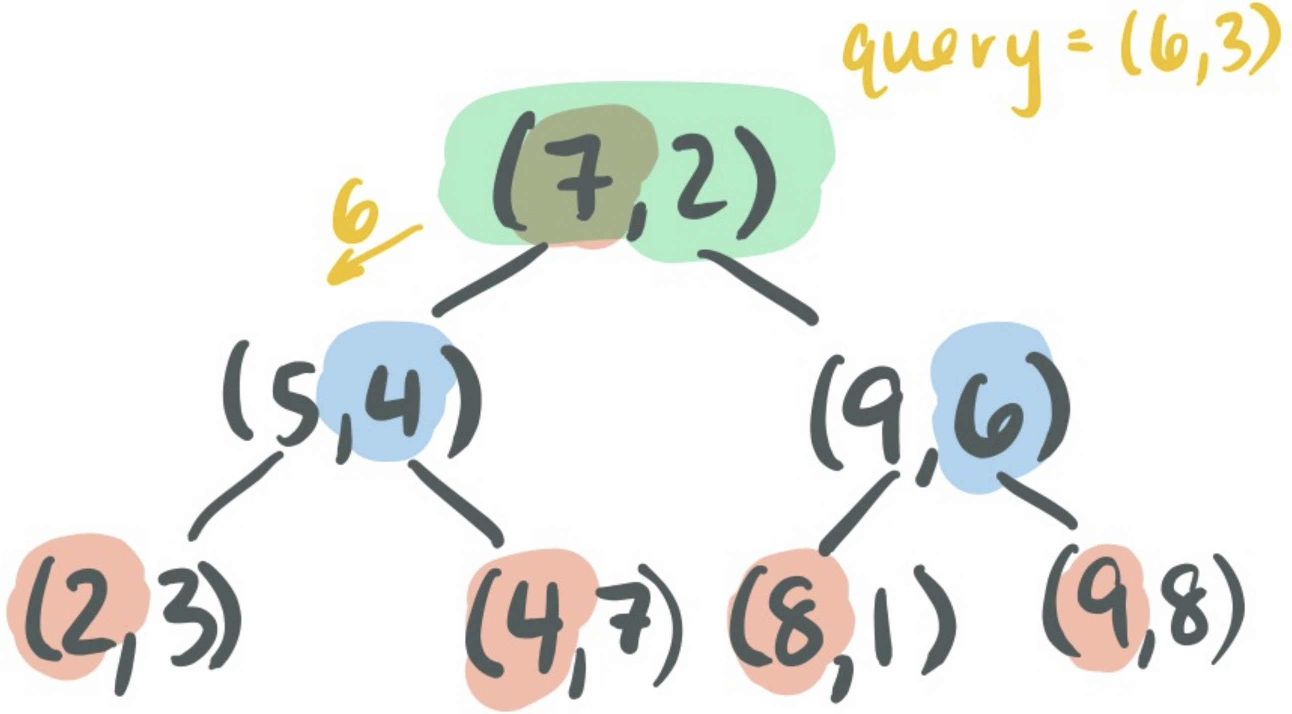
When querying a k-d tree, it acts like a BST* at first…

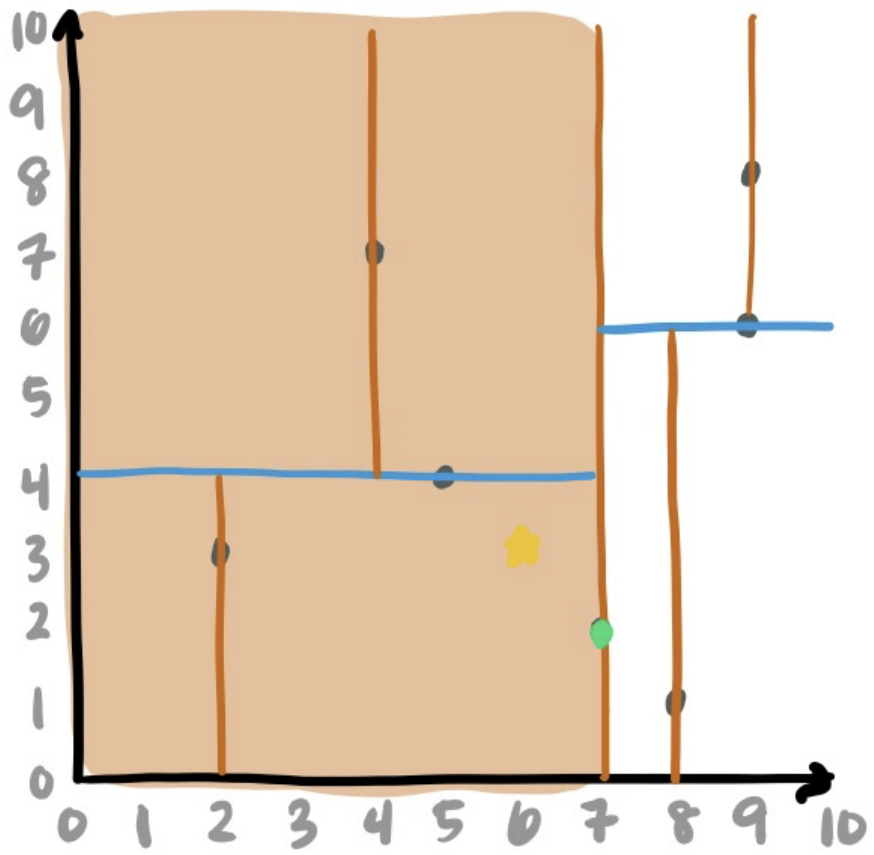




Nearest Neighbor: k-d tree

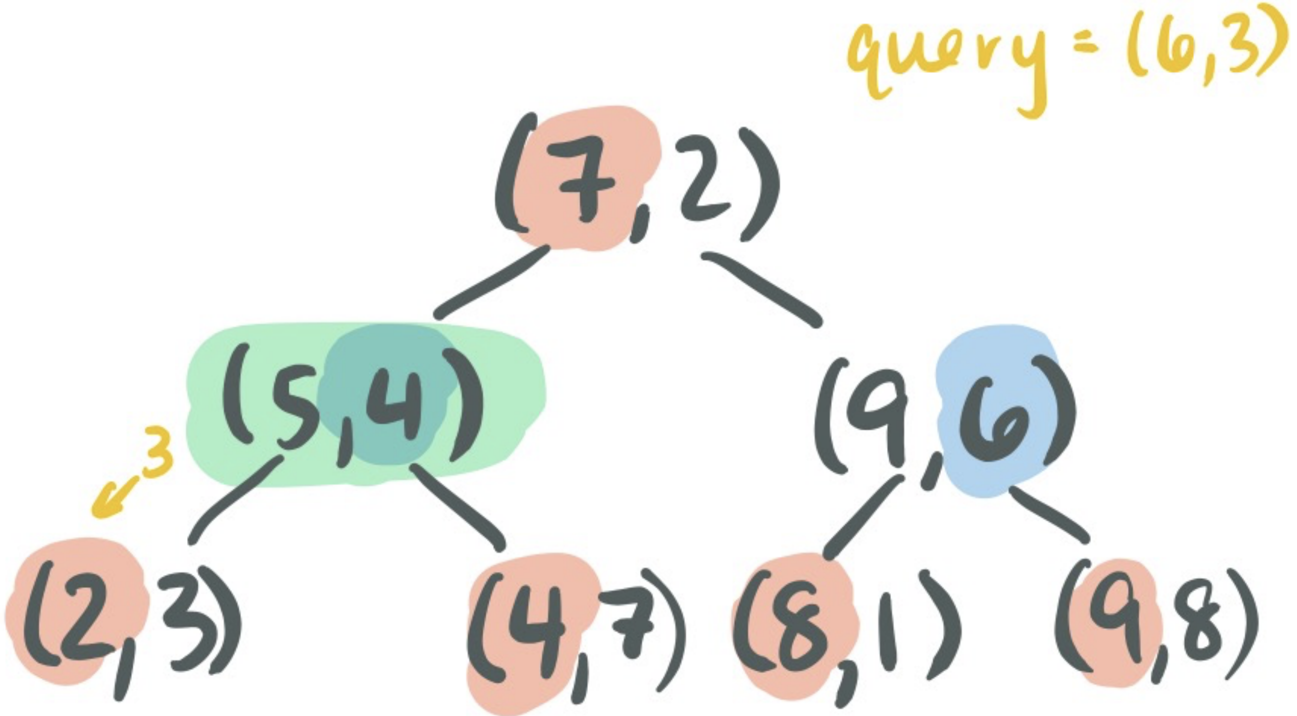
When querying a k-d tree, it acts like a BST* at first…

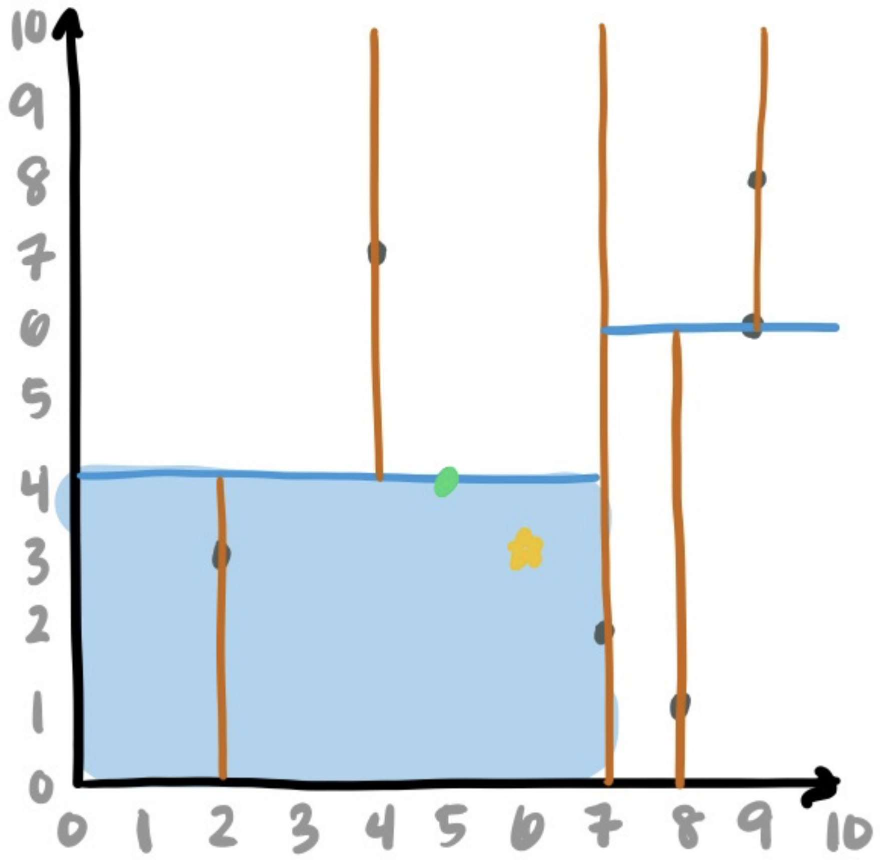




Nearest Neighbor: k-d tree

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

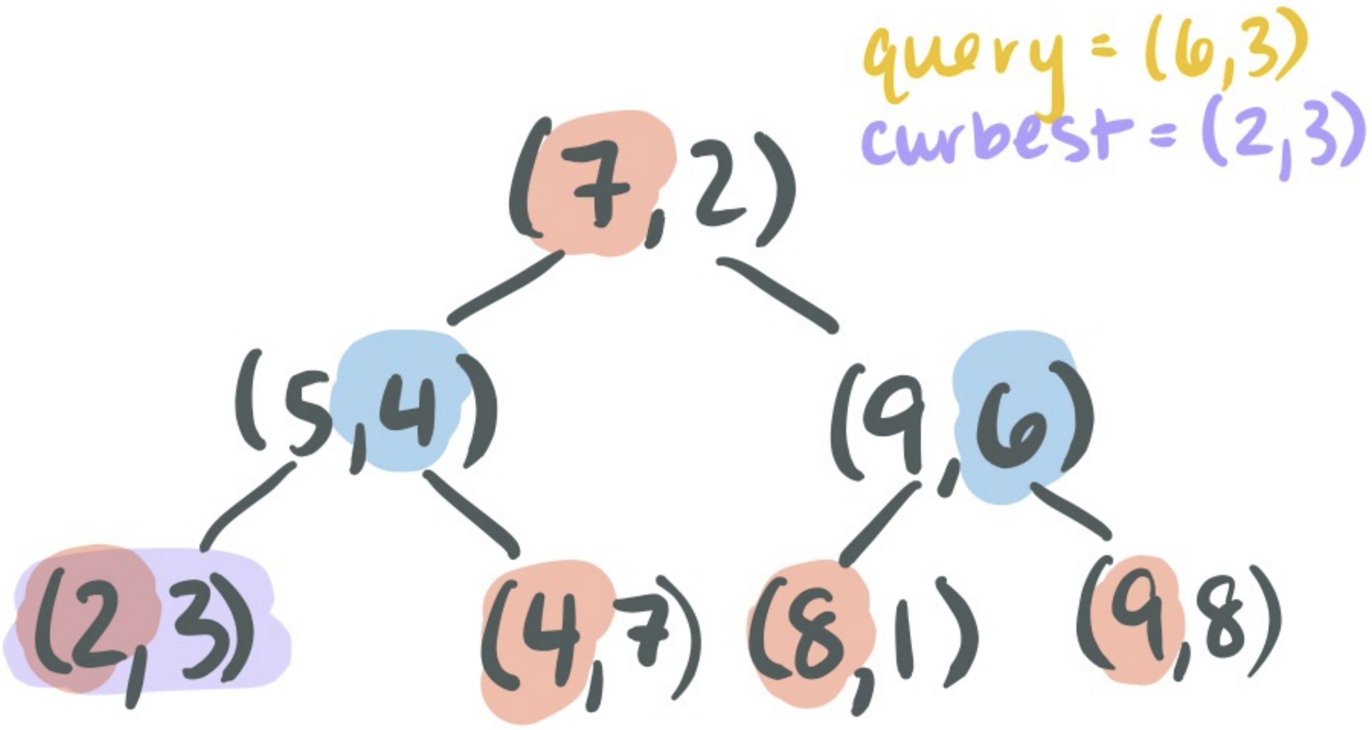


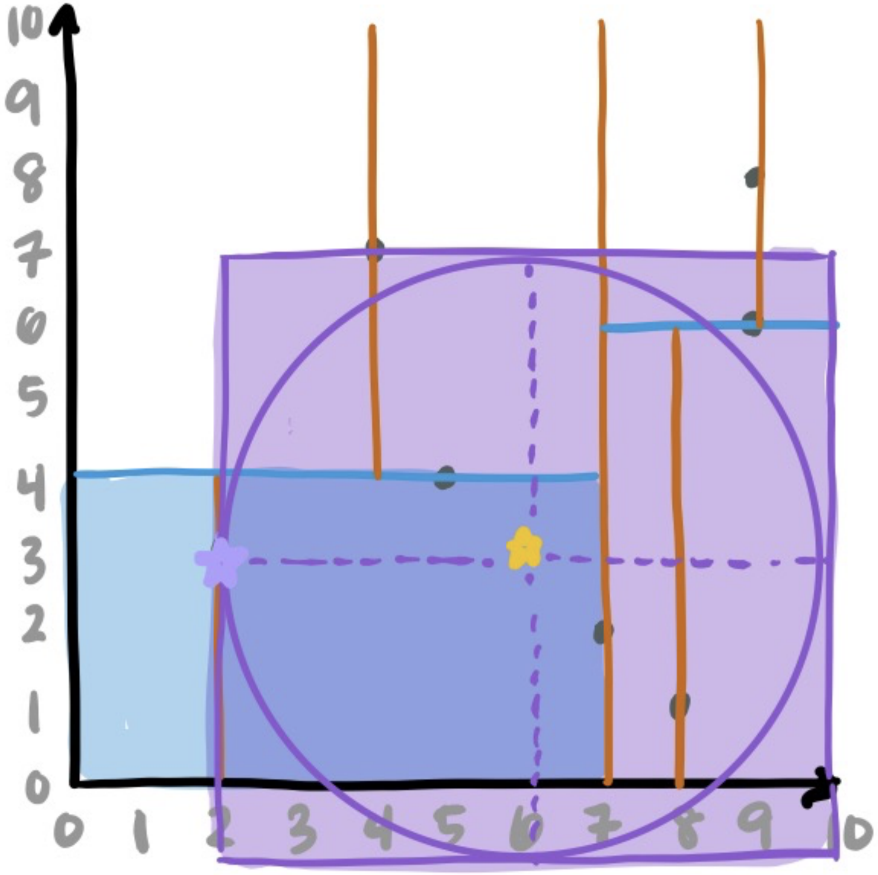


Nearest Neighbor: k-d tree

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

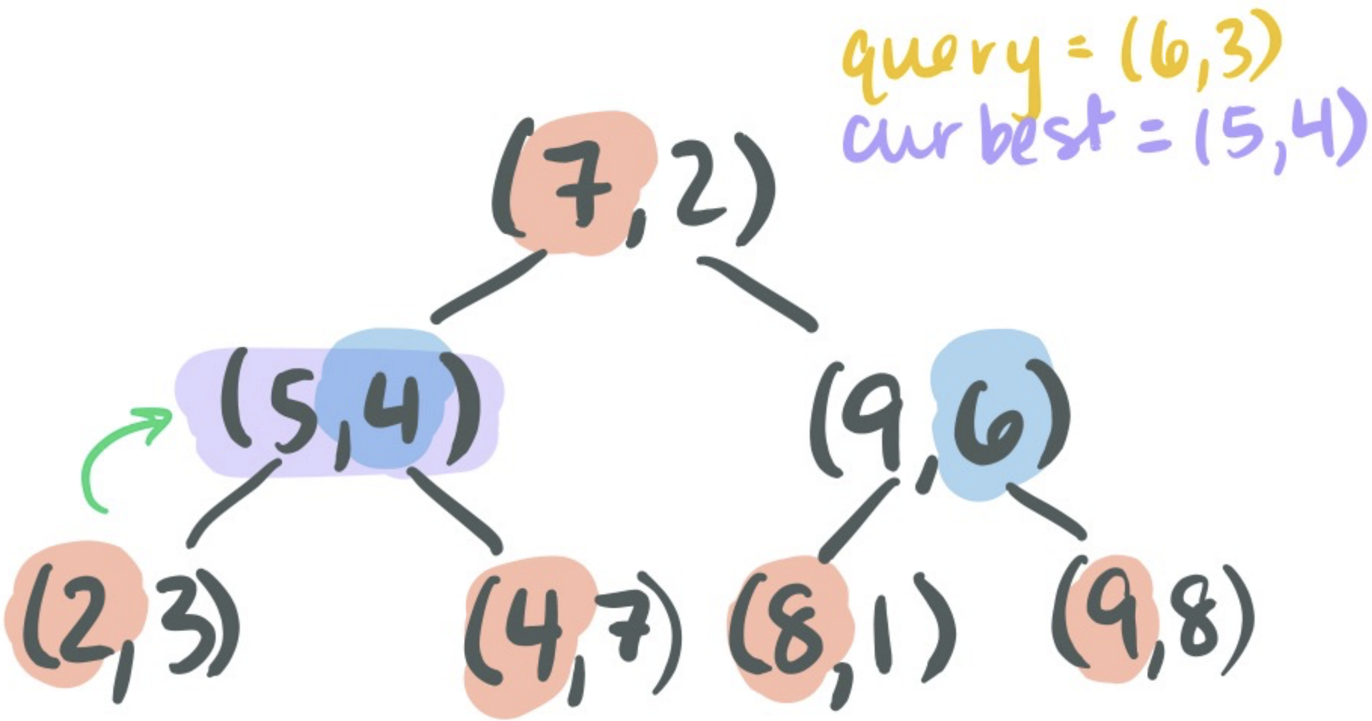
… But if we dont find exact match, have to find nearest neighbor

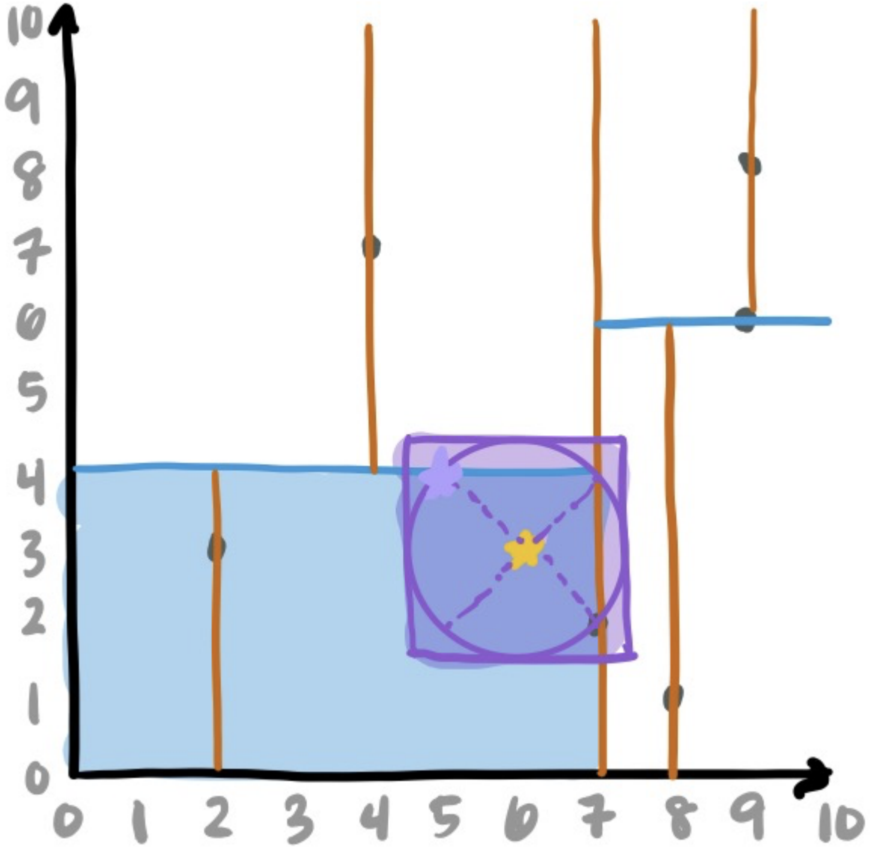




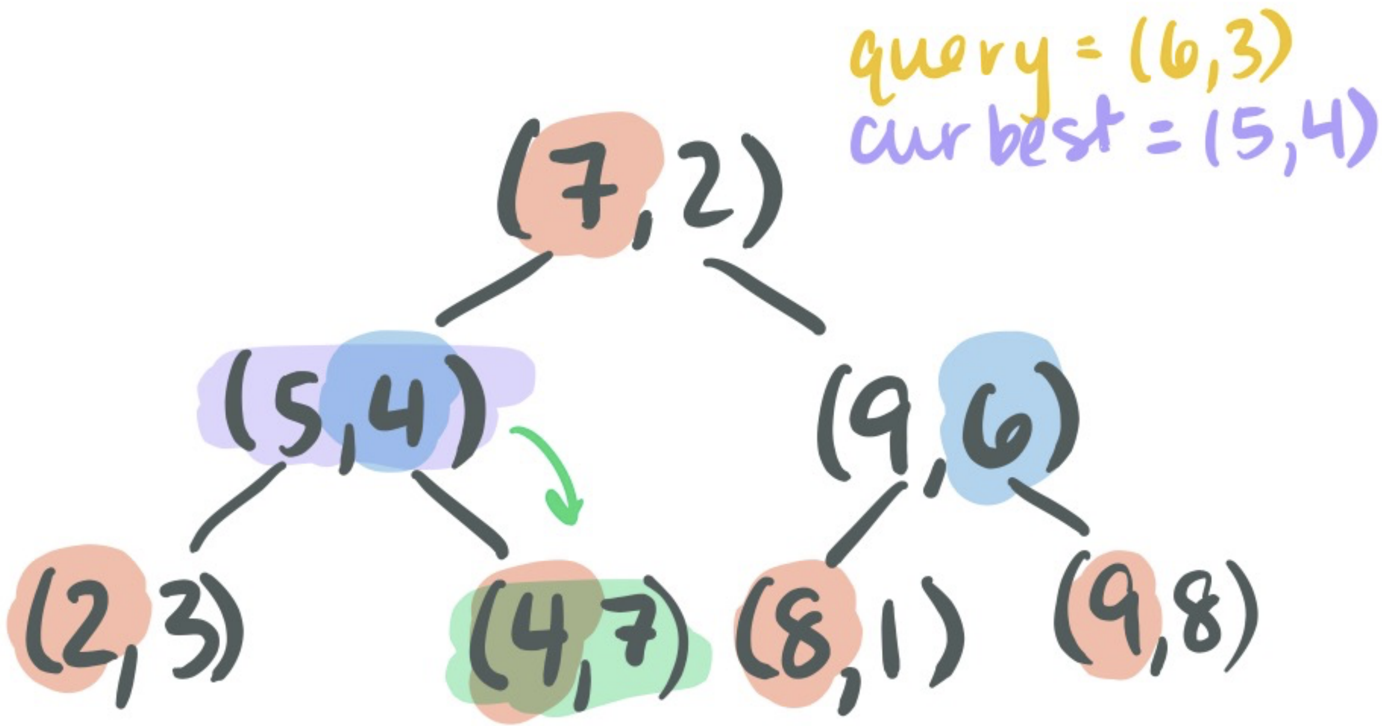
Nearest Neighbor: k-d tree

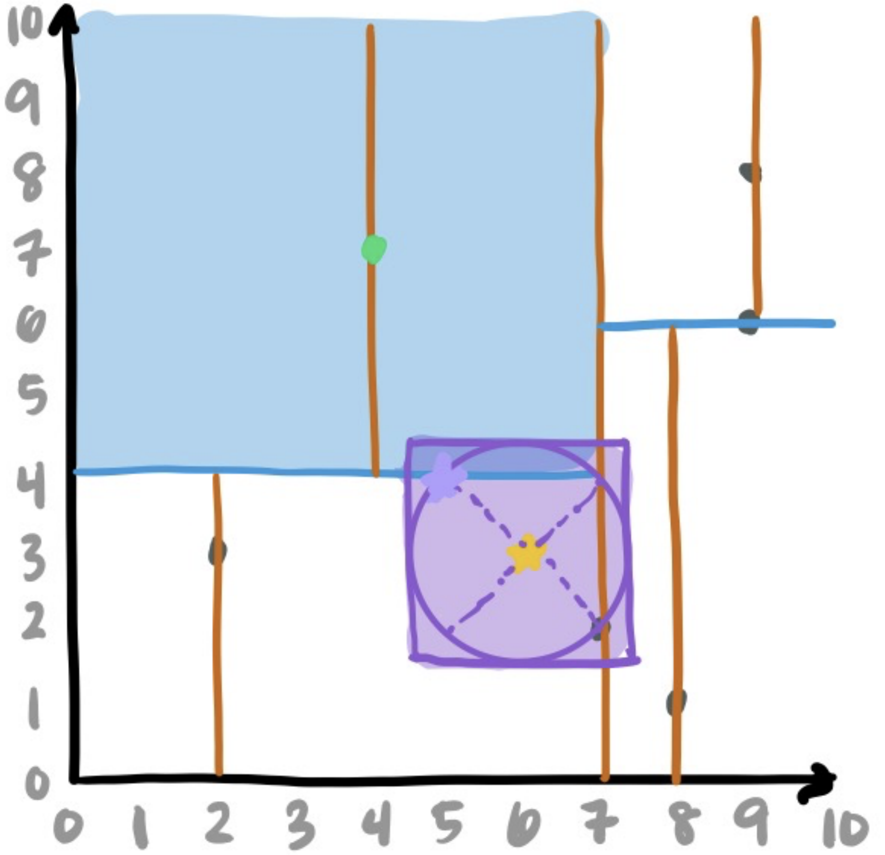
Backtracking: start recursing backwards -- store "best" possibility as you trace back





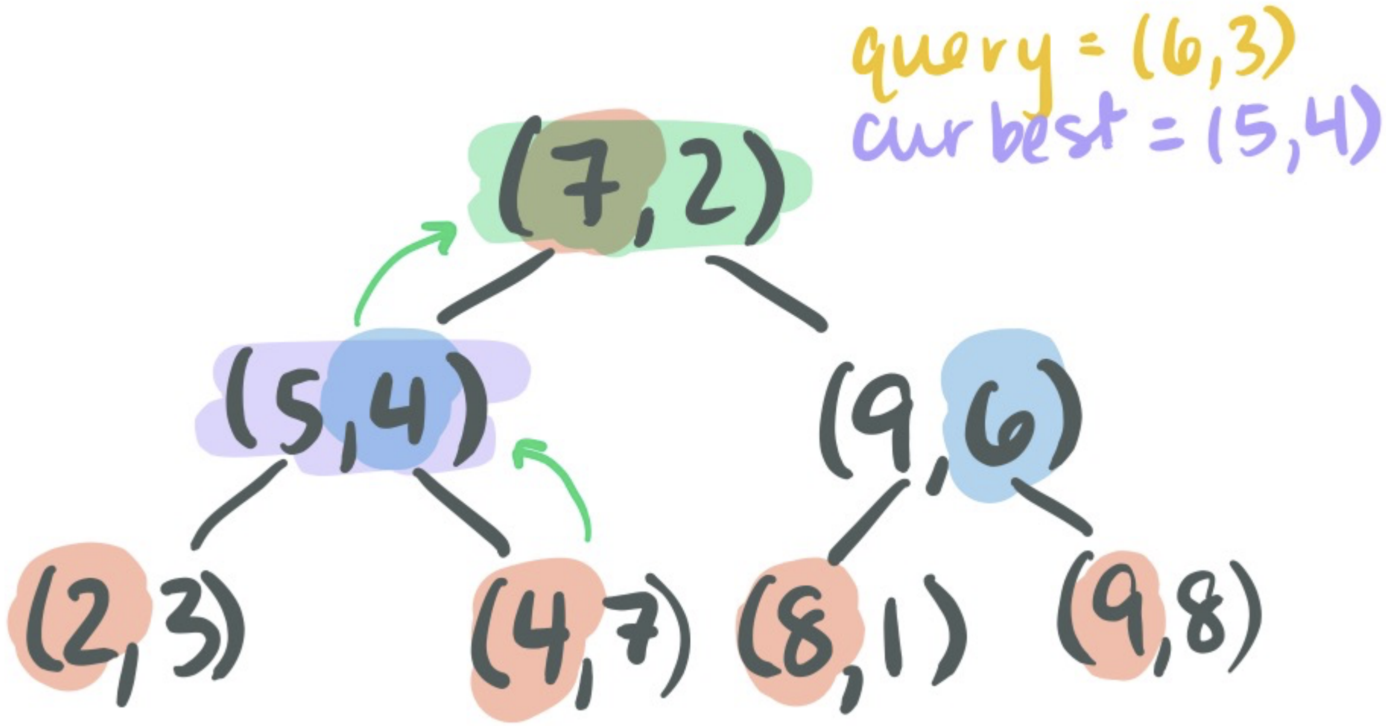
Nearest Neighbor: k-d tree

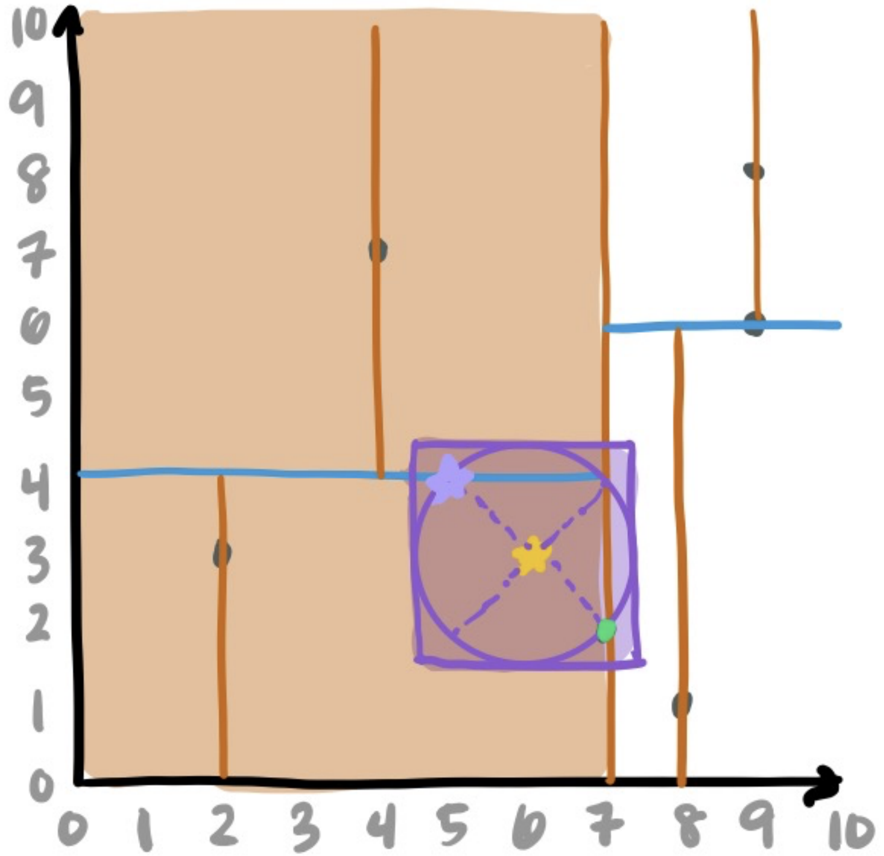




Nearest Neighbor: k-d tree

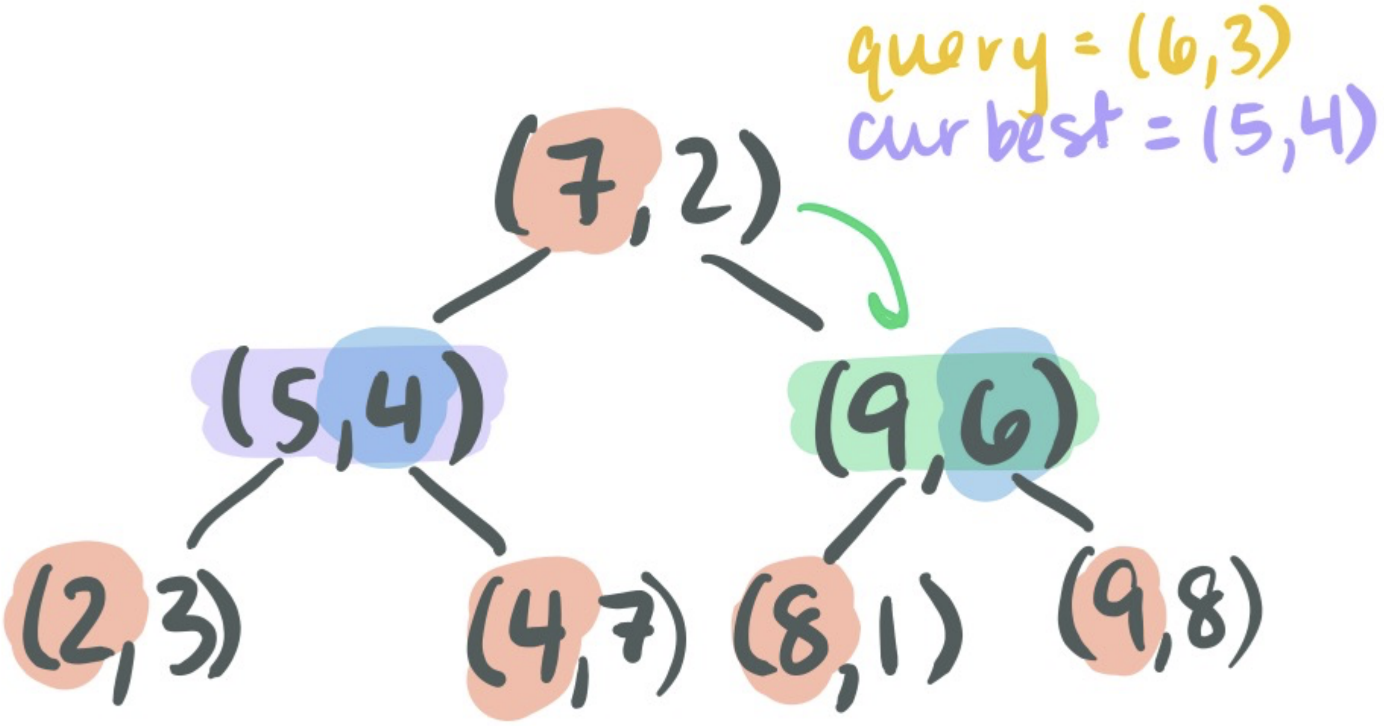
On ties, use smallerDimVal to determine which point remains curBest

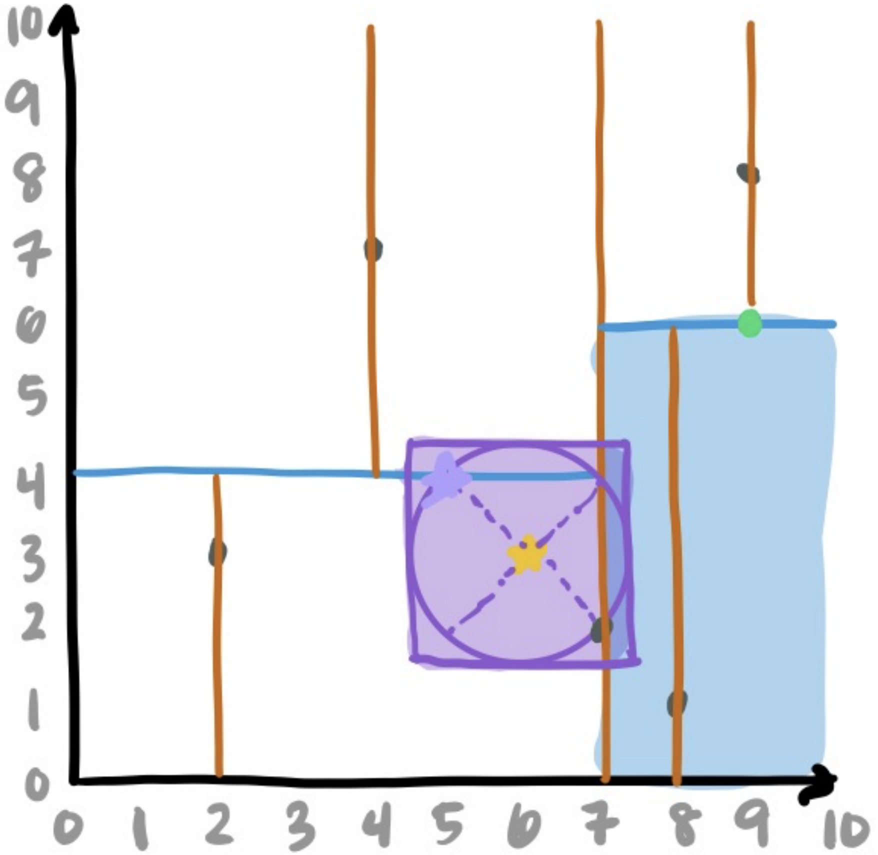




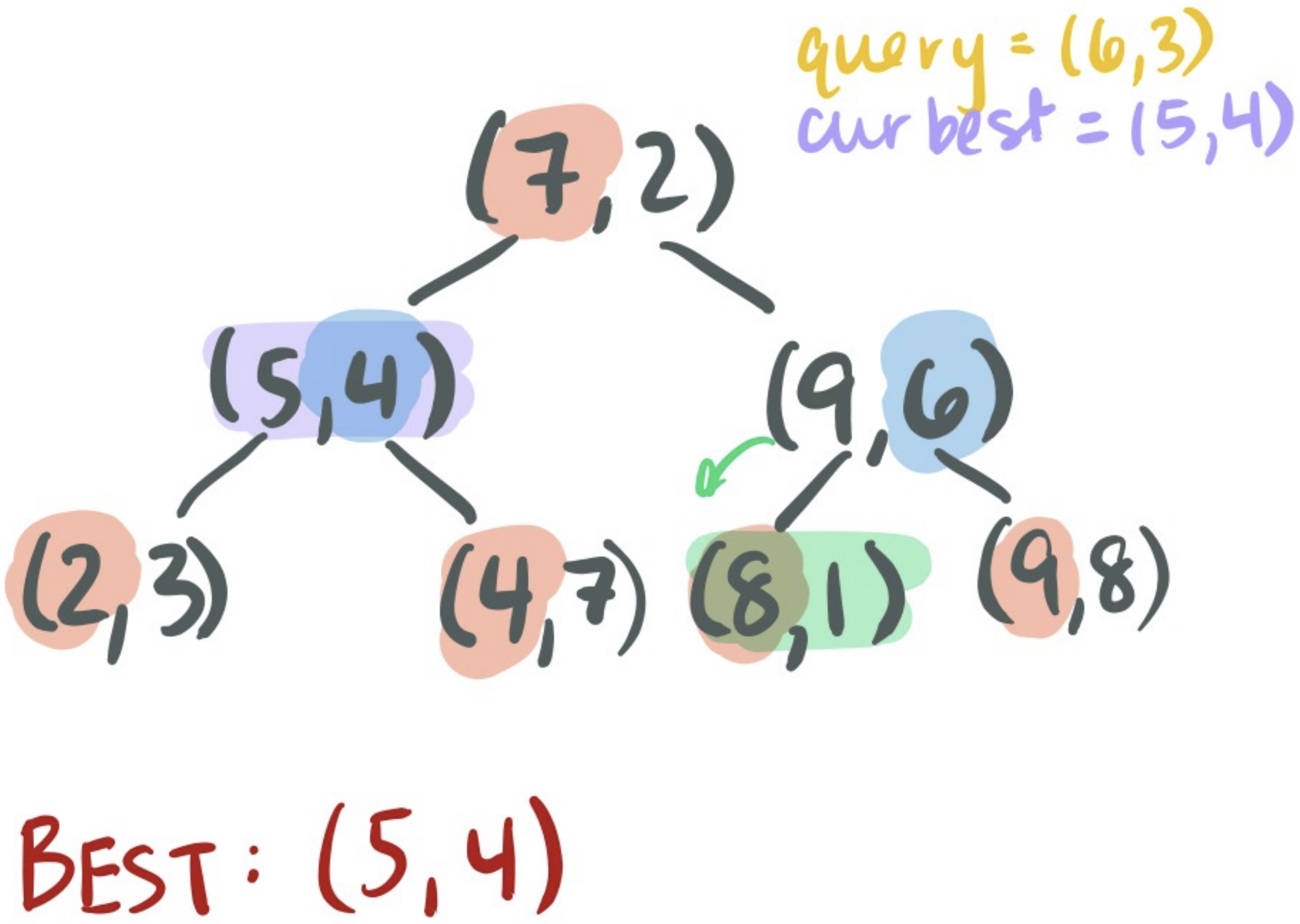
Nearest Neighbor: k-d tree

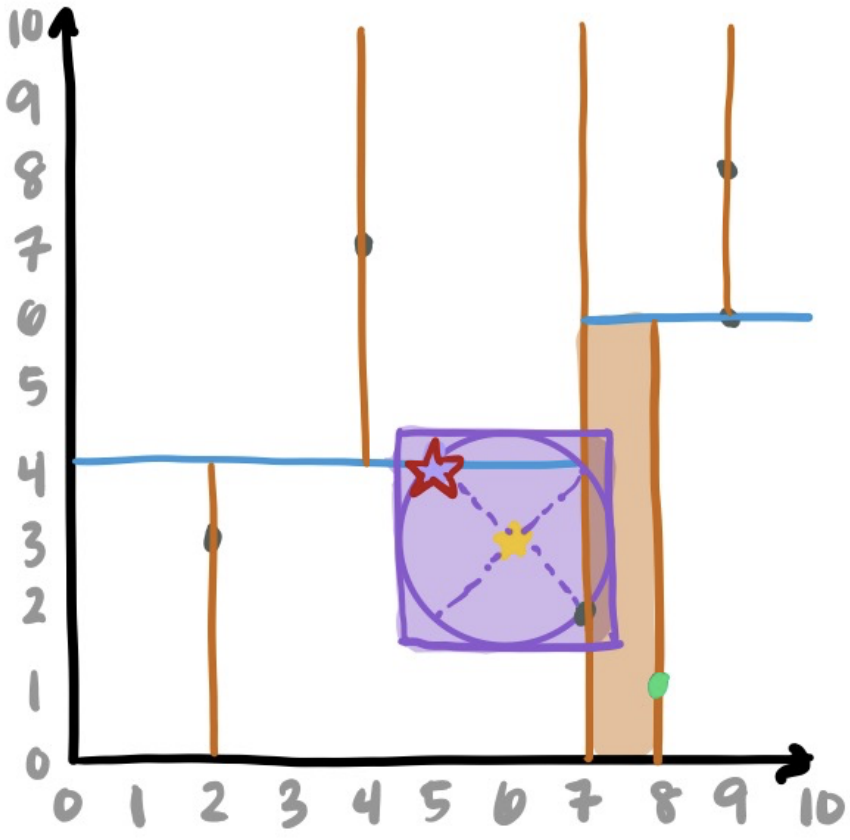
Why do we need to explore this subtree?





Nearest Neighbor: k-d tree





Kd tree : Pros and Cons

	KD tree	Comments
Build	**O(n log n)**	Worth paying this cost if we anticipate many queries
Range Search	O(n^(1 - 1/k) + m)	Good for low dimensions Curse of Dimensionality - Bad as k increases
Nearest Neighbor Search	O(log n) : Average Case O(n) : Worst case	Depends on distribution of data
Insert/Remove data	O(log n) : Average Case O(n) : Worst case	Depends on distribution of data

m : #outputs

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

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

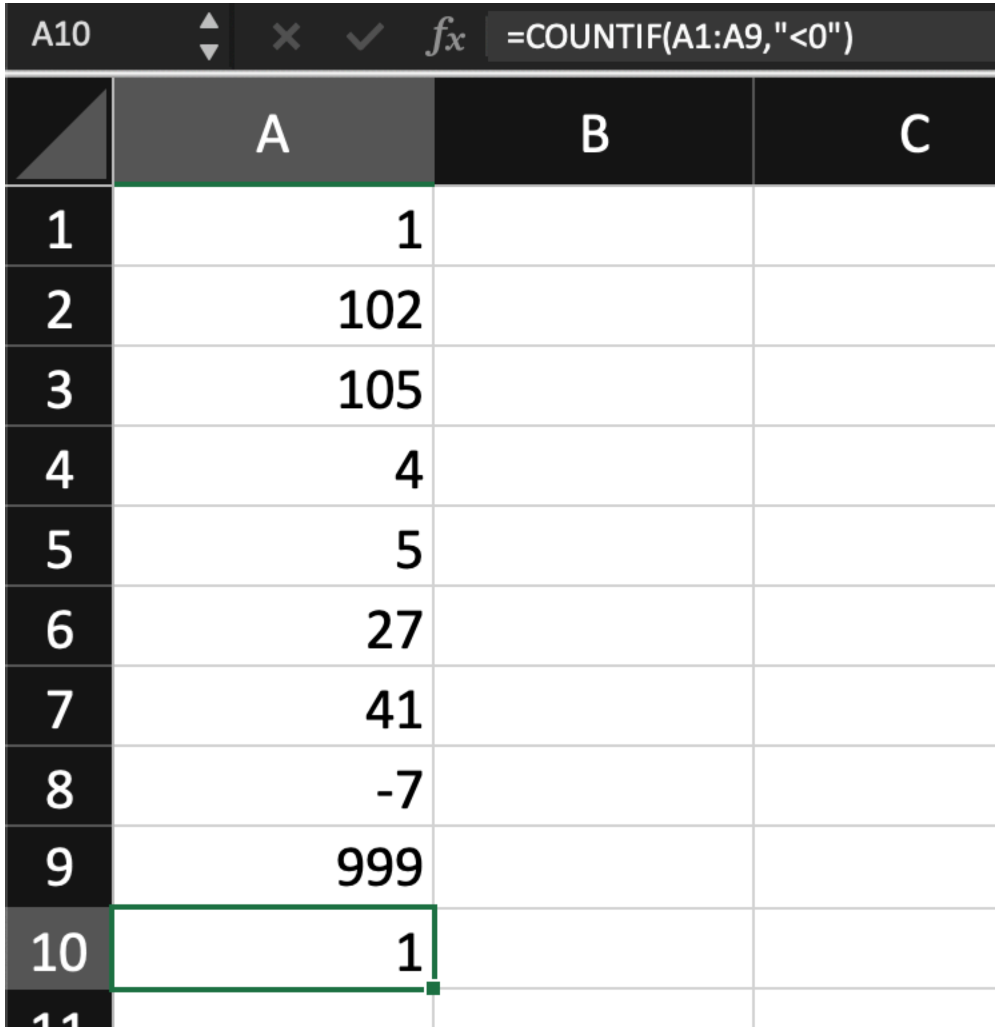
```
template <typename RandIter, typename Comparator>
void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
{
    /**
     * @todo Implement this function!
     */

  }
```

2. Review, understand, and use lambda functions

Functions as arguments

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



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();
```

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

```
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