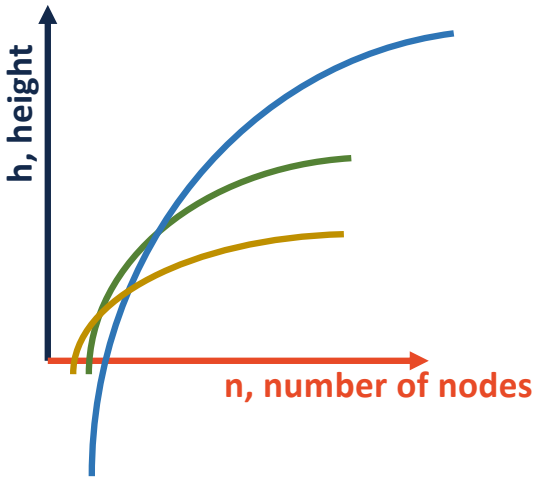
CS 225

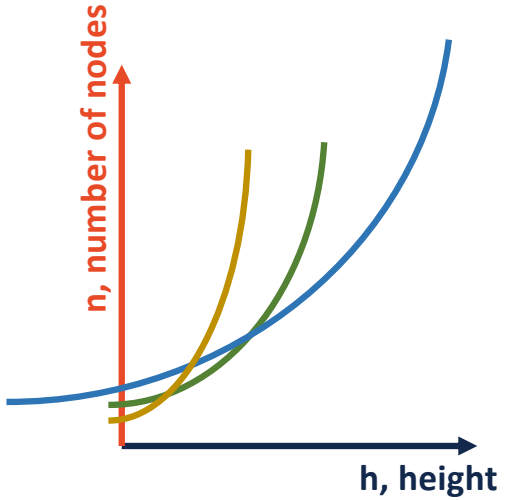
Data Structures

February 21 – BBST Wrap and Lambda

G Carl Evans

AVL Tree Analysis





- The number of nodes in the tree, $\mathbf{f^{-1}(h)}$, will always be <u>greater than</u> $\mathbf{c \times g^{-1}(h)}$ for all values where $\mathbf{n > k}$.

Prove a Theorem

V. Using a proof by induction, we have shown that:

…and inverting:

Summary of Balanced BST

Red-Black Trees

- Max height: 2 * lg(n)
- Constant number of rotations on insert, remove, and find

AVL Trees

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

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 & )
```

Summary of Balanced BST

Pros:

- Running Time:

  - Improvement Over:

- Great for specific applications:

Summary of Balanced BST

Cons:

- Running Time:

- In-memory Requirement:

Every Container Data Structure So Far

	Unsorted Array	Sorted Array	Unsorted List	Sorted List	Binary Tree	BST	AVL
Find							
Insert							
Remove							
Traverse							

Iterators Types

- Forward
- Bidirectional
- Random Access

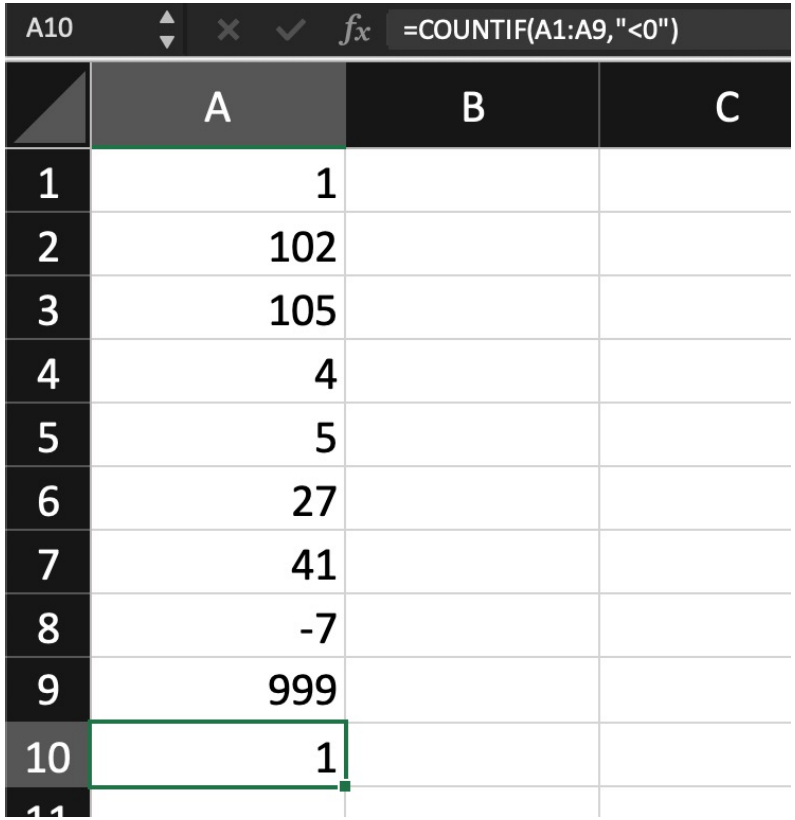
Random Access Iterators

- Big Steps

- Distance

Functions As Data

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



COUNTIF in C++

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  }
```

Ways to use Countif()

main.cpp

```
12 bool isNegative(int num) { return (num < 0); }
13
14 class IsNegative {
15 public:
16     bool operator() (int num) { return (num < 0); }
17 };
18
19 int main() {
20   std::vector<int> numbers = {1, 102, 105, 4, 5, 27, 41, -7, 999};
21
22   auto isnegl = [](int num) { return (num < 0); };
23   auto isnegfp = isNegative;
24   auto isnegfuctor = IsNegative();
25
26   std::cout << "There are " << Countif(numbers.begin(), numbers.end(), _______)
27     << " negative numbers" << std::endl;
```

Lambdas in C++ (functions with no name)

[ ]( ){ }

Power of the lambda

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    std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)
36      << " big numbers" << std::endl;
37  }
38
```

Range-based Searches

Balanced BSTs are useful structures for range-based and nearest-neighbor searches.

Q: Consider points in 1D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

…what points fall in [11, 42]?

Ex:

Range-based Searches

Q: Consider points in 1D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

…what points fall in [11, 42]?

Ex:

Red-Black Trees in C++

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

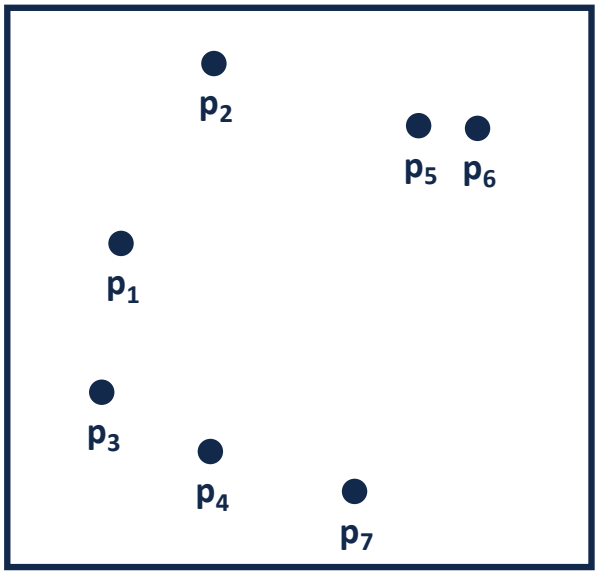
Range-based Searches

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

Q: What points are in the rectangle:

$\mathbf{[\ (x_1, y_1),\ (x_2, y_2)\ ]?}$

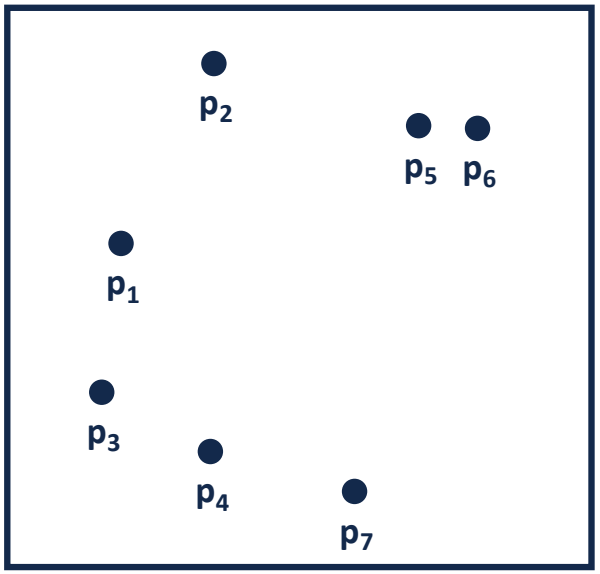
Q: What is the nearest point to $\mathbf{(x_1, y_1)}$?



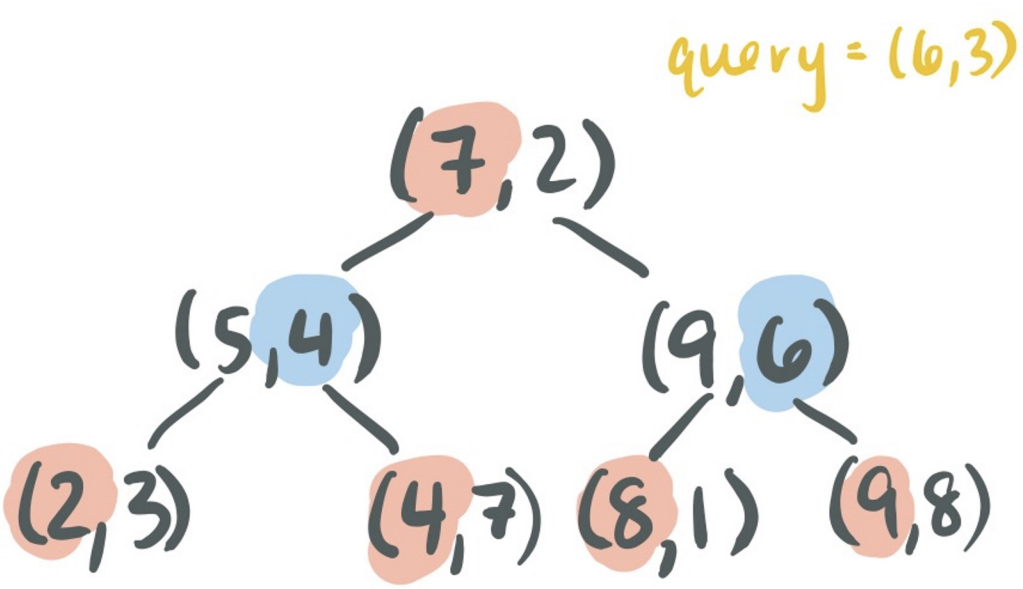
Range-based Searches

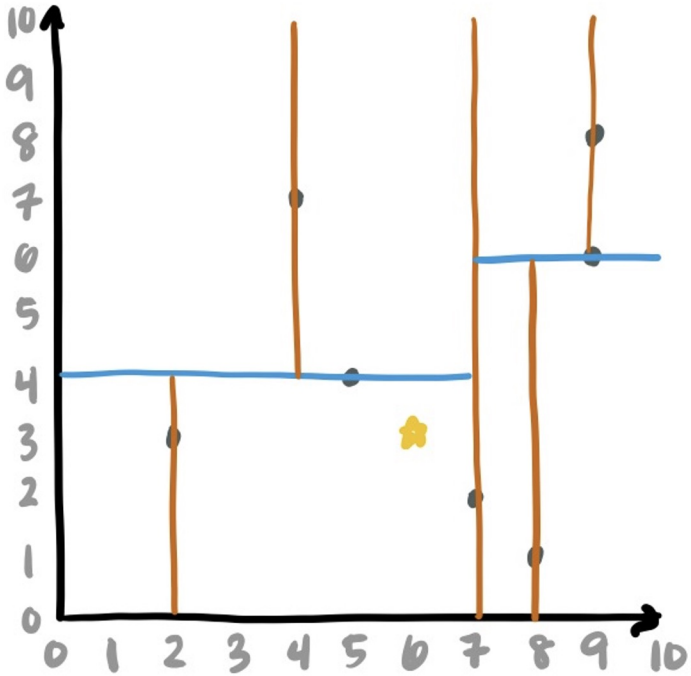
Consider points in 2D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

Tree construction:

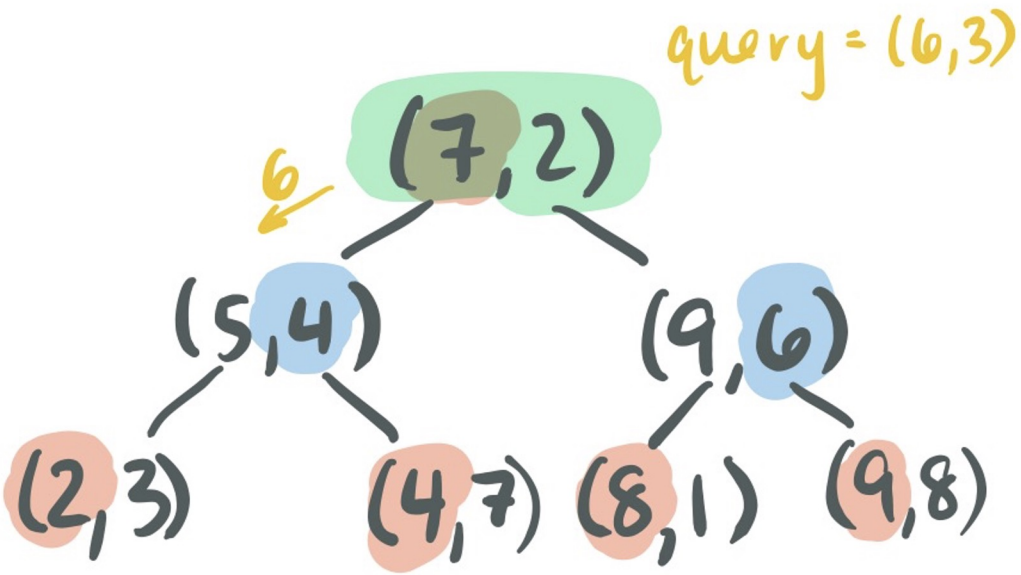


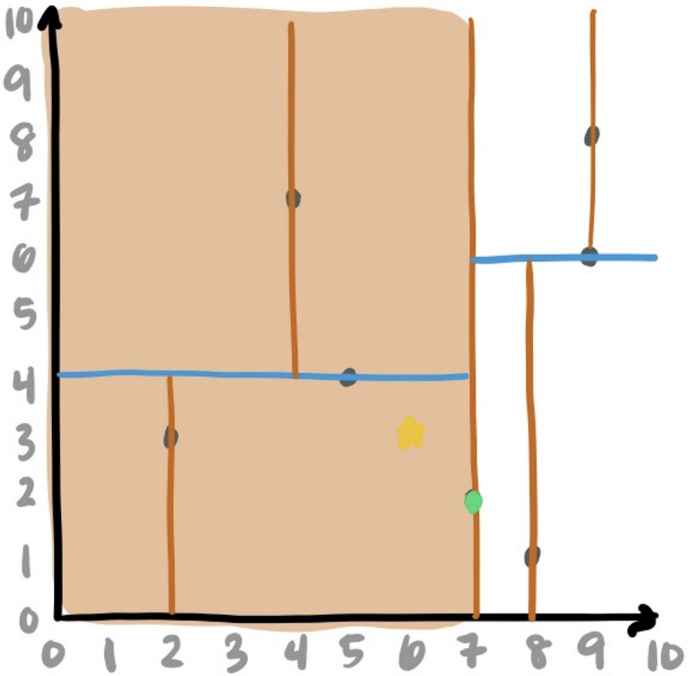
Nearest Neighbor – k-d Tree



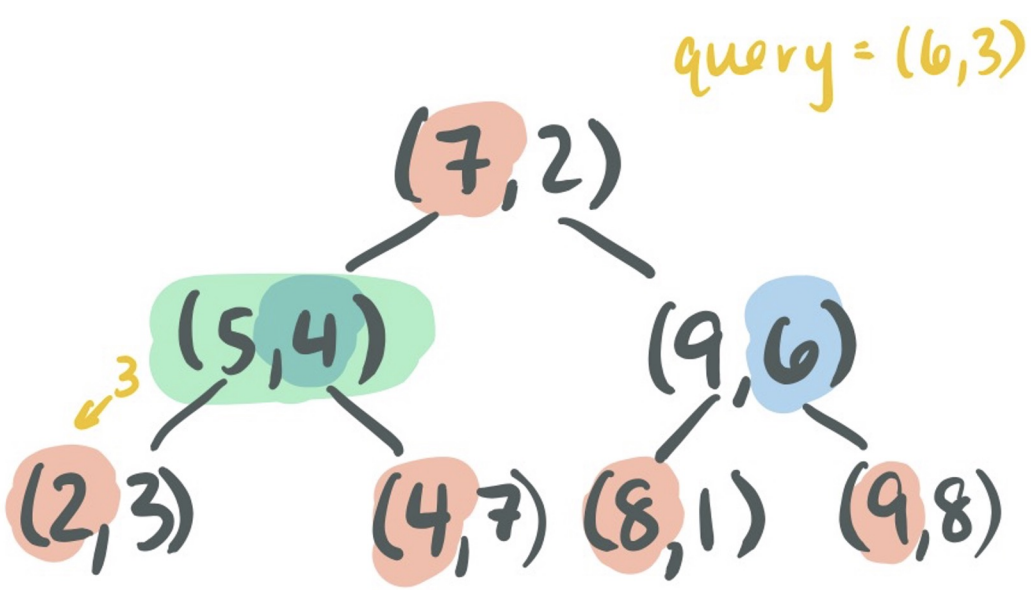


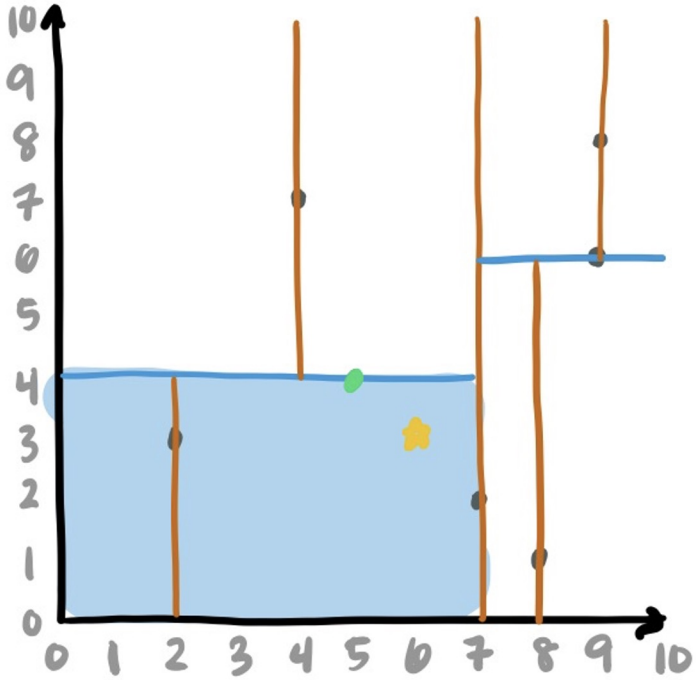
Nearest Neighbor - demo



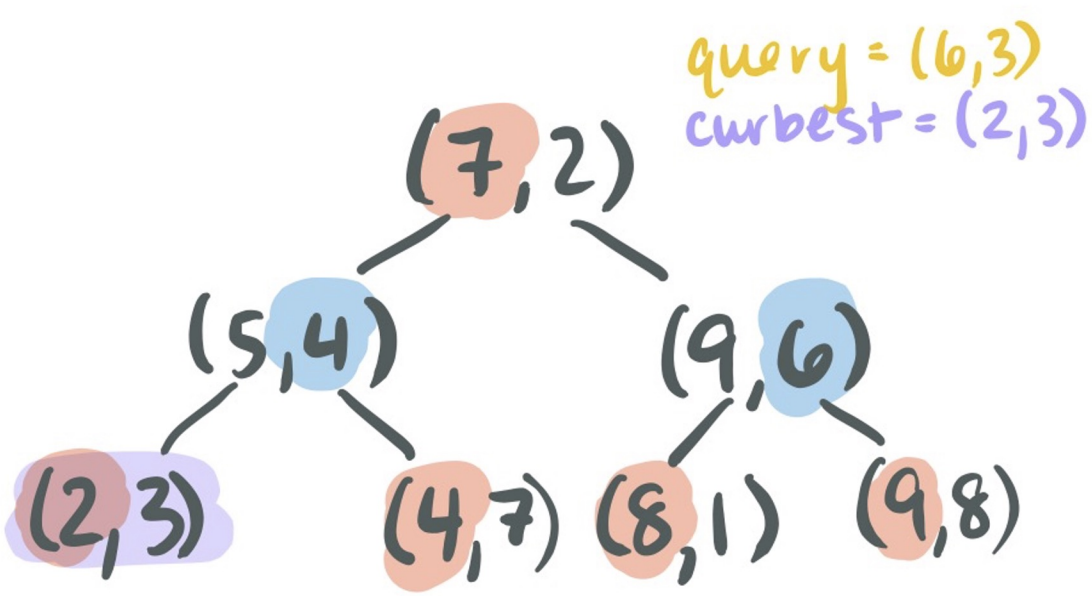


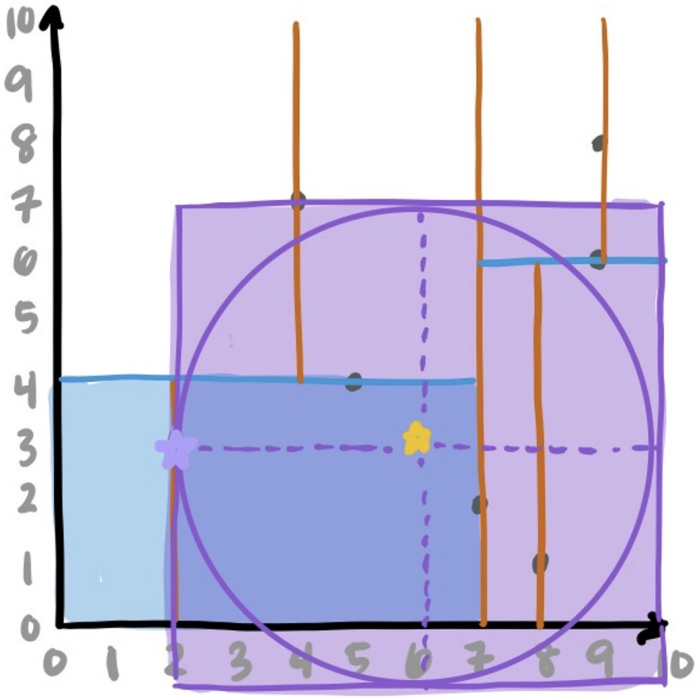
Nearest Neighbor - demo





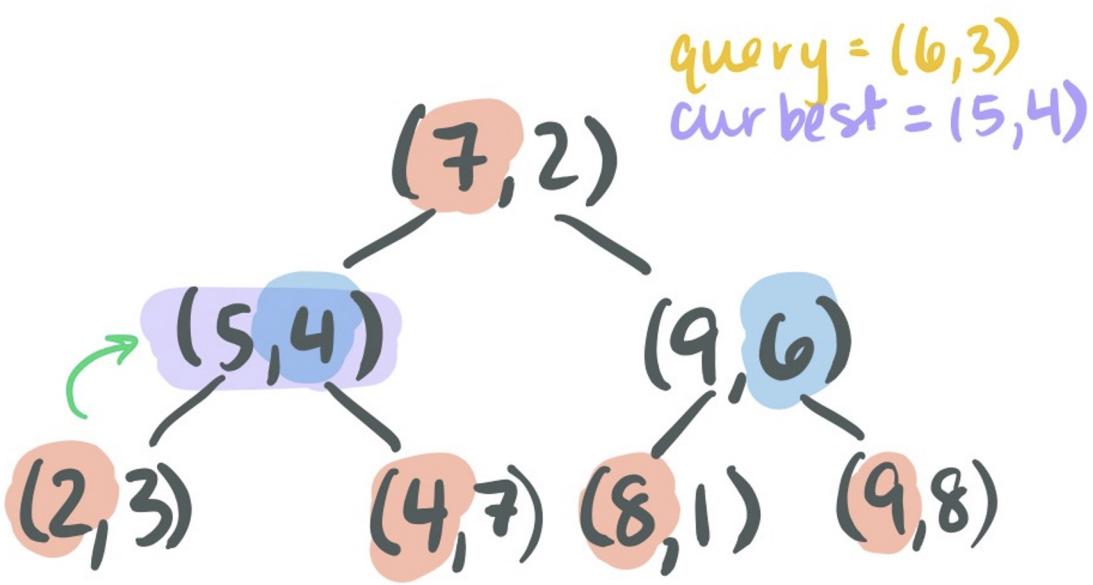
Nearest Neighbor - demo

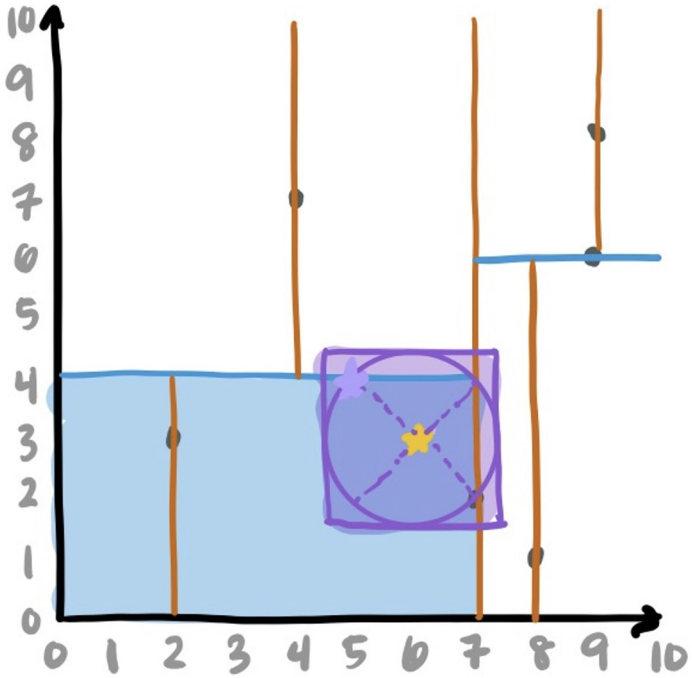




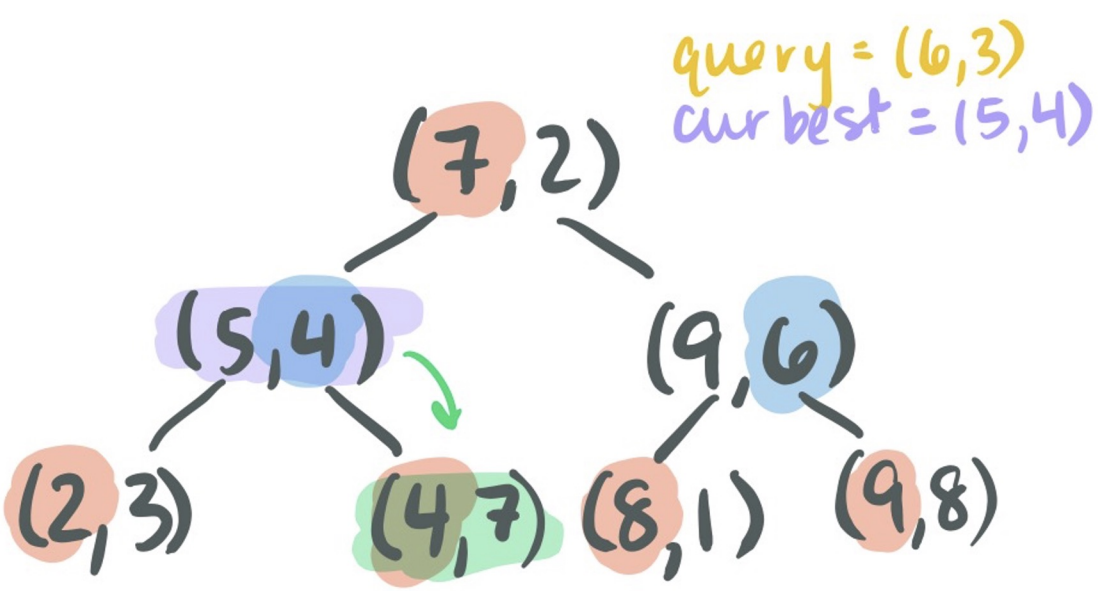
Nearest Neighbor - demo

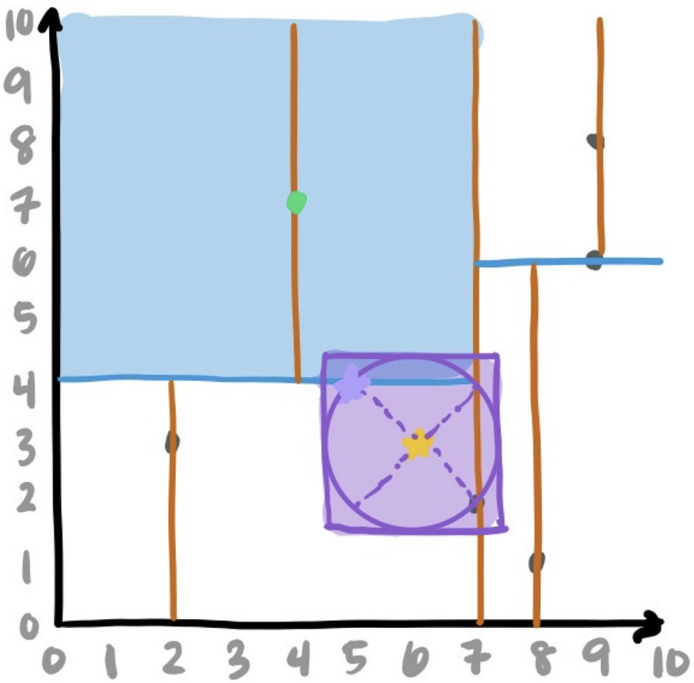
Backtracking: start recursing backwards -- store “best” possibility as you trace back





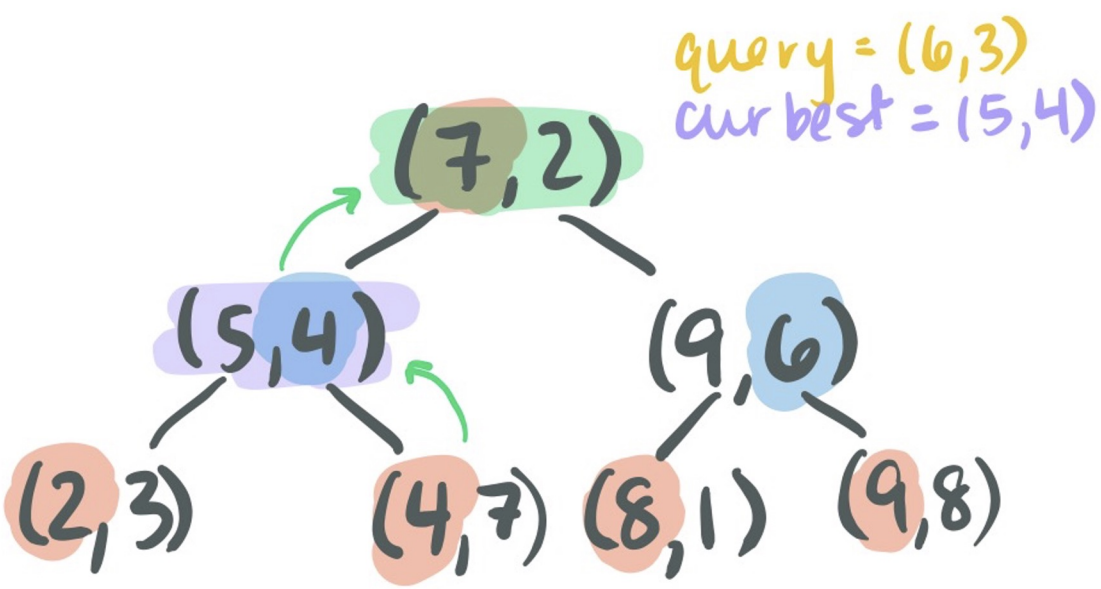
Nearest Neighbor - demo

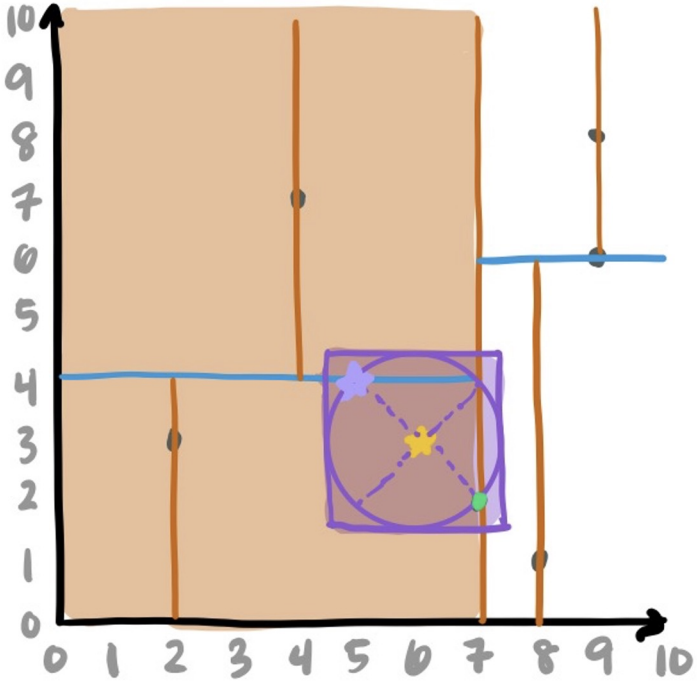




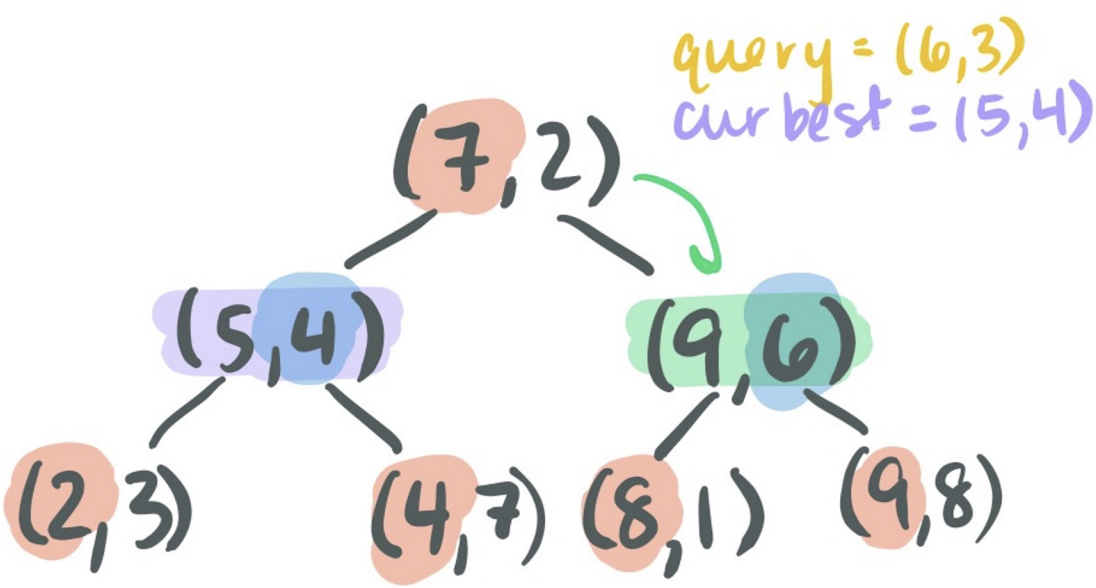
Nearest Neighbor - demo

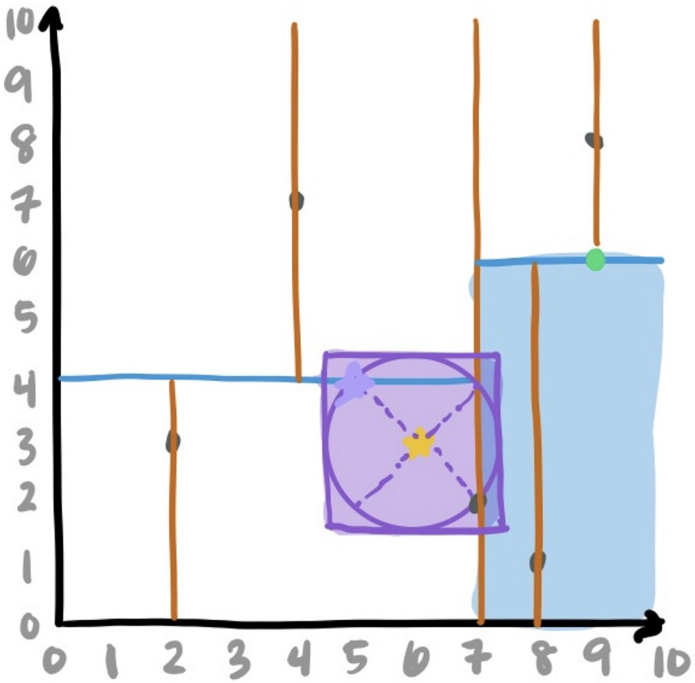
On ties, use `smallerDimVal` to determine which point remains `curBest`



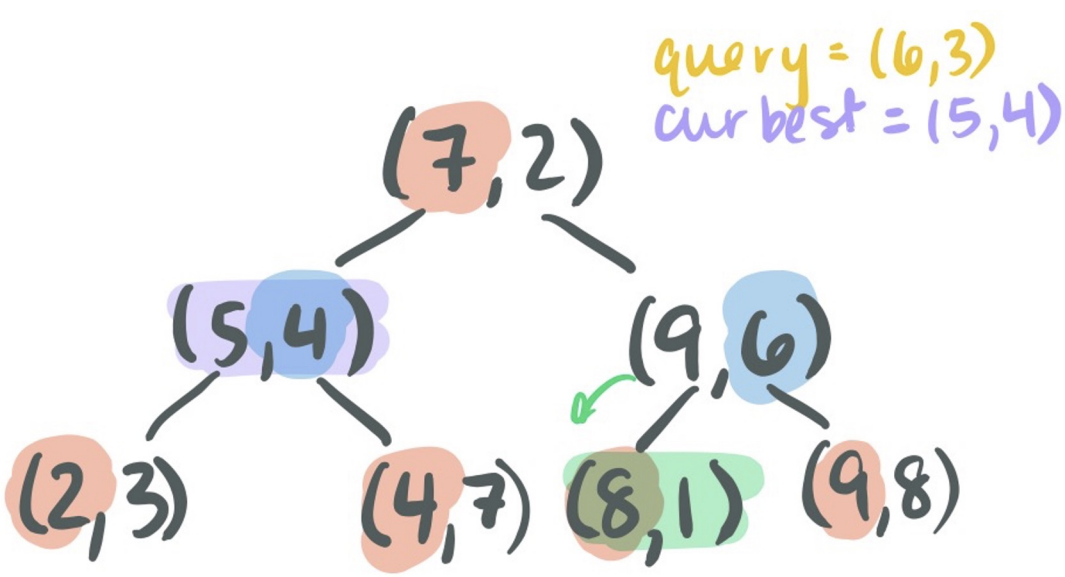


Nearest Neighbor - demo





Nearest Neighbor - demo



BEST: (5,4)