CS 225

Data Structures

February 23 – BBST Range Search

G Carl Evans

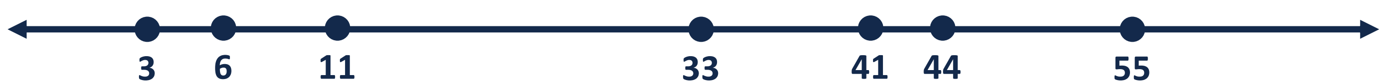
Range-based Searches

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:

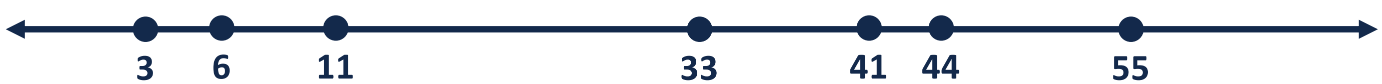


Range-based Searches

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++

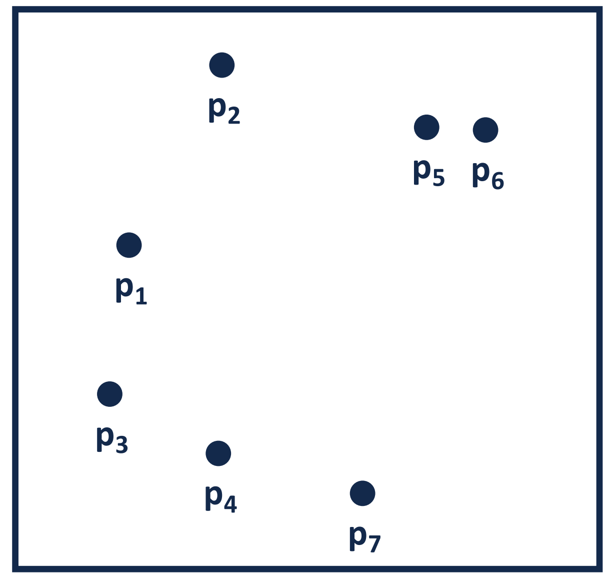
```
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```

Range-based Searches

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

Q: What points are in the rectangle:
$[\ (x_1, y_1),\ (x_2, y_2)\ ]$?

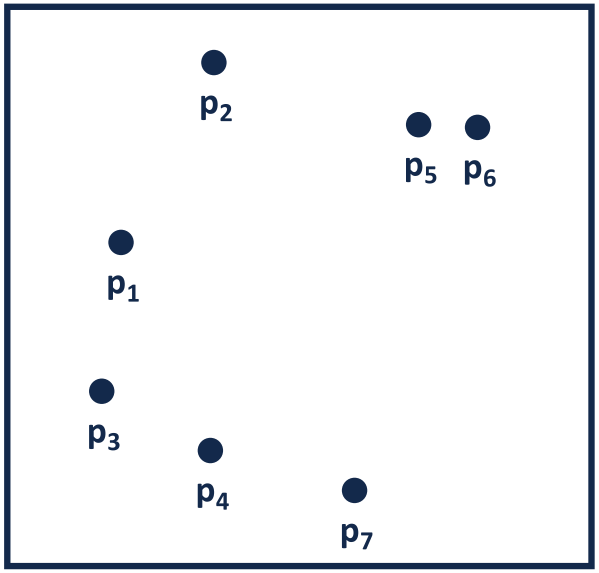
Q: What is the nearest point to $\mathbf{(x_1, y_1)}$?



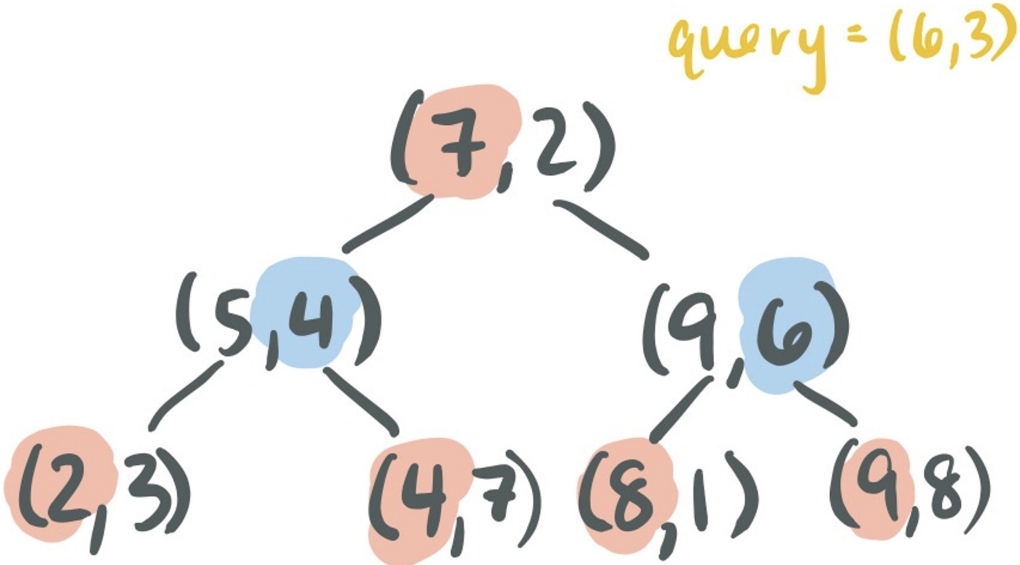
Range-based Searches

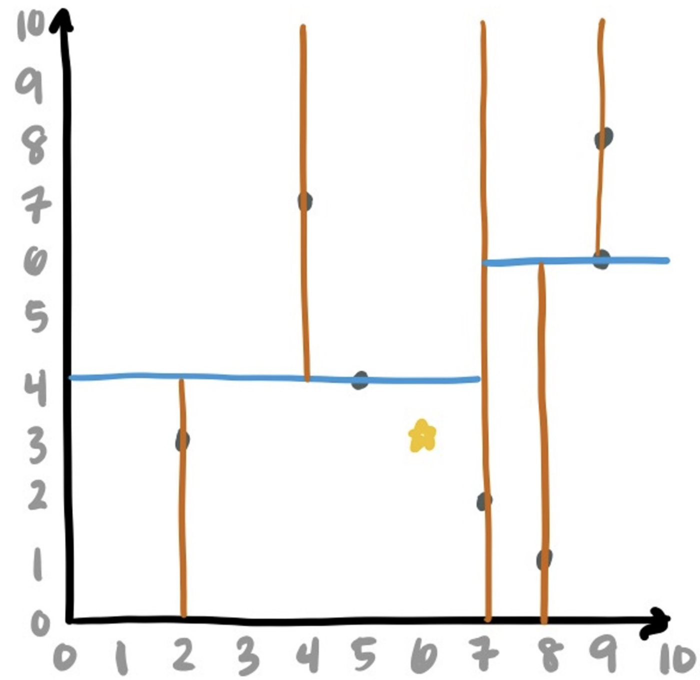
Consider points in 2D: $\mathbf{p} = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}$.

Tree construction:

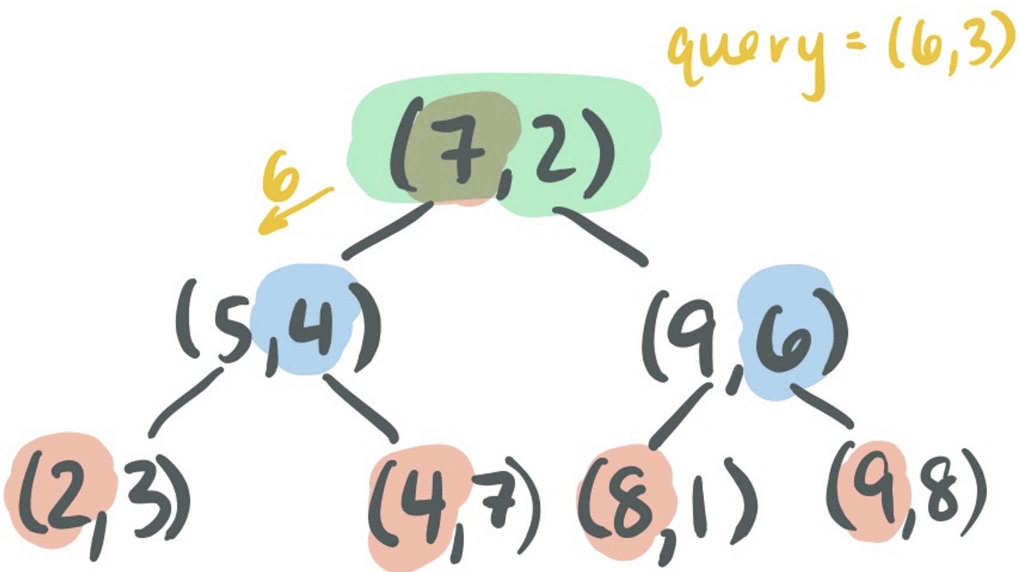


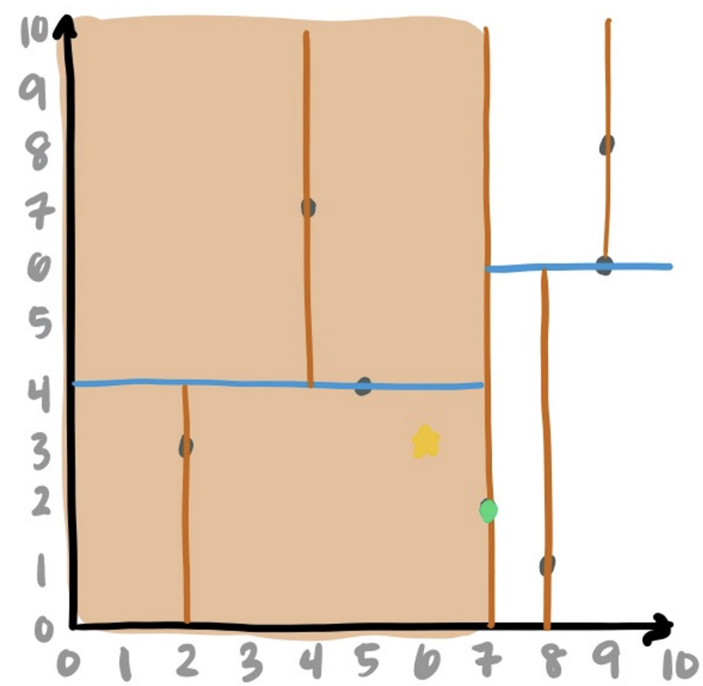
Nearest Neighbor – k-d Tree



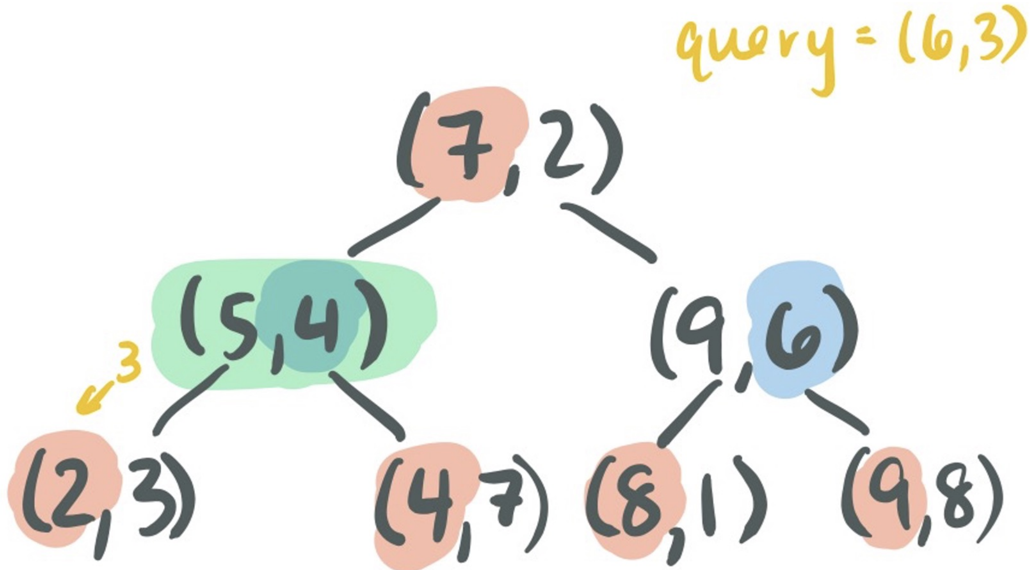


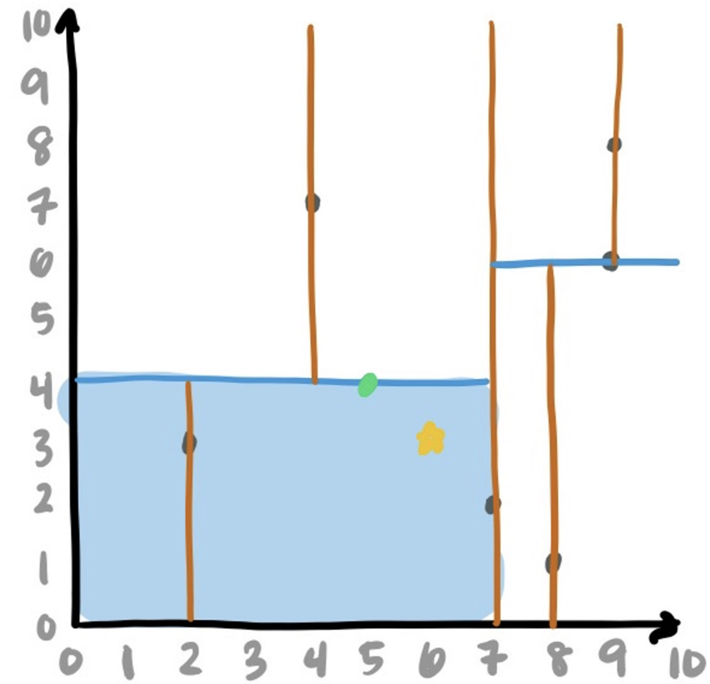
Nearest Neighbor - demo



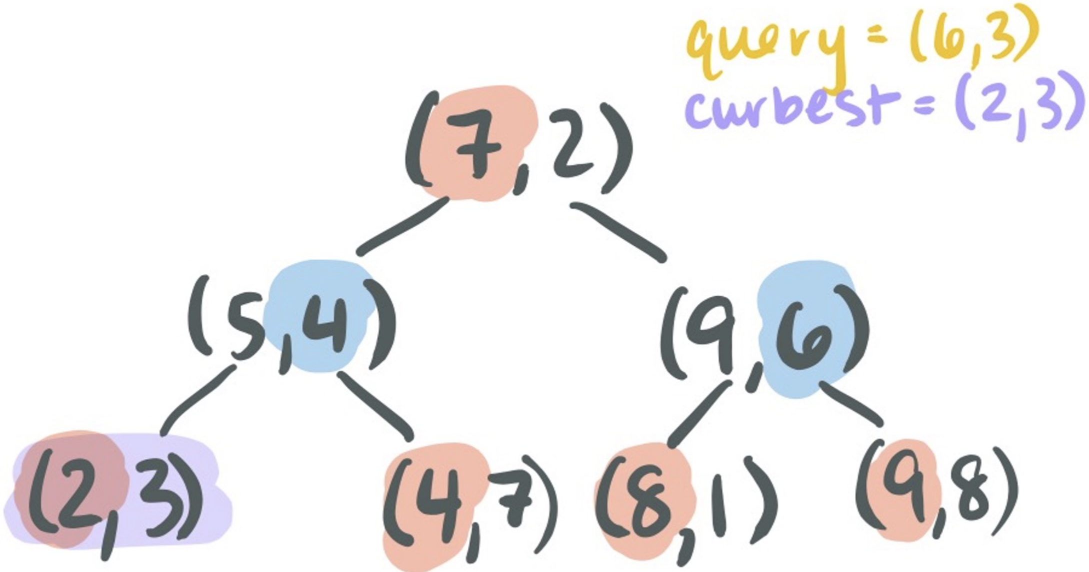


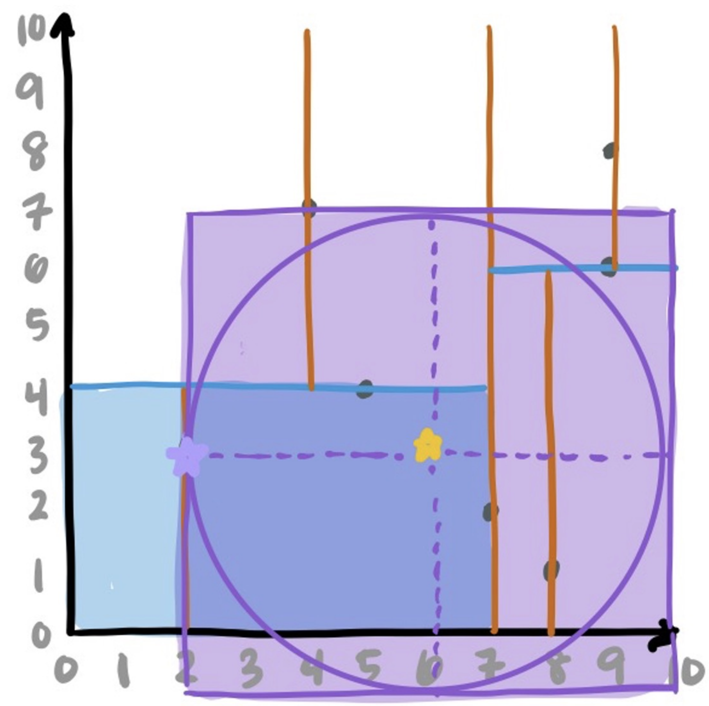
Nearest Neighbor - demo





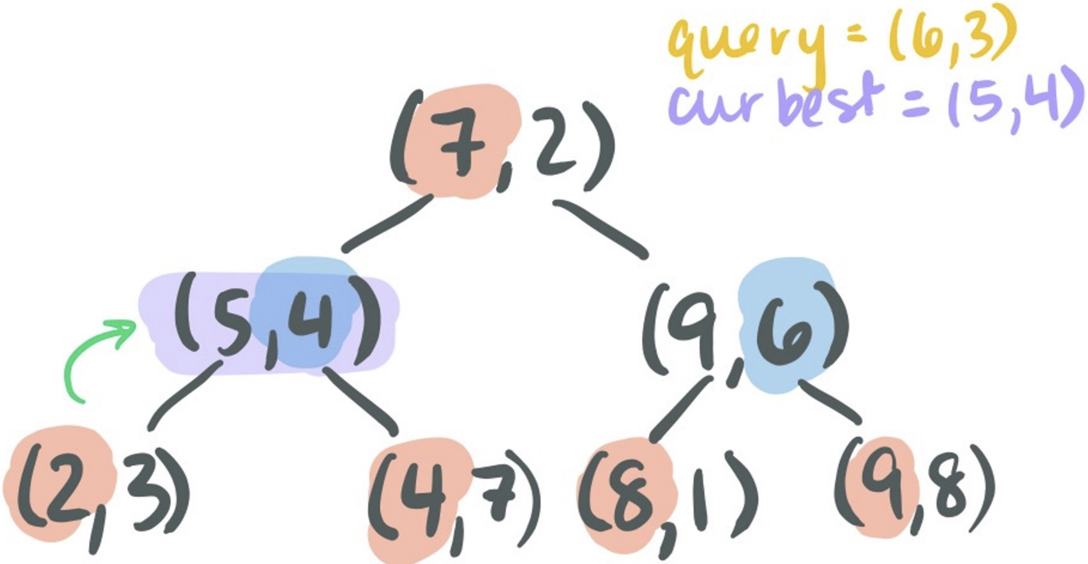
Nearest Neighbor - demo

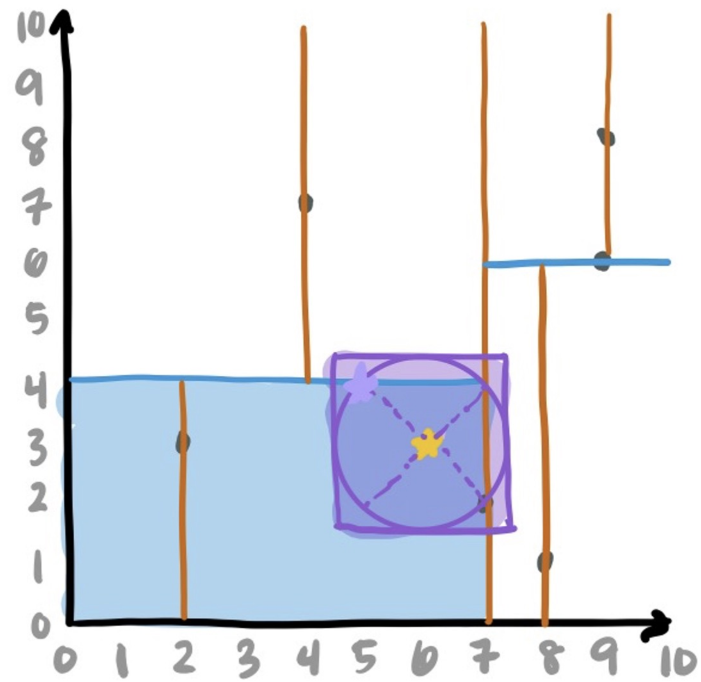




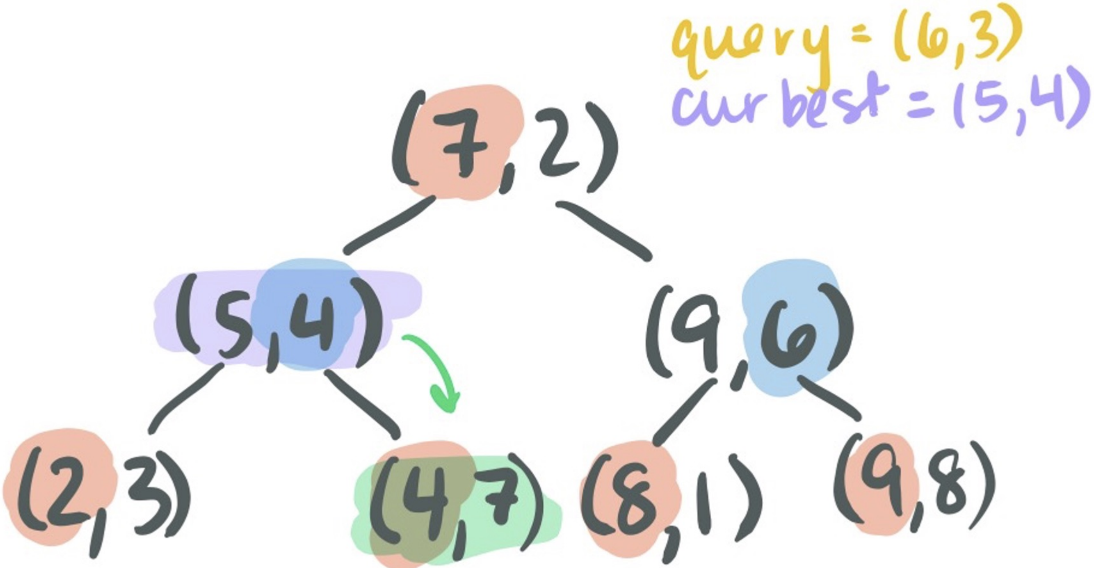
Nearest Neighbor - demo

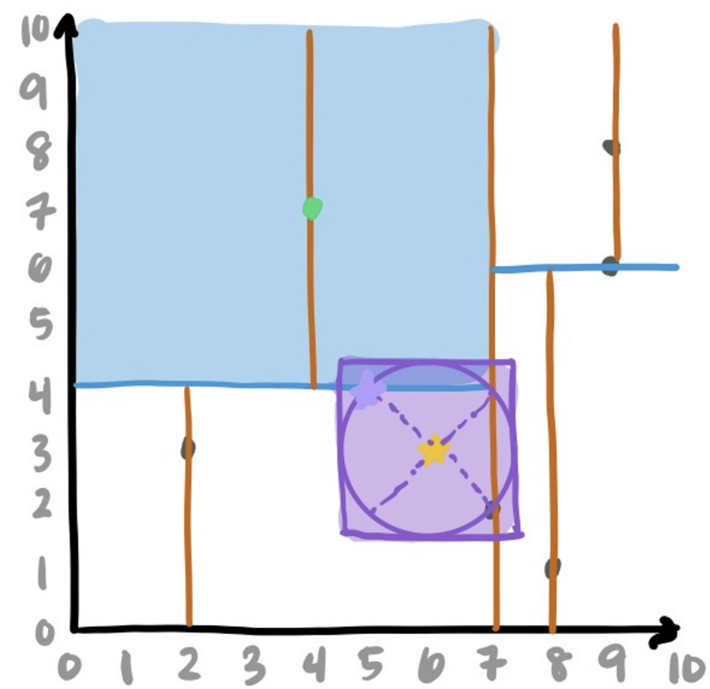
Backtracking: start recursing backwards -- store “best” possibility as you trace back





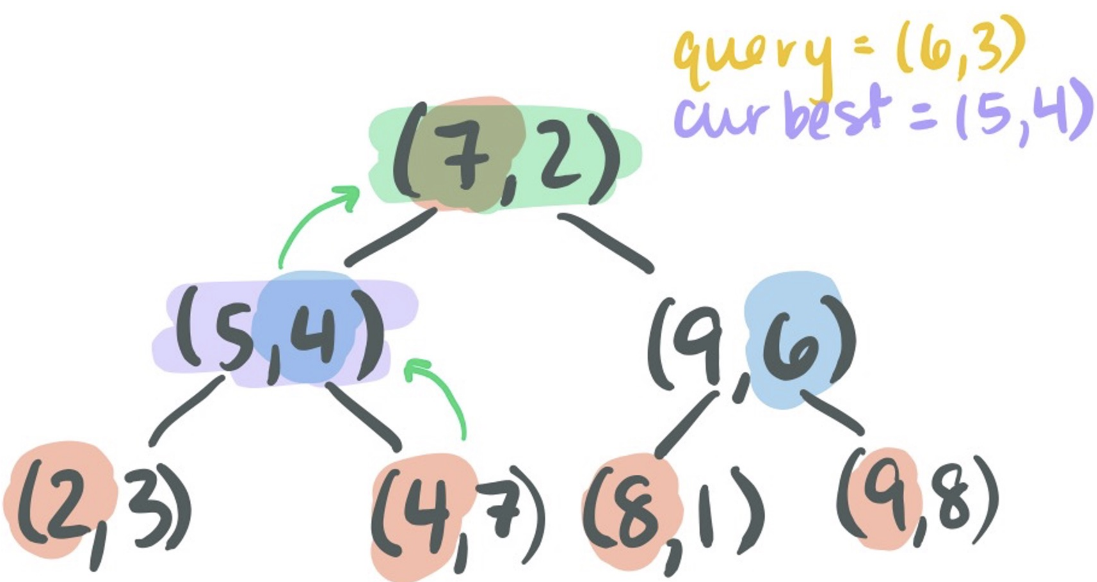
Nearest Neighbor - demo

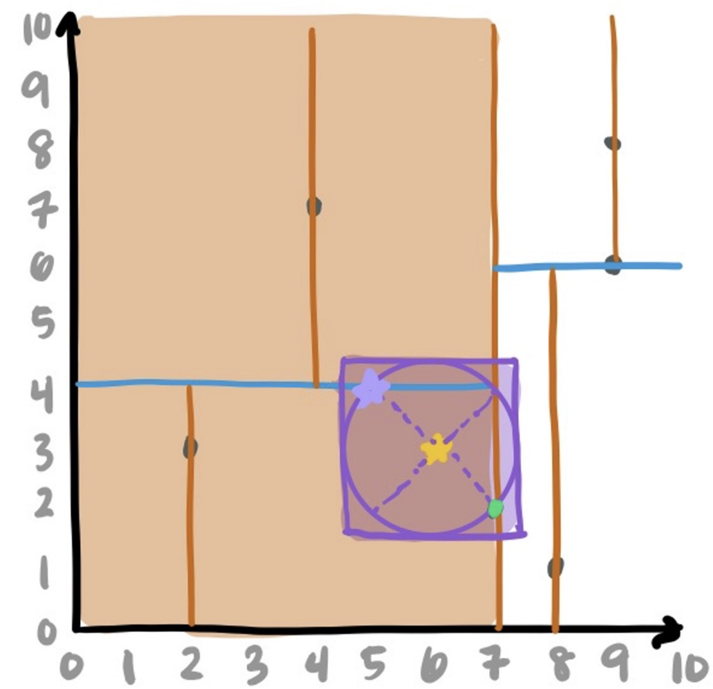




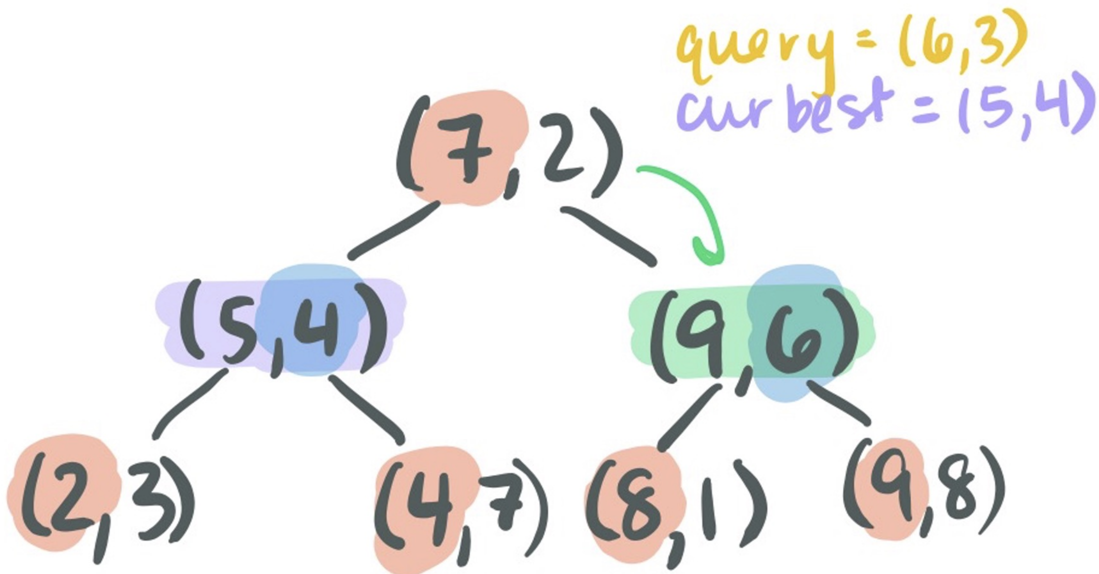
Nearest Neighbor - demo

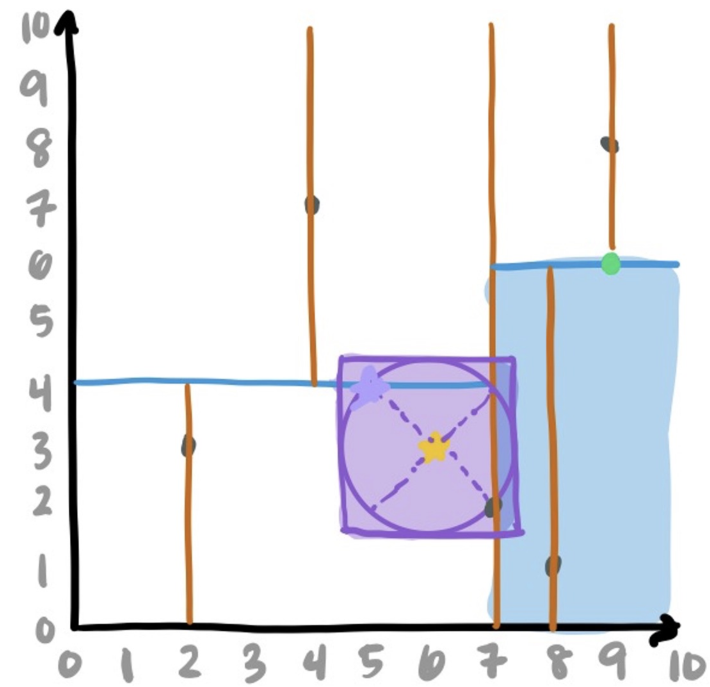
On ties, use `smallerDimVal` to determine which point remains `curBest`



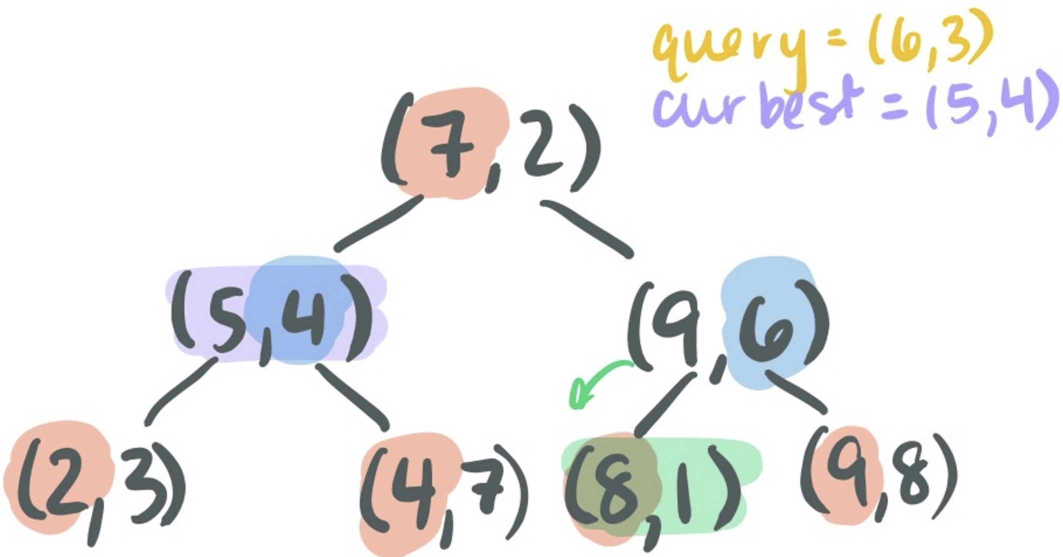


Nearest Neighbor - demo

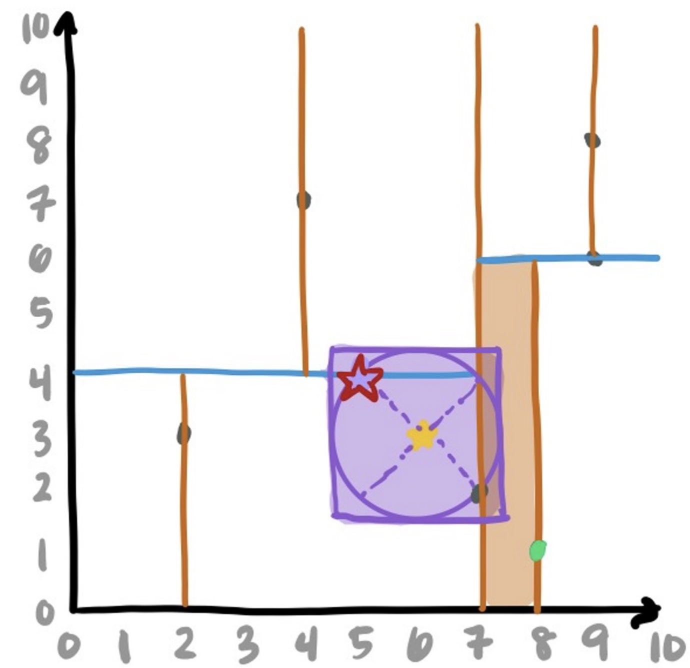




Nearest Neighbor - demo



BEST: (5,4)

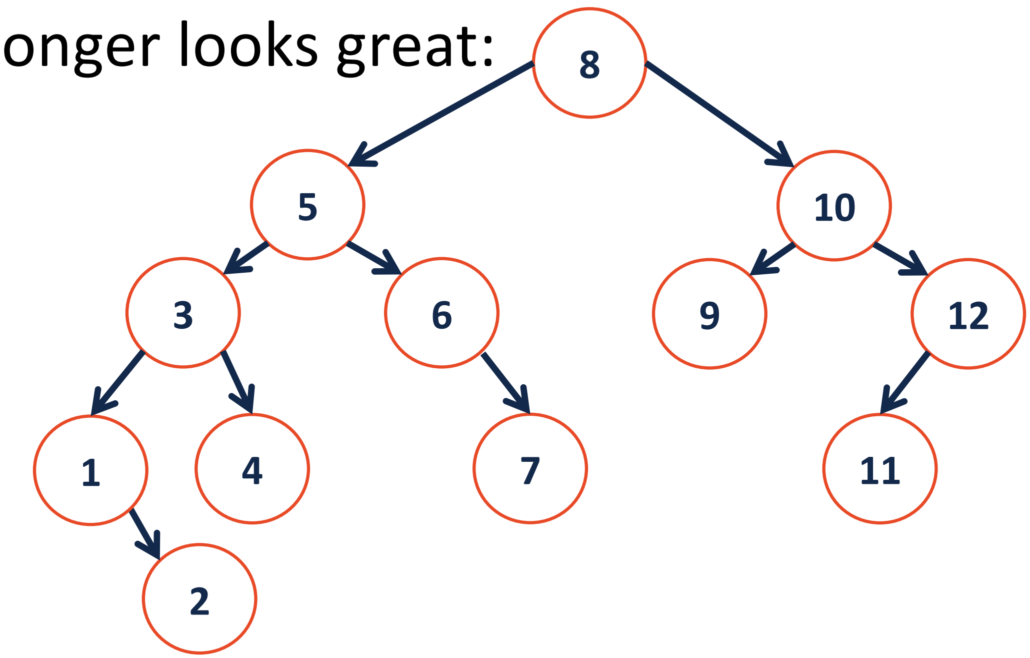


B-Tree Motivation

In Big-O we have assumed uniform time for all operations, but this isn’t always true.

However, seeking data from the cloud may take 40ms+.

…an O(lg(n)) AVL tree no longer looks great:



BTree Design Motivations

Knowing that we have large seek times for data, we want to:

BTree (of order m)

-3	8	23	25	31	42	43	55

m=9

Goal: Minimize the number of reads!

Build a tree that uses ______________________ / node

[1 network packet]

[1 disk block]

BTree Insertion

A **BTrees** of order **m** is an m-way tree:

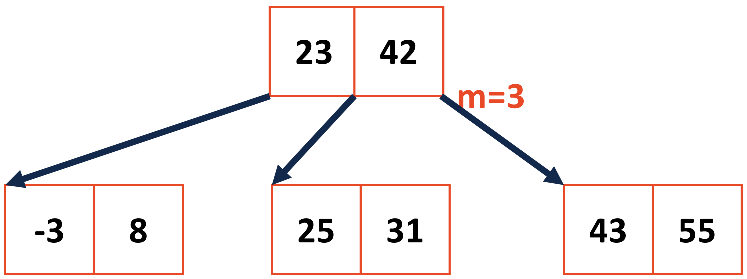
- All keys within a node are ordered
- All leaves contain hold no more than **m-1** keys.

BTree Insertion

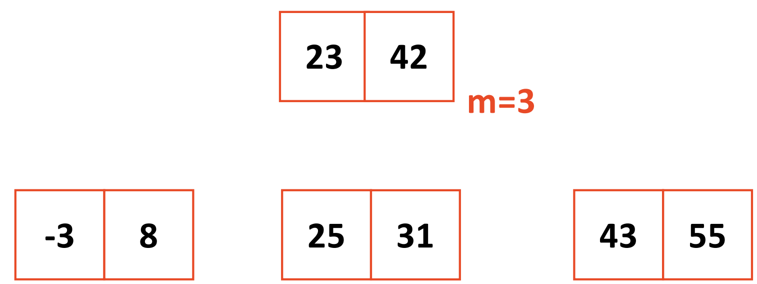
When a BTree node reaches **m** keys:

m=5

BTree Recursive Insert



BTree Recursive Insert



BTree Visualization/Tool

https://www.cs.usfca.edu/~galles/visualization/BTree.html

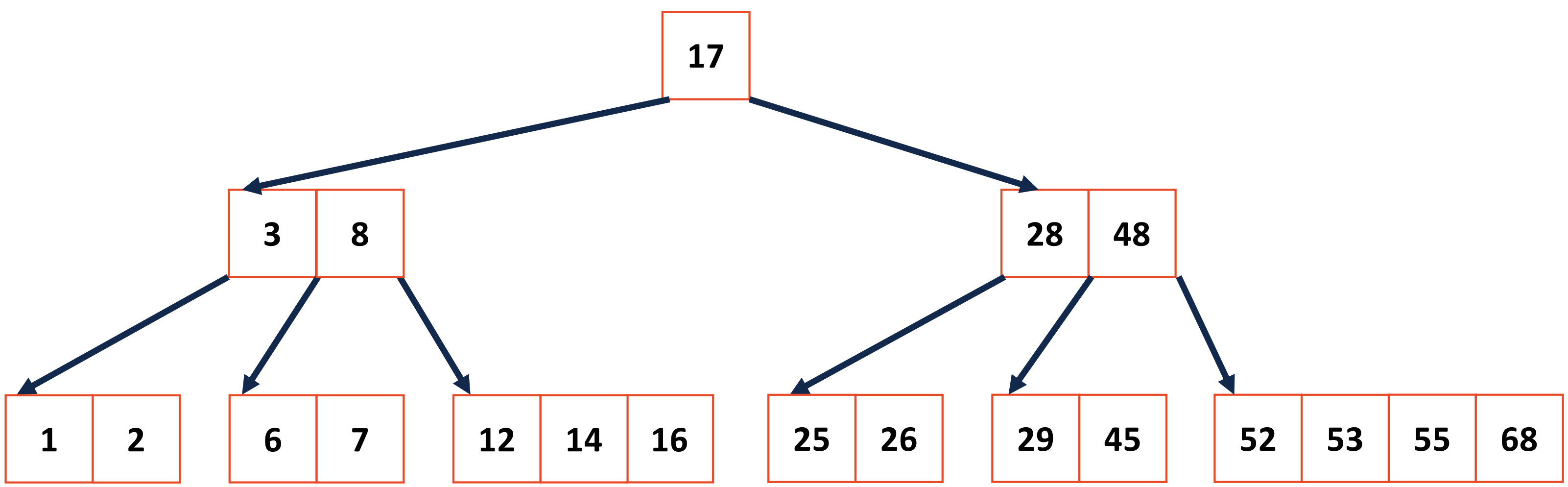
Btree Properties

A **BTrees** of order **m** is an m-way tree:
- All keys within a node are ordered
- All leaves contain no more than **m-1** keys.

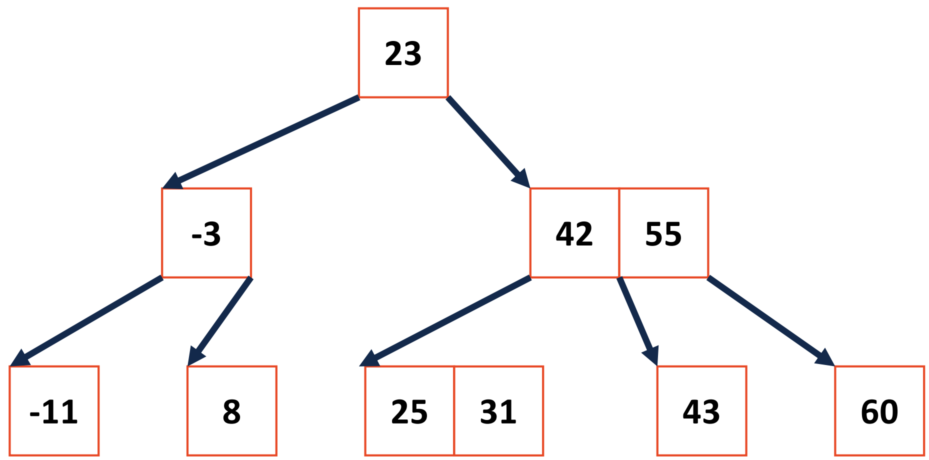
- All internal nodes have exactly **one more child than keys**
- Root nodes can be a leaf or have **[2, m]** children.
- All non-root, internal nodes have **[ceil(m/2), m]** children.

- All leaves are on the same level

BTree



BTree Search



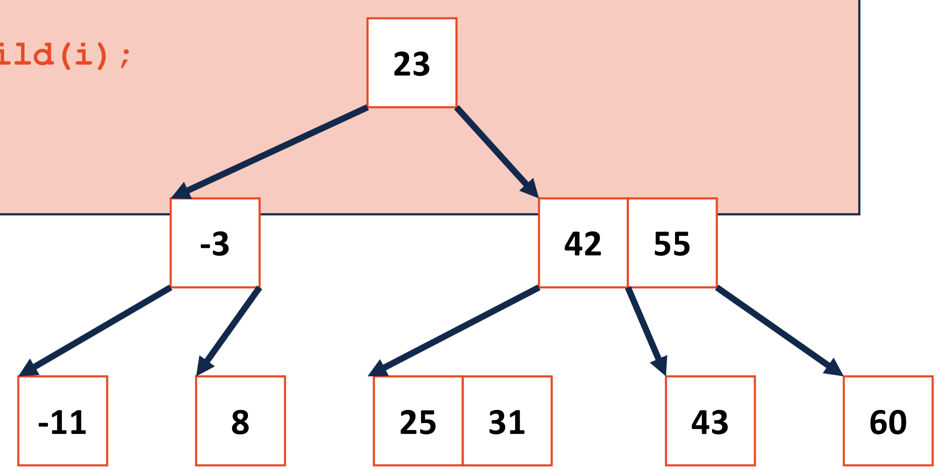
BTree Search

```
bool Btree::_exists(BTreeNode & node, const K & key) {

  unsigned i;
  for ( i = 0; i < node.keys_ct_ && key > node.keys_[i]; i++) { }

  if ( i < node.keys_ct_ && key == node.keys_[i] ) {
    return true;
  }

  if ( node.isLeaf() ) {
    return false;
  } else {
    BTreeNode nextChild = node._fetchChild(i);
    return _exists(nextChild, key);
  }
}
```



BTree Analysis

The height of the BTree determines maximum number of _____________ possible in search data.

…and the height of the structure is: _______________.

Therefore: The number of seeks is no more than ___________.

…suppose we want to prove this!

BTree Analysis

In our AVL Analysis, we saw finding an upper bound on the height (given **n**) is the same as finding a lower bound on the nodes (given **h**).

We want to find a relationship for BTrees between the number of keys (**n**) and the height (**h**).

BTree Analysis

Strategy:

We will first count the number of nodes, level by level.

Then, we will add the minimum number of keys per node (**n**).

The minimum number of nodes will tell us the largest possible height (**h**), allowing us to find an upper-bound on height.

BTree Analysis

The minimum number of **nodes** for a BTree of order m **at each level**:

root:

level 1:

level 2:

level 3:

…

level h:

BTree Analysis

The **total number of nodes** is the sum of all of the levels:

BTree Analysis

The **total number of keys**:

BTree Analysis

The **smallest total number of keys** is:

So an inequality about **n**, the total number of keys:

Solving for **h**, since **h** is the number of seek operations:

BTree Analysis

Given **m=101**, a tree of height **h=4** has:

Minimum Keys:

Maximum Keys: