CS 225

Data Structures

October 16 – AVL Applications

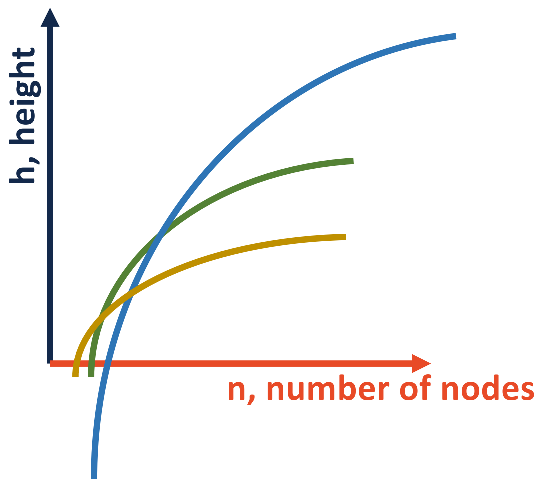
G Carl Evans

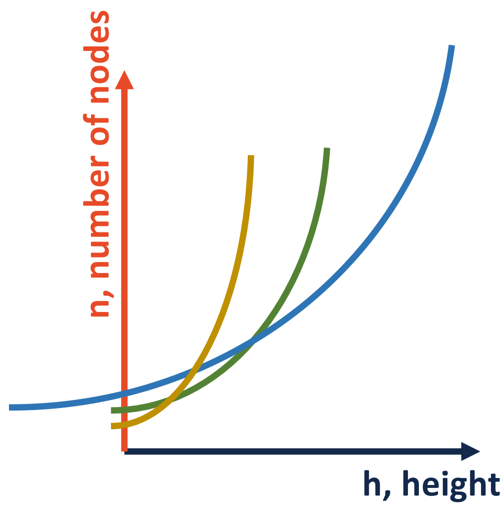
AVL Tree Analysis

We know: insert, remove and find runs in: ___________.

We will argue that: h is __________.

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 $\mathbf{n > k}$.

Plan of Action

Since our goal is to find the lower bound on **n** given **h**, we can begin by defining a function given **h** which describes the smallest number of nodes in an AVL tree of height **h**:

Simplify the Recurrence

$\mathbf{N(h)} = 1 + N(h - 1) + N(h - 2)$

State a Theorem

Theorem: An AVL tree of height h has at least ___________.

Proof:

I. Consider an AVL tree and let **h** denote its height.

II. Case: _______________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

III. Case: _______________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

IV. Case: _______________

By an Inductive Hypothesis (IH):

We will show that:

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

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

…and inverting:

AVL Runtime Proof

On Friday, we proved an upper-bound on the height of an AVL tree is **2 × lg(n)** or **O(lg(n))**:

N(h) := Minimum # of nodes in an AVL tree of height h

N(h) = 1 + **N(h-1)** + **N(h-2)**

$> 1 + \mathbf{2^{h-1/2} + 2^{h-2/2}}$

$> \mathbf{2 \times 2^{h-2/2} = 2^{h-2/2+1} = 2^{h/2}}$

Theorem #1:

Every AVL tree of height h has at least $2^{h/2}$ nodes.

AVL Runtime Proof

On Friday, we proved an upper-bound on the height of an AVL tree is **$2 \times \lg(n)$** or **$O(\lg(n))$**:

# of nodes (n) $\geq N(h) > 2^{h/2}$

$n > 2^{h/2}$

$\lg(n) > h/2$

$2 \times \lg(n) > h$

$h < 2 \times \lg(n)$ ***, for $h \geq 1$***

Proved: The maximum number of nodes in an AVL tree of height h is less than $2 \times \lg(n)$.

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).

Why Balanced BST?

Summary of Balanced BST

Pros:

- Running Time:

  - Improvement Over:

- Great for specific applications:

Summary of Balanced BST

Cons:

- Running Time:

- In-memory Requirement:

Red-Black Trees in C++

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

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

Red-Black Trees in C++

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

Red-Black Trees in C++

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

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

Red-Black Trees in C++

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

CS 225 -- Course Update

Your grades can now be viewed on moodle (https://learn.illinois.edu/)

We will discuss the grades for the course as a whole (ex: average, etc) in lecture on Wednesday.

Iterators

Why do we care?

```
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
  std::cout << (*it) << std::endl;
}
```

Iterators

Why do we care?

```
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
  std::cout << (*it) << std::endl;
}
```

```
DFS dfs(...);
for ( const Point & p : dfs ) {
  std::cout << p << std::endl;
}
```

Iterators

Why do we care?

```
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
  std::cout << (*it) << std::endl;
}
```

```
DFS dfs(...);
for ( const Point & p : dfs ) {
  std::cout << p << std::endl;
}
```

```
ImageTraversal & traversal = /* ... */;
for ( const Point & p : traversal ) {
  std::cout << p << std::endl;
}
```

Every Data Structure So Far

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

Range-based Searches

Q: Consider points in 1D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.
…what points fall in [11, 42]?

Tree construction:

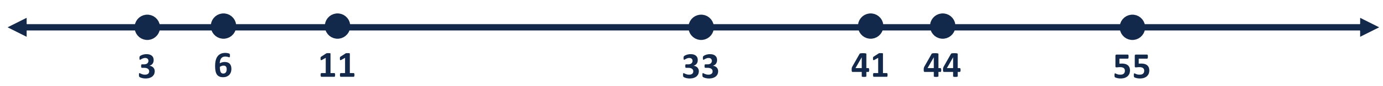
Range-based Searches

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:

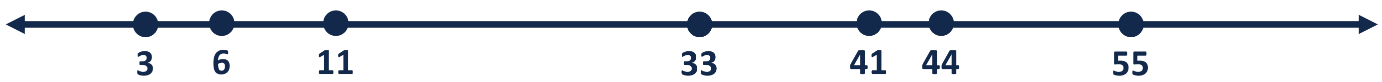


Range-based Searches

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

…what points fall in [11, 42]?

Ex:



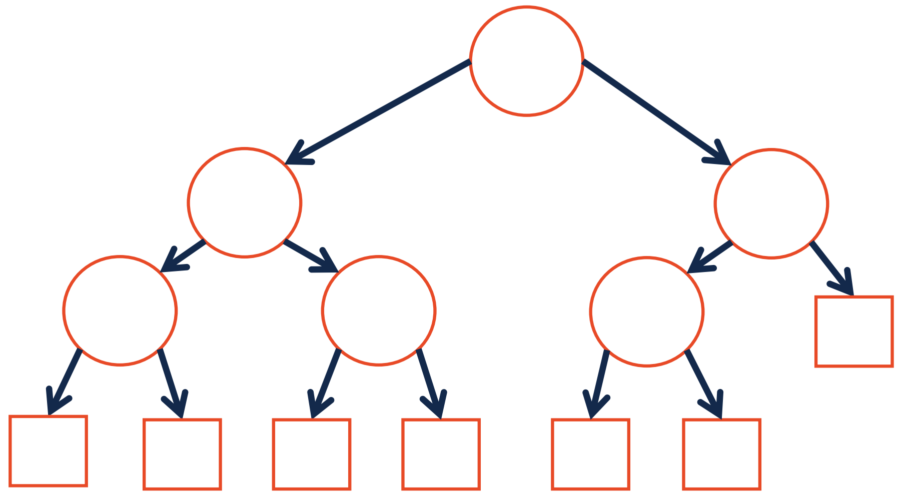
Range-based Searches

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

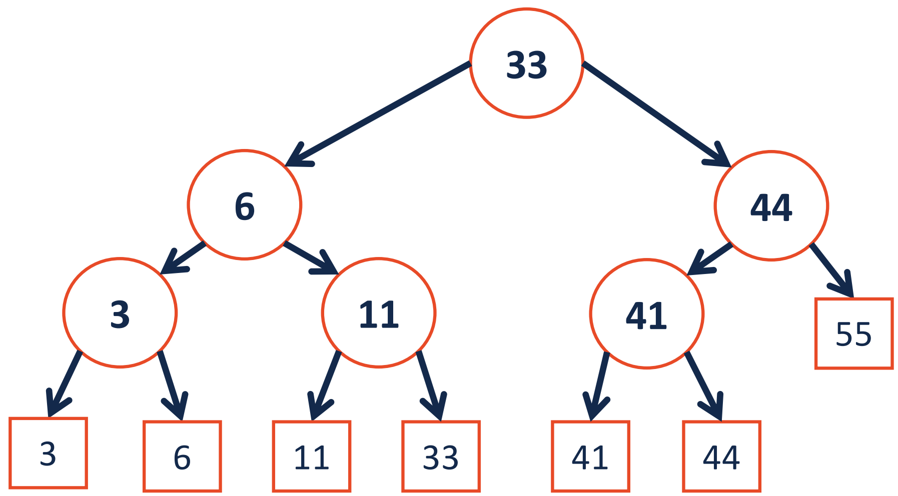
…what points fall in [11, 42]?

Tree construction:

Range-based Searches

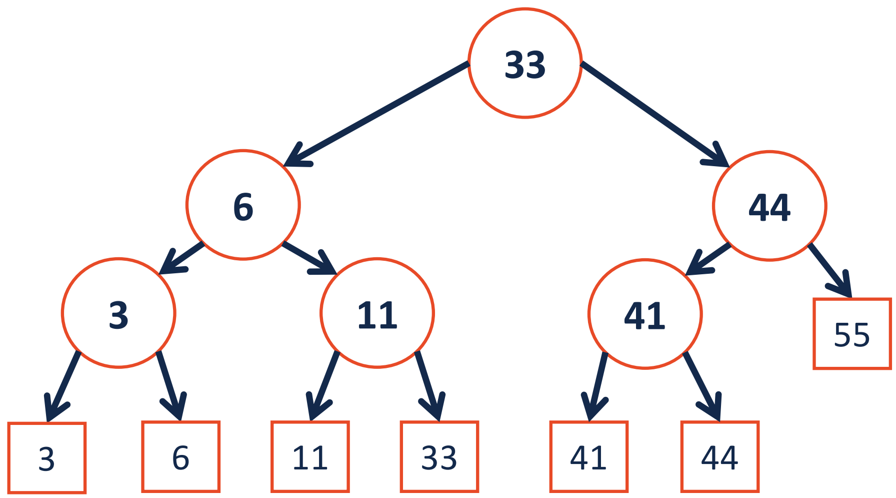


Range-based Searches

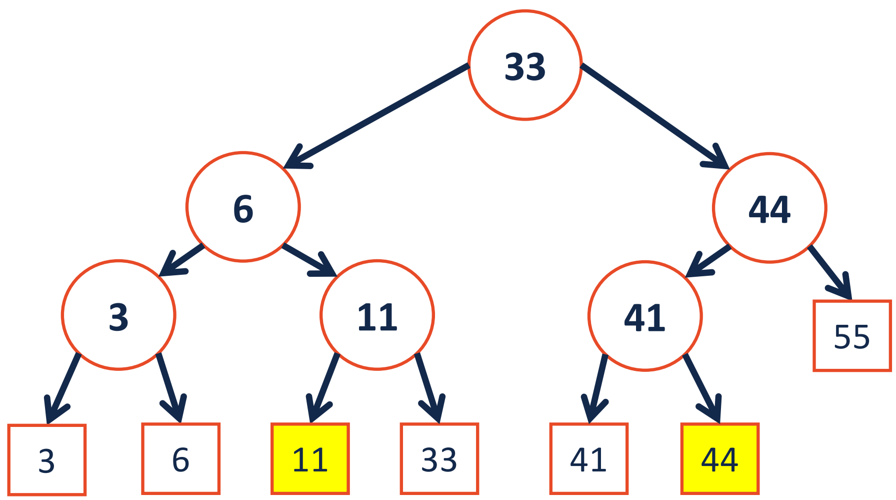


Range-based Searches

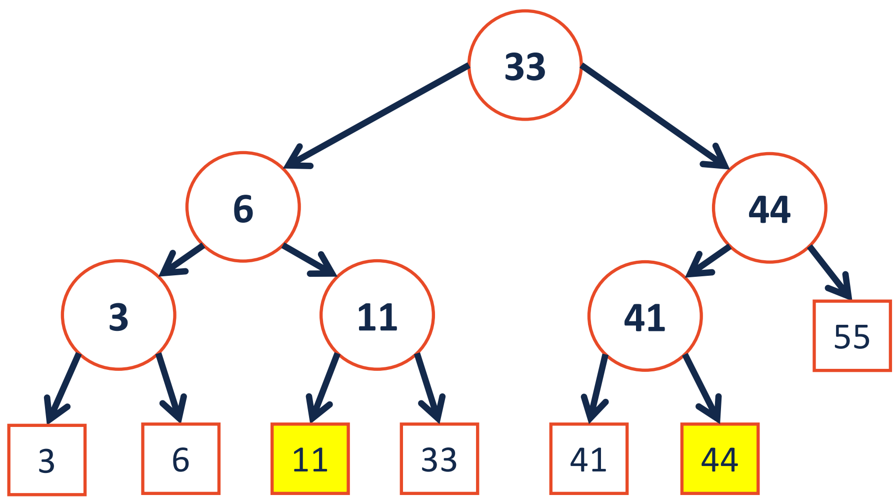
Q: Consider points in 1D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.
…what points fall in [11, 42]?



Range-based Searches



Running Time

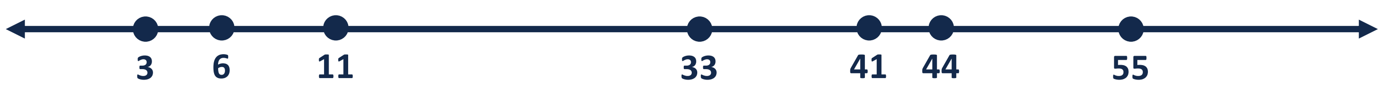


Range-based Searches

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

…what points fall in [11, 42]?

Ex:

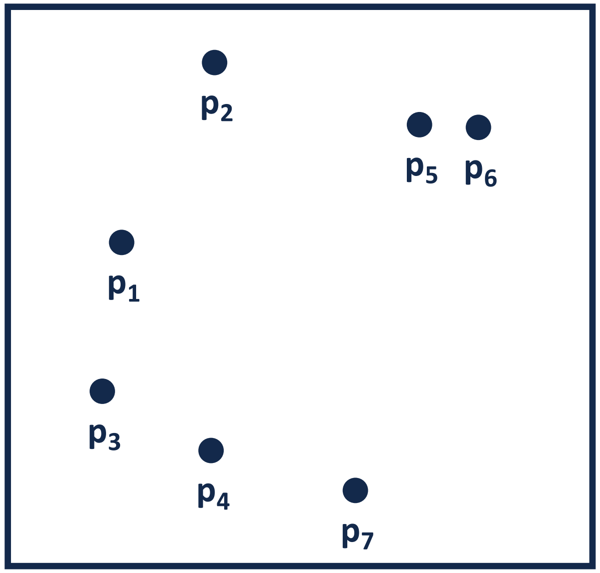


Range-based Searches

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

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

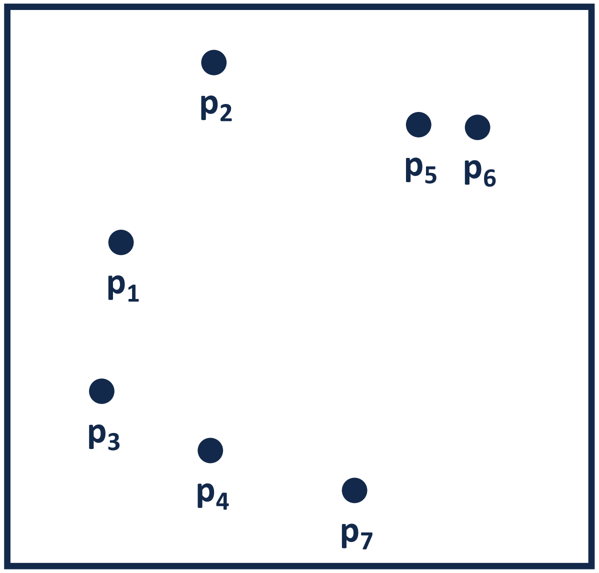
Q: What is the nearest point to $\mathbf{(x_1, y_1)}$?



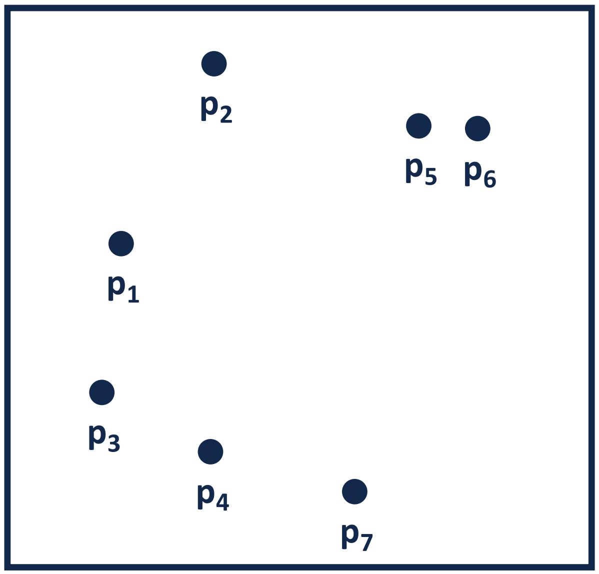
Range-based Searches

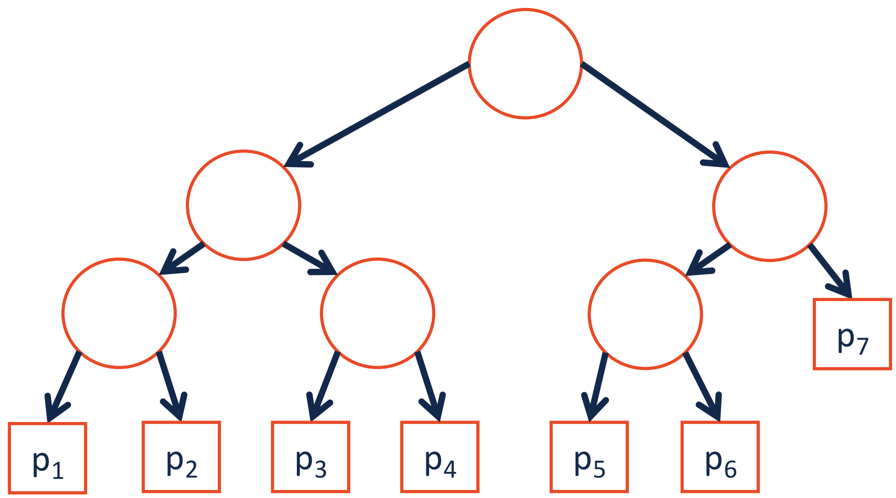
Consider points in 2D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

Tree construction:

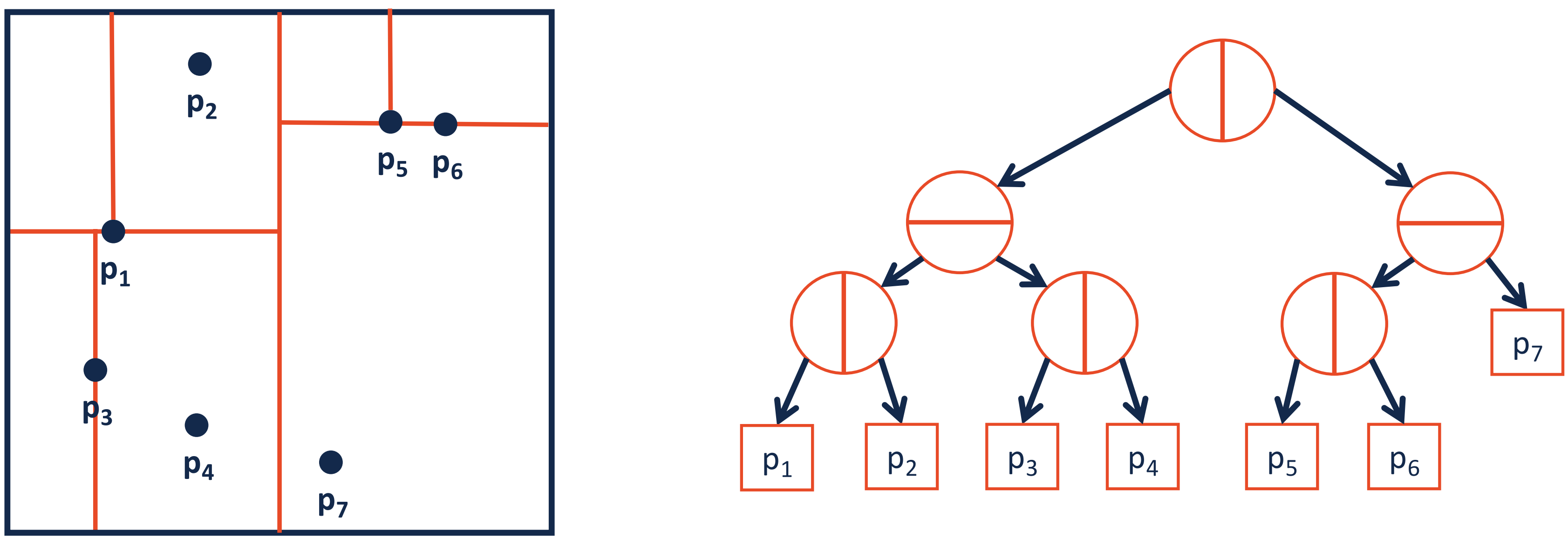


Range-based Searches





kD-Trees



kD-Trees