CS 2230
CS II: Data structures
Meeting 23: binary search trees
Brandon Myers
University of Iowa
Learning objectives

• Identify whether a binary tree is a *binary search tree (BST)*

• Run *add* and *contains* on a BST

• Run *remove* on a BST

• Calculate the running time of BST methods
Our motivation seeing the forest for the trees :)  

**Thought experiment:** Let’s use the Set ADT to implement a dictionary that can look up words.

```java
// recall: ListSet is unsorted
Set<String> dictionary = new ListSet<>();
```

**Write for 1 minute:**

If the number of words in the dictionary is $N$, how long will it take to look up a word?

If a human looked up a word in a book where the words are in a random order, how long will it take? Explain.

If the book was a normal dictionary (words sorted), how long will it take? Explain.
Our motivation seeing the forest for the trees :) 

**Thought experiment:** Let’s use the Set ADT to implement a dictionary that can look up words.

```java
// recall: ListSet is unsorted
Set<String> dictionary = new ListSet<>();

Set<String> dictionary = new BSTSet<>();
```

You will implement a different kind of Set using a “binary search tree” (BST) to learn more about this general concept: *organizing data carefully helps us access it with a lower running time*
Binary search tree (BST)
every element in the **left subtree** is **less than** the root
every element in the **right subtree** is **greater than** the root
true for every node
Identify whether a binary tree is a **binary search tree (BST)**

1) ![Binary Tree 1]

2) ![Binary Tree 2]

3) ![Binary Tree 3]

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Suppose we add the number 18 into this binary search tree. Where could we put 18 to make sure the tree stays a binary search tree?

a) right child of 33  
b) right child of 6  
c) left child of 31  
d) left child of 3  
e) right child of 12  
f) we can’t maintain the BST
find an element
(useful helper method)
find in Java

```java
private static Node find (int num, TreeNode node){
    if (node == null){
        return null;
    }
    else if (node.val == num){
        return node;
    }
    else if (node.val < num){
        return find(num, node.left);
    }
    else{
        return find(num, node.right);
    }
}
```
Run find on a BST

List the order in which nodes will be visited when we call find(17)

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Calculate the running time of BST methods

What is the 1) worst case, 2) best case running time of find(x)?
Answer in terms of N, the number of nodes. Assume all the subtrees are equally sized (as shown).

a) $O(1)$
b) $O(\log N)$
c) $O(N)$
d) $O(N \log N)$
e) $O(N^2)$
f) $O(2^N)$
Why is the height $O(\log N)$?

<table>
<thead>
<tr>
<th>$N$</th>
<th>height</th>
<th>bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

$N = 2^{\text{height}} + \text{bottoms} - 1$

$(N - \text{bottoms} + 1) = 2^{\text{height}}$

$\log_2 (N - \text{bottoms} + 1) = \text{height}$

or equivalently

$\text{height} = \lfloor \log_2 N \rfloor$

and, we know

$\lfloor \log_2 N \rfloor \in O(\log N)$
add an element

• Here is the initial tree.
• We want to insert 24 into this tree
add an element

- Start at root and check if 24 is less than or greater than.
- Move right of 18
- Left of 25
- Right of 24
Run add on a BST

1. Where will the key 11 go when we call add(11)?
2. Where will the key 22 go when we call add(22)?
An exceptional case

If we use our algorithm for add(), what will this BST look like?
(draw it)
It’s possible to get unlucky

add(30)
add(24)
add(18)
add(10)
add(7)
add(5)
add(2)

find is worst case $O(N)$!