CSE 12 – Basic Data Structures and Object-Oriented Design Lecture 19

Greg Miranda & Paul Cao, Winter 2021

This lecture is being recorded

Announcements

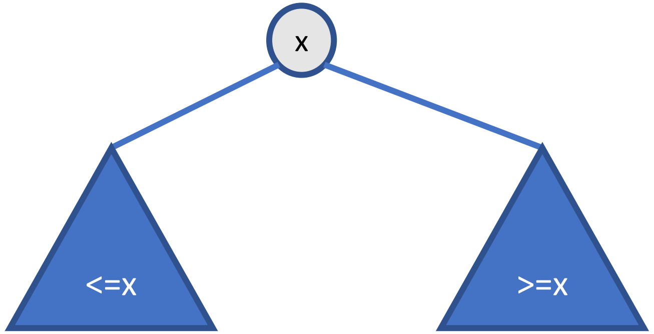
- Quiz 19 due Wednesday @ 8am
- Survey 8 due Friday @ 11:59pm
- PA7 due next Tuesday (3/2) @ 11:59pm
- Exam 2- see Piazza post

Topics

- Questions on Lecture 19?
- Binary Search Trees

Binary Search Tree

- A binary tree where the key in each node must be greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree



BST Find

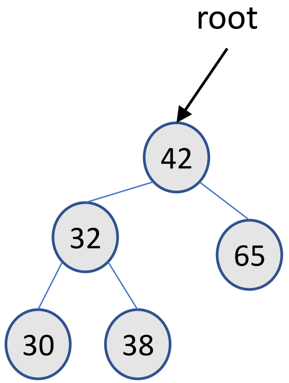
```
//Adopted from a generic binary tree
boolean containsHelper(BSTNode currRoot, Integer toFind) {
    if (currRoot == null) return false;  // first base case
    if (currRoot.value.equals(toFind)) //second base case
       return true;
    return containsHelper(currRoot.left, toFind)
                       || containsHelper(currRoot.right, toFind);
}
```

Can I just use this for BST find?

A. Yes, it will work just fine.
B. Yes, but we can probably do better
C. No, it won’t work for a BST

```
//Adopted from a generic binary tree
boolean findHelper(BSTNode currRoot, Integer toFind) {
    if (currRoot == null) return false;  // first base case
    if (currRoot.value.equals(toFind)) //second base case
       return true;
    return containsHelper(currRoot.left, toFind)
                        || containsHelper(currRoot.right,
toFind);
}
```

FindHelper(root, 38)

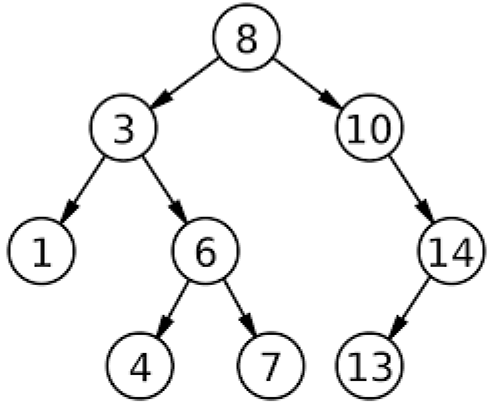


```
//BST version
boolean findHelper(BSTNode curr, Integer value){
  if (curr == null) return false;
  if (curr.value.equals(value)) return true;
  if (curr.value.compareTo(value)<0){ //value is bigger than current node
    return findHelper(curr.right, value);
  }
  else{
    return findHelper(curr.left, value);
  }
}
```

Binary Search Tree

What order does PAE() traverse the tree?

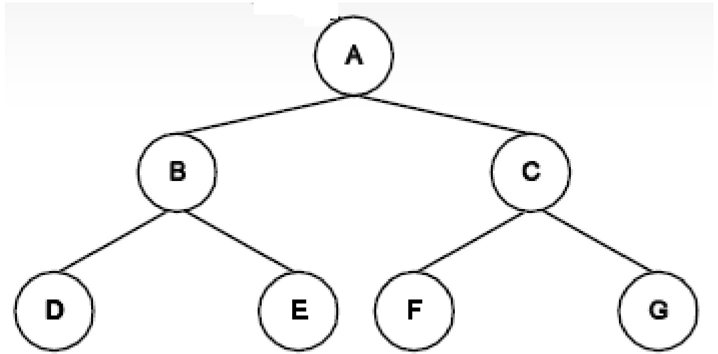
```
void printAllElements(Node<K, N> n) {
  if (n == null ) return;
  System.out.println(n.key);
  printAllElements(n.left);
  printAllElements(n.right);
}
void printAllElement() {
  printAllElements(this.root);
}
```


$$\begin{tikzcd}
 & & & 8 \arrow[dll] \arrow[dr] & \\
 & 3 \arrow[dl] \arrow[dr] & & & 10 \arrow[d] \\
1 & & 6 \arrow[dl] \arrow[d] & & 14 \arrow[dl] \\
 & 4 & 7 & 13 &
\end{tikzcd}$$

What's the post, pre, in-order traversal of this tree?

In-order traversal

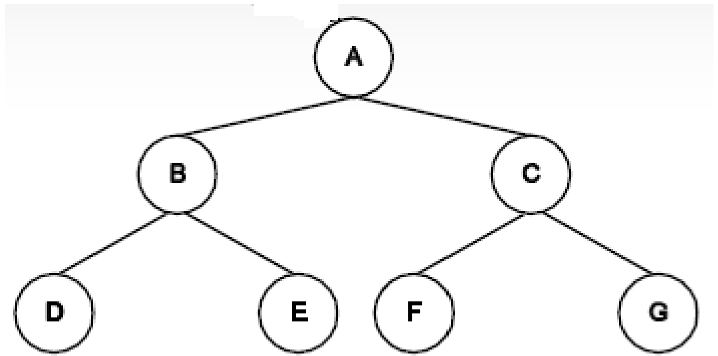
```
inorder(node) {
  if (node != null){
     inorder(node.left)
     visit this node
     inorder(node.right)
  }
}
```



A. D B E A F C G
B. A B D E C F G
C. A B C D E F G
D. D E B F G C A
E. Other

Pre-order traversal

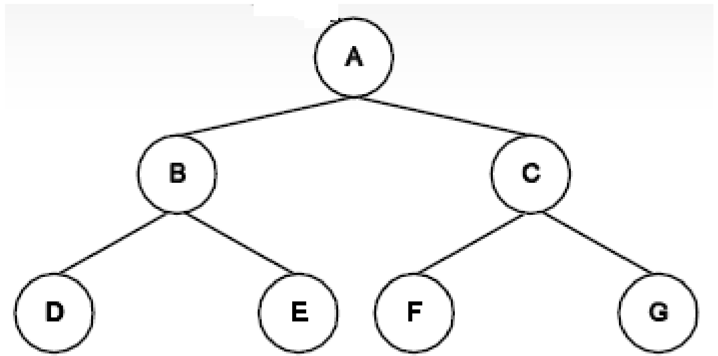
```
preorder(node) {
  if (node != null){
      visit this node
      preorder(node.left)
      preorder(node.right)
  }
}
```



A. D B E A F C G
B. A B D E C F G
C. A B C D E F G
D. D E B F G C A
E. Other/none/more

Post-order traversal

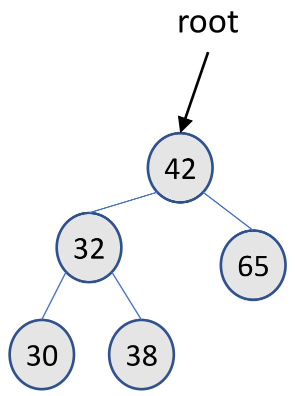
```
postorder(node) {
  if (node != null){
      postorder(node.left)
      postorder(node.right)
      visit this node
  }
}
```



A. D B E A F C G
B. A B D E C F G
C. A B C D E F G
D. D E B F G C A
E. Other/none/more

The BST and BSTNode Classes

```
public class BST<E extends Comparable<E>>
{
  /** Inner class for the BSTNode */
  private class BSTNode {
    BSTNode leftChild;
    BSTNode rightChild;
    BSTNode parent;

    E element;

    public BSTNode(E elem){
      element = elem;
    }
  }
  BSTNode root;
  ......
}
```



What is the WORST CASE cost for doing find() in a BST?

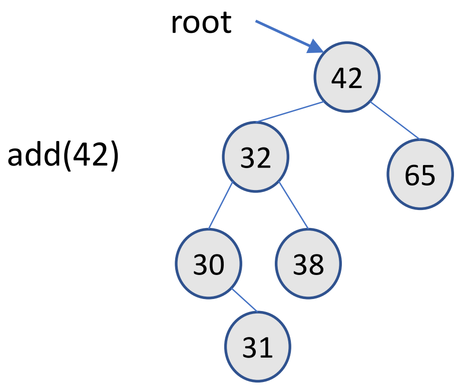
A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. O(n^2)

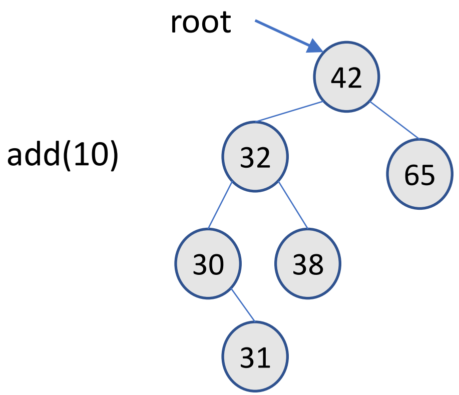
What is the WORST CASE cost for doing find() in a BST ***if the BST is full/"balanced"***?

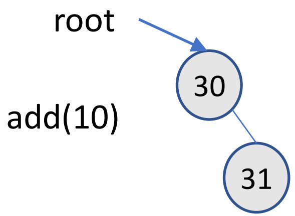
A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. O(n^2)

BST Add: With recursion!

Consider the following:





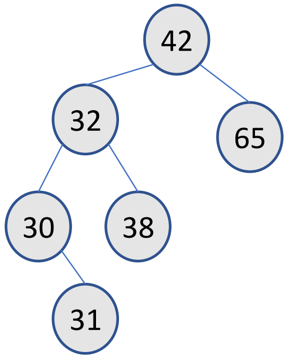


BST Add: Recursively

```
boolean add( E toAdd ) {
   if (toAdd == null) throw new NullPointerException();
   if (root == null) {
      root = new BSTNode(toAdd);
   }
   return addHelper(root, toAdd );
}

boolean addHelper( BSTNode currRoot, E toAdd )
{
      …
}
```

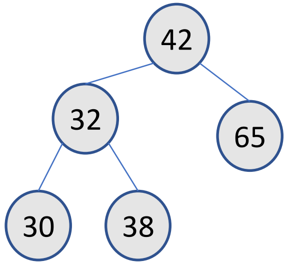
Which of these is/are a base case for addHelper?

A. currRoot is null
B. currRoot's element is equal to toAdd
C. Both A & B
D. Neither of these



BST Add: Recursively

```
boolean add( E toAdd ) {
   if (root == null) {
      root = new BSTNode(toAdd);
   }
   return addHelper( root, toAdd );
}

boolean addHelper( BSTNode currRoot, E toAdd )
{
    int compare = toAdd.compareTo(currRoot.getElement());
    if (compare == 0) {
        return false;
    }
    // Finish the code…
}
```



BST Add: Recursively

```
boolean addHelper(BSTNode curr, Integer value){
  int result = curr.value.compareTo(value);
  if (result == 0){
    return false;
  }
  if (result > 0){
    if (_____________== null){
      _____________ = new BSTNode(value);
      return true;
    }
    else{
      return _______________________
    }
  }
  else{//Similar idea
  }
}
```

What should I fill in the red blank (they should be the same)

A. root.left
B. root.right
C. curr.left
D. curr.right
E. Something else

BST Add: Recursively

```
 boolean addHelper(BSTNode curr, Integer value){
   int result = curr.value.compareTo(value);
   if (result == 0){
     return false;
   }
   if (result > 0){
     if (curr.left == null){
       curr.left = new BSTNode(value);
       return true;
     }
     else{
       return ________________________;
     }
   }
   else{//Similar idea
   }
}
```

What should I fill in the red blank

A. addHelper(root.right, value)
B. addHelper(root.left, value)
C. addHelper(curr.left, value)
D. addHelper(curr.right, value)
E. Something else

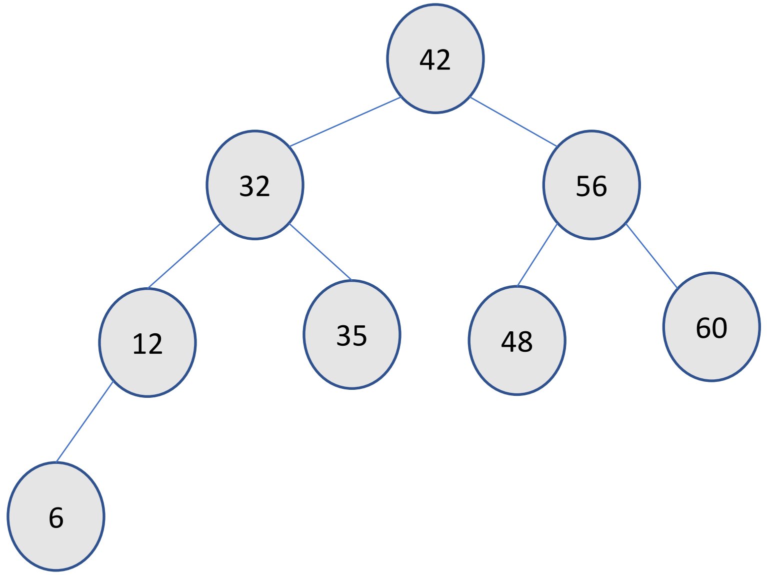
How to debug your code

- Print out your tree

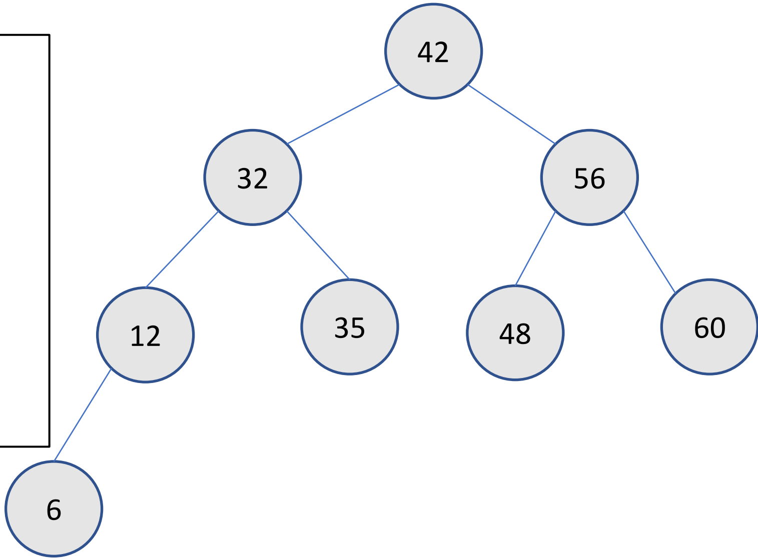
```
public String treePrint(){
  if (root == null) return "";
  return treePrintHelper(root);
}
public String treePrintHelper(BSTNode curr){
  if (curr == null) return "";
  String temp =__________________;
  //make sure you put in curr, curr.left, curr.right
  //for easy identification of node relationships
  return temp + treePrintHelper(curr.left) + treePrintHelper(curr.right);
}
```

Remove from a BST

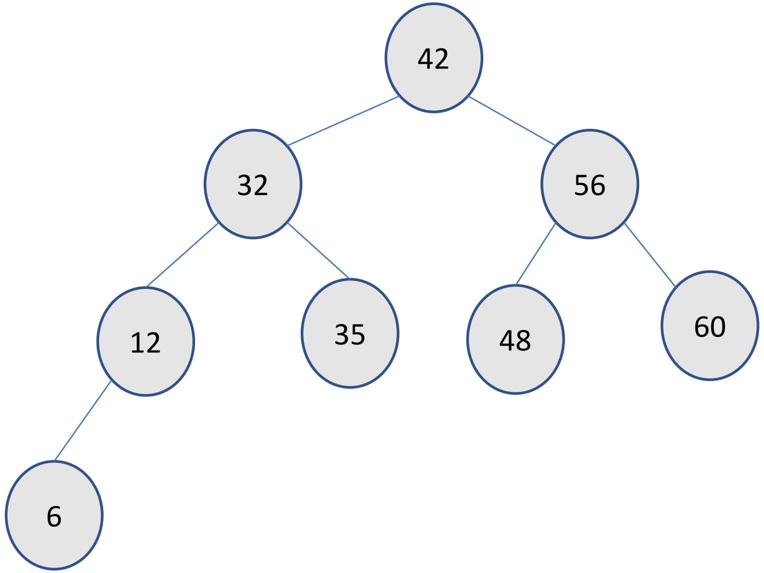
- Find the node while keeping track of the parent of the node you are about to visit
- Delete the node
    1. Node to delete is a leaf node
    2. Node to delete only has one child
    3. Node to delete has two children



```
public void removeHelper(BSTNode curr){

  if (curr.left != null && curr.right != null){
     //a node with 2 children

   }

  if (curr == root){
     //root node

   }

  if (curr.left!= null){

  }
  else if (curr.right != null){

  }

  if (curr.left == null && curr.right == null){
     //leaf node

   }
}
```



Successor of a Node



What is the successor of 32?

A. 35
B. 42
C. 60
D. 6
E. Something else

What is the successor of 12?

A. 32
B. 35
C. 42
D. 48
E. Something else

Which class is a better fit to have the successor function?

A. In BSTNode class
B. In BST class
C. Either one is fine. It depends on your design

min of a BST

```
public BSTNode min(BSTNode curr){
  if (________A_________){
    return curr;
  }
  else{
    return_________B_____________;
  }
}
```

What should I fill in blank A?

A. root.left == null
B. root.left != null
C. curr != null
D. curr.left != null
E. Something else

What should I fill in blank B?

A. min(root.left)
B. min(curr.right)
C. min(curr.left)
D. min(curr.left.right)
E. Something else

Questions on Lecture 19?