Binary Search Tree (BST)
Lecture Question

Task: Write a method to convert a BST to a List

- In the week8.trees.BinarySearchTree class write a method named toList that takes no parameters and returns the values of the tree in a List in sorted order
BST - Definition

- For each node:
  - All values in the left subtree are less than the node's value
  - All values in the right subtree are greater than the node's value
  - Duplicate values handled differently based on implementation
    - Sometimes not allowed at all

![BST Diagram]

-3  2  4  5  7  8  14
BST - Code

• To make the BST generic
• Take a type parameter
• Take a comparator to decide the sorted order
• Store a reference to the root node

```scala
class BinarySearchTree[A](comparator: (A, A) => Boolean) {
  var root: BinaryTreeNode[A] = null
  def insert(a: A): Unit
  def find(a: A): BinaryTreeNode[A]
}
```
class BinarySearchTree[A](comparator: (A, A) => Boolean) {
  var root: BinaryTreeNode[A] = null

  def insert(a: A): Unit
  def find(a: A): BinaryTreeNode[A]
}

val intLessThan = (a: Int, b: Int) => a < b
val bst = new BinarySearchTree[Int](intLessThan)
bst.insert(5)
bst.insert(2)
bst.insert(8)
bst.insert(4)
bst.insert(7)
bst.insert(14)
bst.insert(-3)

val node = bst.find(4)
BST - Find

- If the value to find is less than the value of the node - Move to the left child
- If the value to find is greater than the value of the node - Move to right child
- If value is found - return this node
- If value is not found - return null

```python
def find(a: A): BinaryTreeNode[A] = {
    findHelper(a, this.root)
}

def findHelper(a: A, node: BinaryTreeNode[A]): BinaryTreeNode[A] = {
    if(node == null){
        null
    }else if(comparator(a, node.value)){
        findHelper(a, node.left)
    }else if(comparator(node.value, a)){
        findHelper(a, node.right)
    }else{
        node
    }
}
```
BST - Find

- Find the value 4
BST - Find

- Find the value 4
- 4 < 5
 BST - Find

- Find the value 4
- $4 < 5$
- $4 > 2$
• Find the value 4
• $4 < 5$
• $2 < 4$
• $4 == 4$ - return this node
BST - Insert

- Run find until a null node is reached - insert new node here
- If value is a duplicate, move to the left

```scala
def insert(a: A): Unit = {
  if(this.root == null){
    this.root = new BinaryTreeNode(a, null, null)
  }else{
    insertHelper(a, this.root)
  }
}

def insertHelper(a: A, node: BinaryTreeNode[A]): Unit = {
  if(comparator(node.value, a)){
    if(node.right == null){
      node.right = new BinaryTreeNode[A](a, null, null)
    }else{
      insertHelper(a, node.right)
    }
  }else{
    if(node.left == null){
      node.left = new BinaryTreeNode[A](a, null, null)
    }else{
      insertHelper(a, node.left)
    }
  }
}
```
BST - Insert

- Insert 7

Diagram:

```
  5
 / \
2   8
/   /  \
-3  4   14
```
BST - Insert

- Insert 7
- $5 < 7$
BST - Insert

- Insert 7
- 5 < 7
- 7 < 8 and left child is null - Insert here
BST - Insert

- Insert 7
- $5 < 7$
- $7 < 8$ and left child is null - Insert here
In-Order traversal of a BST iterates over the values in sorted order

- Visit all elements of the left subtree
  - Elements less than the node's value
- Visit the node's value
- Visit all elements of the right subtree
  - Elements greater than the node's value

Printed:
-3 2 4 5 7 8 14
BST - Efficiency

- Vocab: A tree is balanced if each node has the same number of descendants in its left and right subtrees

- *If a BST is balanced*
  - The number of nodes from the root to a leaf - the height of the tree - is $O(\log(n))$
  - Insert and find take $O(\log(n))$ time
  - Inserting n elements effectively sorts in $O(n*\log(n))$ time
  - Advantage: Sorted order is efficiently maintained as new elements are added in $O(\log(n))$

- Array takes $O(n)$ to insert
- Linked list takes $O(n)$ to find where to insert
• What if the tree is not balanced?

```scala
val intLessThan = (a: Int, b: Int) => a < b
val bst = new BinarySearchTree[Int](intLessThan)
bst.insert(-3)
bst.insert(2)
bst.insert(4)
bst.insert(5)
bst.insert(8)
bst.insert(7)
bst.insert(14)
```
BST - Inefficiency

- If elements are inserted in sorted order
- Tree effectively becomes a linked list
- $O(n)$ insert and find
• How do we keep the tree balanced and still insert in $O(\log(n))$ time
• How would we remove a node while maintaining sorted order?
• How do we handle duplicate values?
  • Should duplicates even be allowed?

• Answers to these questions and more..
  • In CSE250
Lecture Question

Task: Write a method to convert a BST to a List

• In the week8.trees.BinarySearchTree class write a method named toList that takes no parameters and returns the values of the tree in a List in sorted order