CSE 143
Binary Search Trees
reading: 17.3 – 17.4

IT’S A CHRISTMAS TREE WITH A
HEAP OF PRESENTS UNDERNEATH!

... WE’RE NOT INVITING YOU HOME NEXT YEAR.
Binary search trees

- **binary search tree** ("BST"): a binary tree where each non-empty node $R$ has the following properties:
  - elements of $R$'s left subtree contain data "less than" $R$'s data,
  - elements of $R$'s right subtree contain data "greater than" $R$'s,
  - $R$'s left and right subtrees are also binary search trees.
  ```java
  System.out.println(contains(42));
  ```

- BSTs store their elements in sorted order, which is helpful for searching/sorting tasks.
Adding to a BST

- Suppose we want to add new values to the BST below.
  - Where should the value 14 be added?
  - Where should 3 be added? 7?
  - If the tree is empty, where should a new value be added?

- What is the general algorithm?
Change point, version 2

- What is the state of the object referred to by \( p \) after this code?

```java
public static void main(String[] args) {
    Point p = new Point(1, 2);
    change(p);
    System.out.println(p);
}

public static void change(Point thePoint) {
    thePoint = new Point(3, 4);
}

// answer: (1, 2)
```

\[ p \rightarrow \begin{array}{c}
    x \ 1 \\
    y \ 2 \\
\end{array} \]

\[ x \ 3 \\
y \ 4 \]

// answer: (1, 2)
Changing references

- If a method *dereferences a variable* (with .) and modifies the object it refers to, that change will be seen by the caller.

  ```java
  public static void change(Point thePoint) {
    thePoint.x = 3; // affects p
    thePoint.setY(4); // affects p
  }
  ```

- If a method *reassigns a variable to refer to a new object*, that change will *not* affect the variable passed in by the caller.

  ```java
  public static void change(Point thePoint) {
    thePoint = new Point(3, 4); // p unchanged
    thePoint = null; // p unchanged
  }
  ```

- What if we want to make the variable passed in become null?
What is the state of the object referred to by \( p \) after this code?

```java
public static void main(String[] args) {
    Point p = new Point(1, 2);
    change(p);
    System.out.println(p);
}

public static Point change(Point thePoint) {
    thePoint = new Point(3, 4);
    return thePoint;
}

// answer: (1, 2)
```

Answer: (1, 2)
What is the state of the object referred to by \( p \) after this code?

```java
public static void main(String[] args) {
    Point p = new Point(1, 2);
    p = change(p);
    System.out.println(p);
}

public static Point change(Point thePoint) {
    thePoint = new Point(3, 4);
    return thePoint;
}

// answer: (3, 4)
```
If you want to write a method that can change the object that a variable refers to, you must do three things:

1. pass in the original state of the object to the method
2. return the new (possibly changed) object from the method
3. re-assign the caller's variable to store the returned result

```java
public static Point change(Point thePoint) {
    thePoint = new Point(99, -1);
    return thePoint;
}
```

We call this general algorithmic pattern \( x = \text{change}(x) \);

- also seen with strings: \( s = s\.toUpperCase() \);
Applying \( x = \text{change}(x) \)

- Methods that modify a tree should have the following pattern:
  - input (parameter): old state of the node
  - output (return): new state of the node

- In order to actually change the tree, you must reassign:

```plaintext
node = change(node, parameters);
node.left = change(node.left, parameters);
node.right = change(node.right, parameters);
overallRoot = change(overallRoot, parameters);
```