CSM 61B
Spring 2019

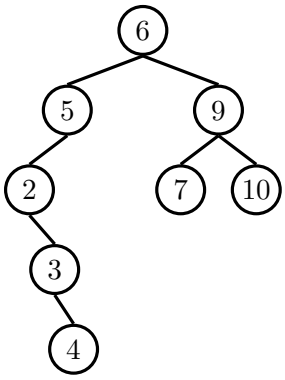
Trees & Hash

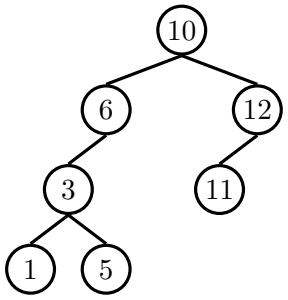
Mentoring 7: March 11, 2019

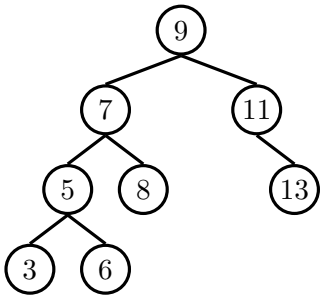
1 Binary Search Trees

```
public class BinarySearchTree<T extends Comparable<T>> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
}
```

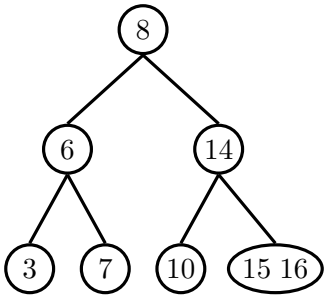
1.1 For each of the following binary search trees, determine if the height of the tree is the same as the height of the optimal tree with the same elements.



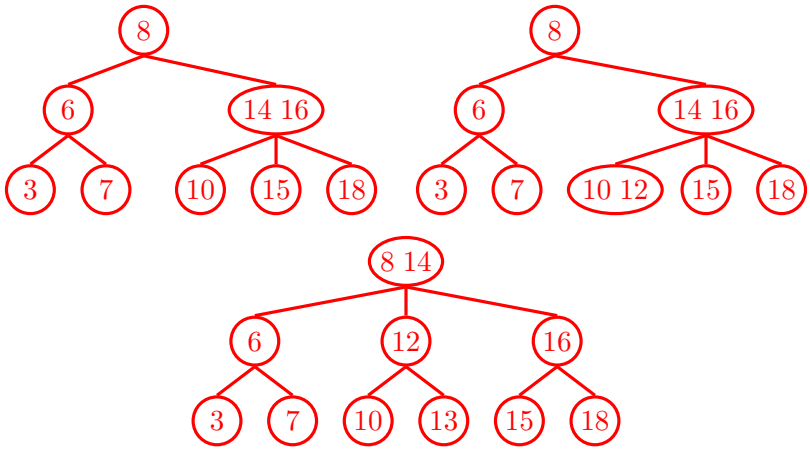




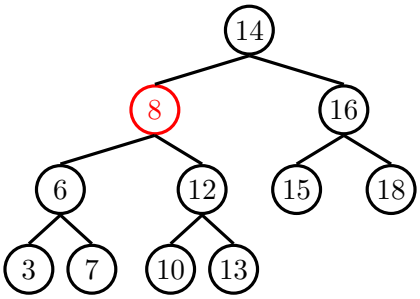
2 Balanced Trees



2.1 Draw what the 2-3 tree would look like after inserting 18, 12, and 13.



2.2 Now, convert the resulting 2-3 tree to a left-leaning red-black tree.



3 Hashing

3.1 (a) Draw the diagram that results from the following operations on a Java `HashMap`. `Integer::hashCode` returns the integer's value.

```
put(3, "monument");
put(8, "shrine");
put(3, "worker");
put(5, "granary");
put(13, "worker");
```

(b) Suppose a resize occurs, doubling the array to size 10. What changes?

4 Hash Codes

4.1 What does it mean for a hashcode to be valid?

4.2 Which of the following hashcodes are valid? Good?

```
class Point {
    private int x, y;
    private static int count = 0;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        count += 1;
    }
}
```

(a)

```
    public void hashCode() {
        System.out.print(this.x + this.y);
    }
```

(b)

```
    public int hashCode() {
        Random random = new Random();
        return random.nextInt();
    }
```

(c)

```
    public int hashCode() {
        return this.x + this.y;
    }
```

(d)

```
    public int hashCode() {
        return count;
    }
```

(e)

```
public int hashCode() {
    return 4;
}
```

5 Extra Practice: Trees

5.1 Given a node in a binary search tree (with parent pointers), implement `successor` which returns the next node in the in-order traversal of the BST. If there is no successor, return **`null`**.

```
public class BinarySearchTree<T extends Comparable<T>> {
    protected Node root;
    protected class Node {
        public T value;
        public Node parent, left, right;
    }
    private Node successor(Node node) {
```