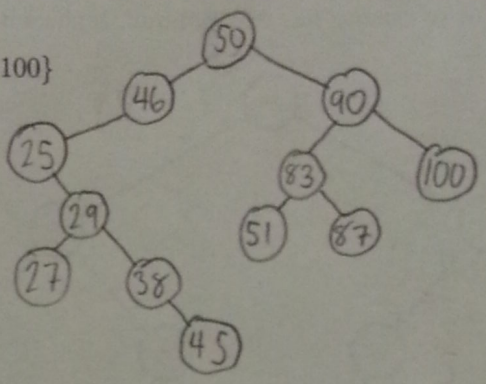
Key pg. 1

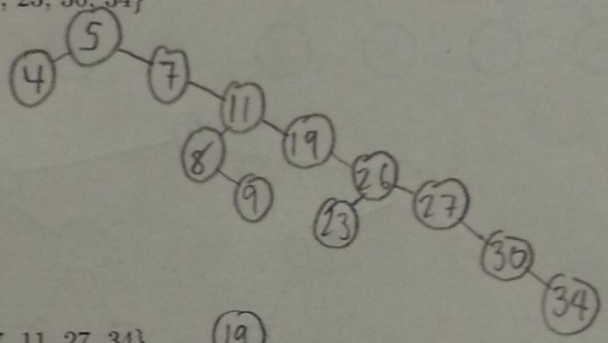
Recitation 4: Binary Search Trees

For problems 1-3, add the provided numbers, in the order given, to an empty Binary Search Tree. Draw the final state of the BST. You do not need to show intermediate steps.

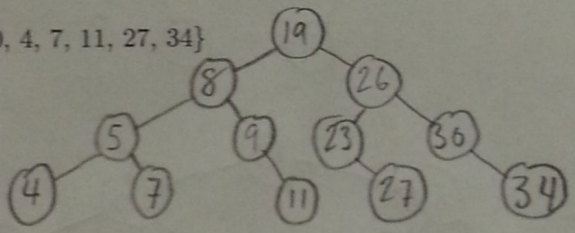
1. {50, 90, 83, 46, 87, 25, 29, 51, 38, 27, 45, 100}



2. {5, 7, 4, 11, 19, 8, 26, 27, 9, 23, 30, 34}



3. {19, 8, 26, 5, 9, 23, 30, 4, 7, 11, 27, 34}



4. a) What is the height of the tree you made in problem 2?

   7

   b) What is the height of the tree you made in problem 3?

   3

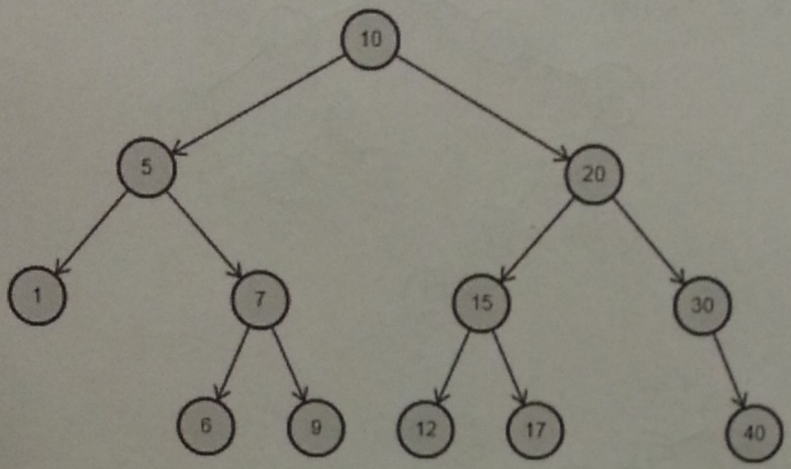
5. How does the order in which you add elements to a BST effect the efficiency of methods such as add(), remove(), and contains()?

If the elements are added in ascending or descending order, Big O of add(), remove(), and contains() becomes O(n), which is much worse than the O(log n) that the BST tries to achieve.

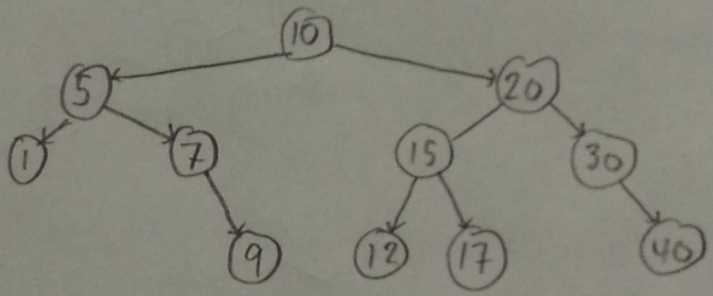
Key pg. 2

Recitation 4: Binary Search Trees

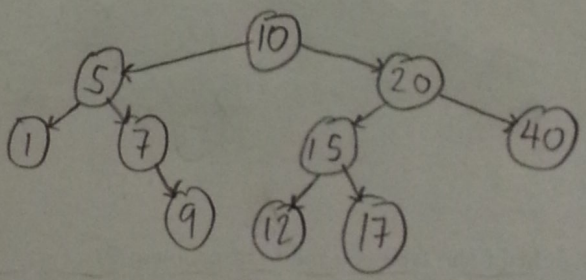
6. Redraw the given Binary Search Tree after each of the following operations. In the case that a node you want to remove has two children, replace it with the successor.



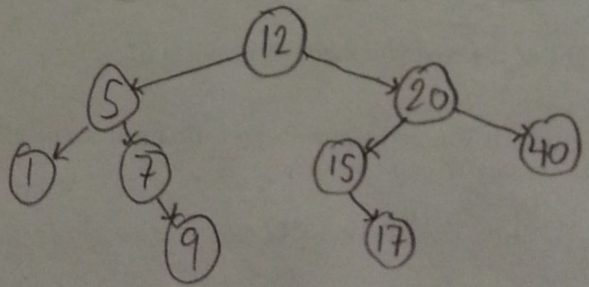
a) remove(6)



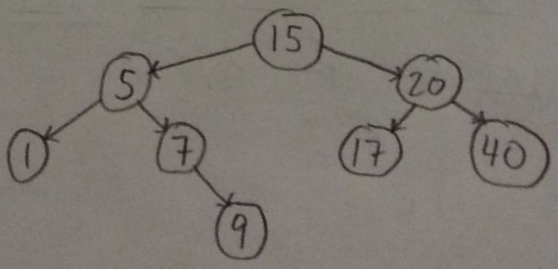
b) remove(30)



c remove(10)



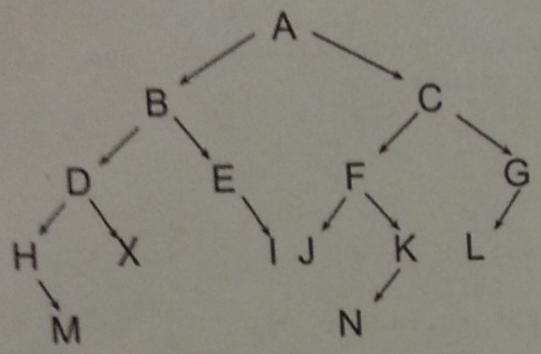
d) remove(12)



Key pg. 3

Recitation 4: Binary Search Trees

7. For each type of traversal, list the elements in the order they are traversed through.



a) Pre-Order Traversal: A B D H M X E I C F J K N G L

b) Post-Order Traversal: M H X D I E B J N K F L G C A

c) In-Order Traversal H M D X B E I A J F N K C L G

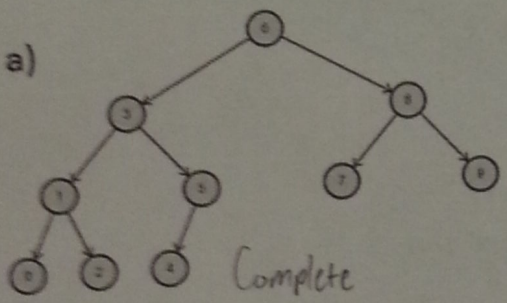
c) Level-Order Traversal A B C D E F G H X I J K L M N

8. A perfect BST (every row is completely filled) of height h contains how many nodes?

2^(h+1) - 1

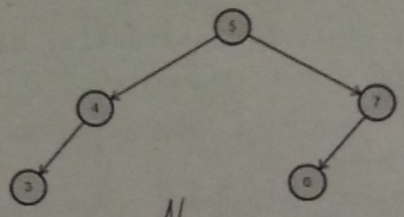
9. Identify whether the following trees are full, complete, and/or perfect.

a)



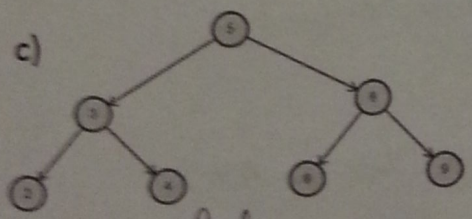
Complete

b)



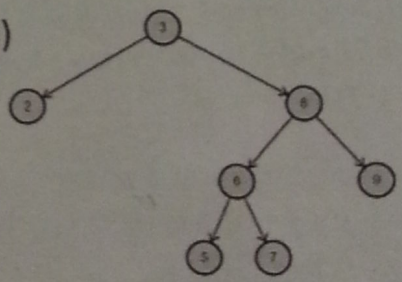
None

c)



Perfect

d)



Full

```
    private Node left;
    private Node right;
}

private Node root;

public void add(int data) { ... }
public void remove(int data) { ... }

/**
 * Determines if this tree contains the specified element.
 * @param data  value to be checked for containment in this tree
 * @return true if this tree contains the specified element
 */
public boolean contains(int data) {
    if: root is null, return false
    else: return containsHelper(data, root)
} // end contains
private boolean containsHelper(int data, Node curr) {
    if: data compared to curr.data = 0, return true
    else if: data compared to curr.data < 0 {
        if: curr.left != null, return containsHelper(data, curr.left)
        else: return false
    }
    else if: data compared to curr.data > 0 {
        if: curr.right != null, return containsHelper(data, curr.right)
        else: return false
    }
    return false

} // end containsHelper
}
```