Problem Set 8, Part I

Problem 1: Checking for keys below a value

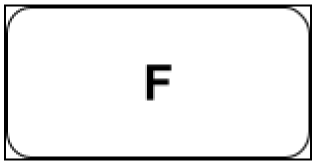
1-1) The efficiency of this method does not depend on the shape of the tree because it will check every single value in the worst case that the key is the last item. If this was implemented differently than O(logn) would be the average case, but in this case since it's always going to check each node until it finds the key. It has best-case O(1) in the case that the root is the key and a worst-case of O(n) if it has to check every single node within the tree.

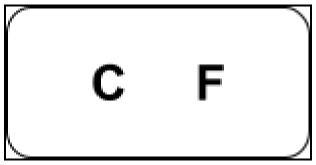
1-2)

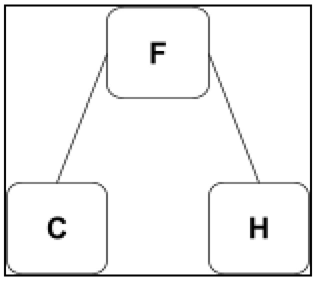
```
private static boolean anySmallerInTree(Node root, int v){
        if (root == null) {
            return false;
        }
       if(root.key >= v && root.left == null){
            return false;
        } // leaf node
        if(root.key < v){
            return true;
        }

       if(root.key >= v){
            boolean rest = anySmallerInTree(root.left);
            return rest;
        } // a value smaller than v, so we can return

        //return false;
    }
```

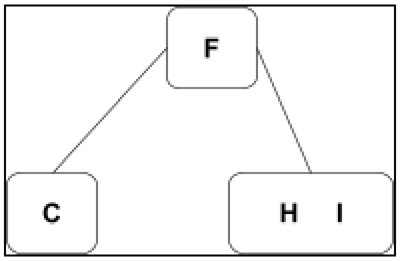
1-3) The big O notation for my algorithm is O(logn) for the worst case because only the entire left most branch will ever be used if its the last node, but in the best case it's O(1), because if the root node is less than the key, then the code will stop there. However, if the tree is not balanced, then it will be O(n), because it will essentially be a linked list.

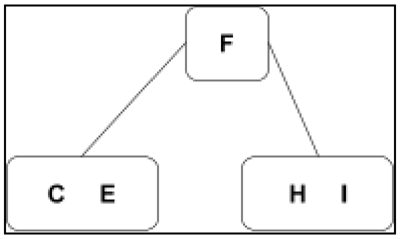
Problem 2: Balanced search trees

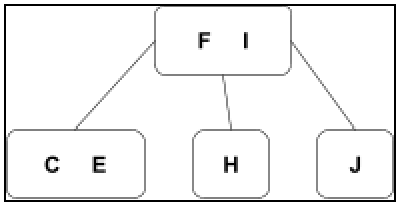

F

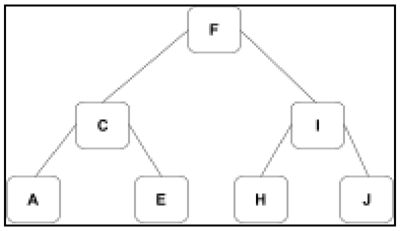

C F

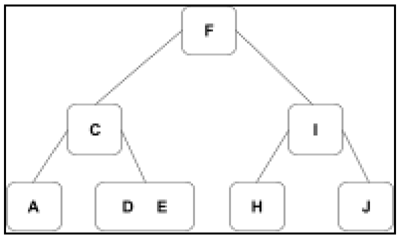

F
C
H

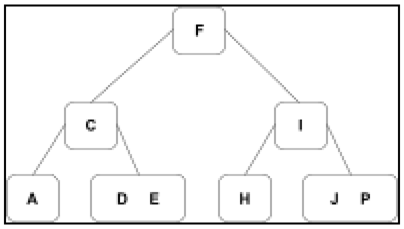

F
C
H I

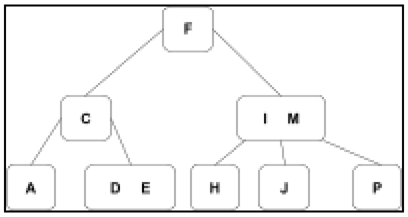

F
C E
H I


F I
C E
H
J


F
C
I
A
E
H
J


F
C
I
A
D E
H
J


F
C
I
A
D E
H
J P


F
C
I M
A
D E
H
J
P

Problem 3: Hash tables

	3-1) linear	**3-2) quadratic**	**3-3) double hashing**
0	`ant`		`ant`
1	`flea`		`bat`
2	`bat`		`flea`
3	`cat`	`cat`	`cat`
4	`goat`	`goat`	`goat`
5	`dog`	`bird`	`bison`
6	`bird`	`bison`	`bird`
7	`bison`	`dog`	`dog`

3-4) probe sequence:3,0,5,2

3-5) table after the insertion:

0	`bobcat`
1	
2	`eel`
3	`fly`
4	
5	`koala`
6	
7	`penguin`

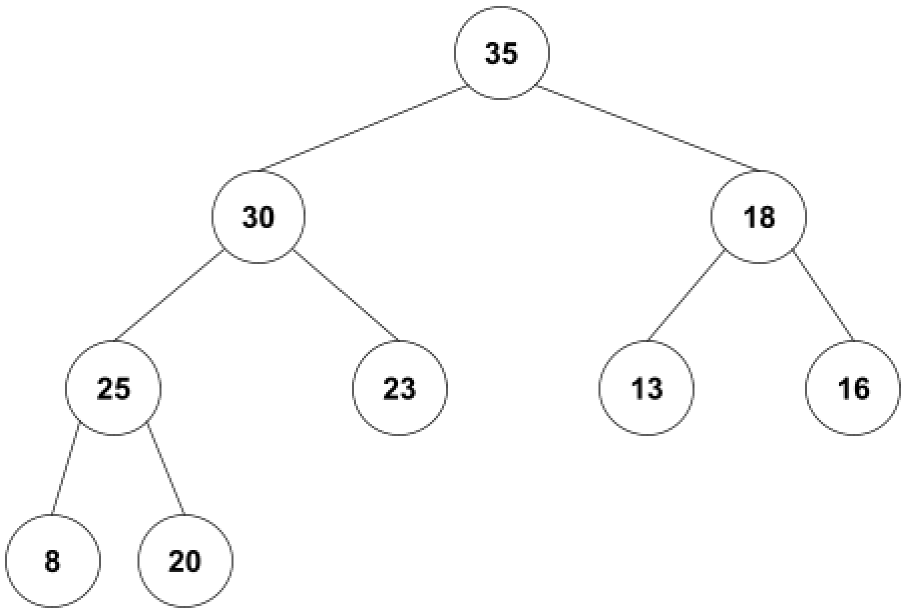
Problem 4: Complete trees and arrays

4-1) left child: 2r+1 if 2r + 1 < n == 2*23 + 1 = 47
right child: 2r+2 if 2r+2 < n == 2*23 + 2 = 48
parent:(r-1)/2 → (23-1)/2 == 11

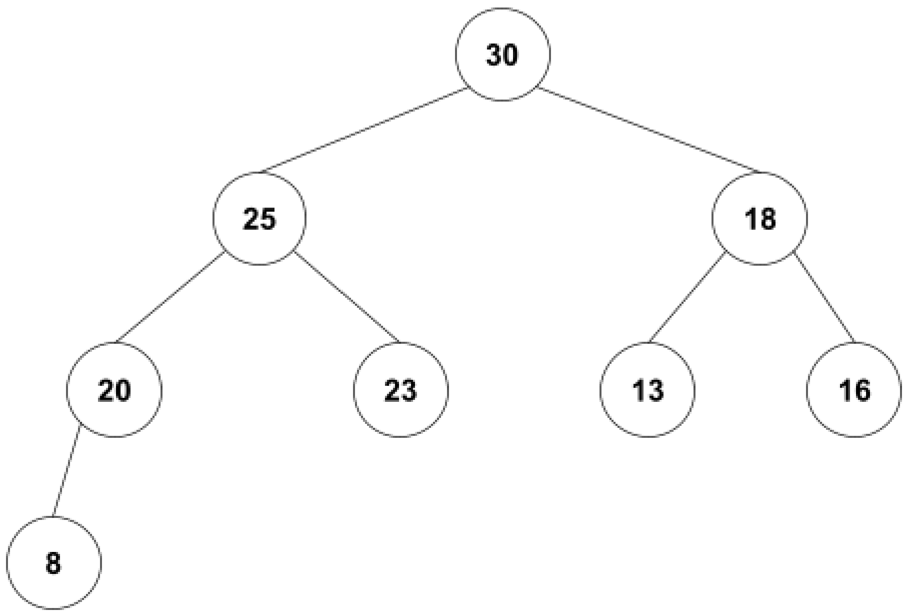
4-2) log_2(2023) = 10——-------------> , the equation for the height of a tree is log base 2 of n, where n is the number of nodes. Flooring this function returns the height of the tree.

4-3) 2022 right child. The last index holds a value of 2022, because the length of the array is 2023 and the actual last index holds a value of 2022, and since this is even, it's a right child.

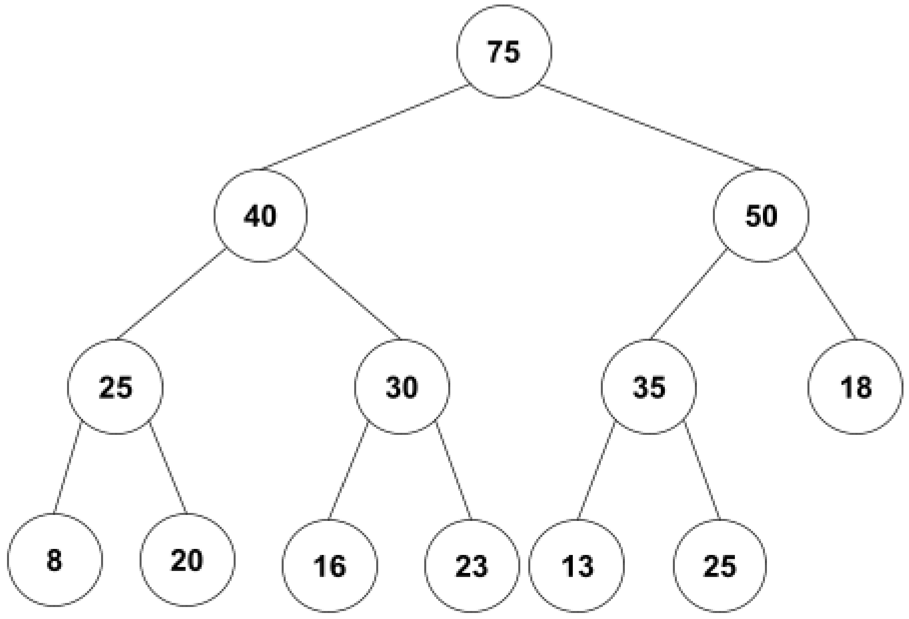
Problem 5: Heaps
5-1)
after one removal



after a second removal
(copy your revised diagram from part 1 here, and edit it to show the result of the second removal)


30
25
18
20
23
13
16
8

5-2)


75
40
50
25
30
35
18
8
20
16
23
13
25