**Assignment/Homework #2**
**COP-3530 Data Structures, Summer C 2018 PID: 3816842**

**Problem #1: (35 pts)**
In this problem, you will write some **Java code** for simple operations on **binary search trees** where keys are integers. Assume you already have the following code and assume that the method bodies, even though not shown, are correct and implement the operations as we defined them in class.

```
public class BinarySearchTreeNode
{
        public int key;
        public BinarySearchTreeNode left;
        public BinarySearchTreeNode right;
}
public class BinarySearchTree
{
        private BinarySearchTreeNode root;
        public void insert(int key) { ... }
        public void delete(int key) { ... }
        public boolean find(int key) { ... }
}
```

a) Add a method **public int keySum()** to the BinarySearchTree class that returns the sum of all the keys in the tree. You will need an assistant/helper method.
Please review java implementation

b) Add method **public void deleteMin()** to the BinarySearchTree class that deletes the minimum element in the tree (or does nothing if the tree has no elements).
Please review java implementation

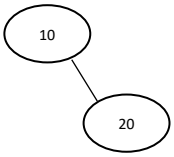
c) Add method **public void printTree()** to the BinarySearchTree class that iterates over
the nodes to print then in increasing order. So the tree...Produces the output "1 2 3 4 5".Note: You will need an assistant/helper method.
Please review java implementation

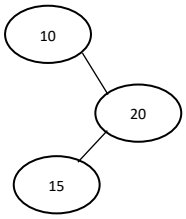
d) Add method **public void printPostorder()** to the BinarySearchTree class that prints out the nodes of the tree according to a "postorder" traversal. So the tree... Produces the output "1 3 2 5 4". Note: You will need an assistant/helper method.
Please review java implementation

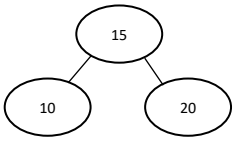
**Assignment/Homework #2**
**COP-3530 Data Structures, Summer C 2018 PID: 3816842**

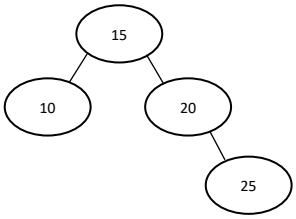
**Problem #2: (35 pts)**

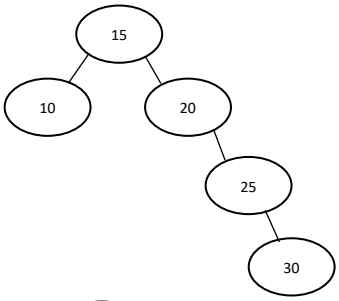
a) Show the result of inserting the following sequence of elements into an initially empty **AVL-tree**: 10, 20, 15, 25, 30, 16, 18, 19.

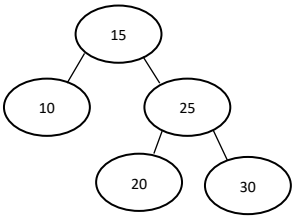


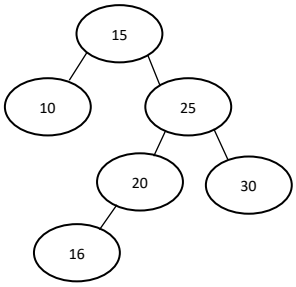



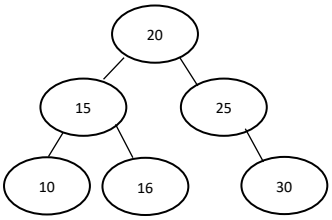



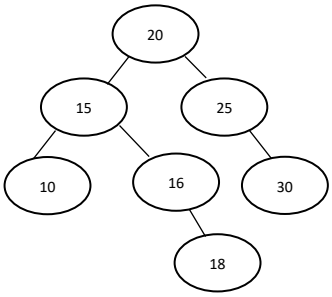



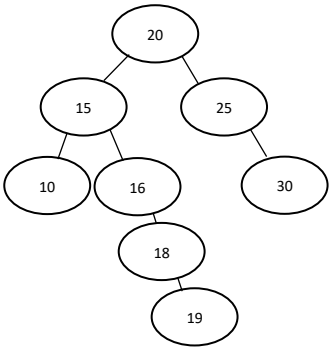



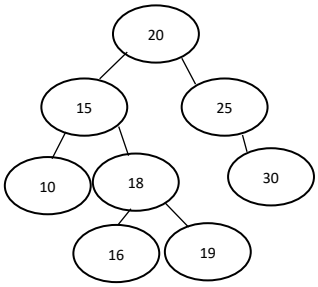



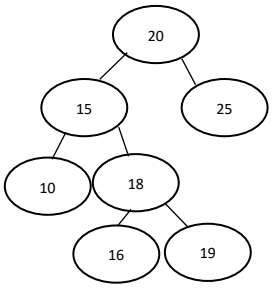



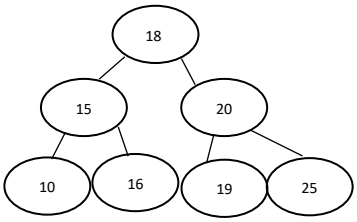



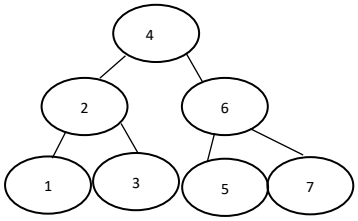




b) Show the resulting AVL-tree, after physical deletion (NOT a lazy deletion) of the record with the key 30 from the AVL tree that you got in the previous exercise.



c) Show the result when an initially empty AVL tree has keys 1 through 7 inserted in order.




d) Draw an AVL tree of height 4 that contains the minimum possible number of nodes.

S(h), is an AVL tree height h

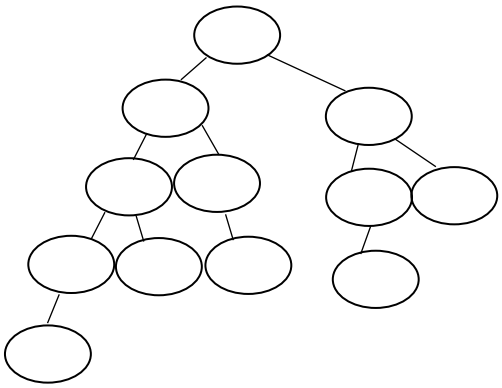
S(h) = S(h-1) + S(h-2) +1

S(4) = S(3) + S(2) + **1** = **(7+4+1)** = 12

S(2) + S(1) + 1 = **7**

S(1) + S(0) + 1 = **4**

S(1) + S(0) + 1 = **4**



**Assignment/Homework #2**
**COP-3530 Data Structures, Summer C 2018 PID: 3816842**

**Problem #3: (30 pts)**
(a) Consider a binary min-heap. We have a method that print the keys as encountered in a **preorder** traversal. Is the output sorted? Justify your answer. Attempt the same question for **inorder** and **postorder** traversal.

No, because a binary-hep is not a binary search tree and thus does not fall within the 'tree' domain for maintaining order of Left-Tree < Root < Right-Tree. There is no guaranty that smaller nodes will all be placed to the left tree and vice-versa.

The properties of a binary-hep are:

- Complete binary tree

-The key of a node <= the keys of the children.

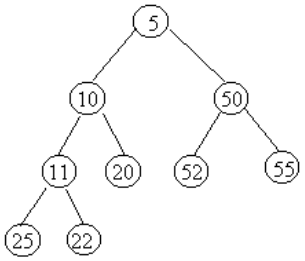
The binary-hep operations do not consist of any type of traversals, only findMin/deleteMin operations, which will return the minimum values in order with a O(1) time complexity.

However, we can traverse the Heap by the following.

Parent index: floor((i - 1) / 2)

Child indices: 2 * i + 1, 2 * i + 2

*Given the following min heap*




| 5 | 10 | 50 | 11 | 20 | 52 | 55 | 25 | 22 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Tree Traversals not Heap Traversal**

Preorder: 5, 10, 11, 25, 22, 20, 50, 52, 55

Inorder: 25, 11, 22, 10, 20, 5, 52, 50, 55

Postorder: 25, 22, 11, 20, 10, 52, 55, 50, 5

(b) Propose an efficient algorithm (only pseudocode) to find all nodes less than some value X in a binary min-heap. Analyze its complexity.

I propose to start from the root the move recursively across the array because checking the children of each node given: 2 * i + 1, 2 * i + 2 . The root in known to store the minimum value within the binary-hep. If the roots values is greater than X we can achieve constant time of O(1) by only checking the root(one element).

Best case: The root is greater than then x O(1) or the set is empty

Worst case: x is greater than every node. O(N).

NOTE: The root of the tree is at A[1], i.e., the indexing typically begins at index 1 (not 0). A[0] can be reserved for the variable heap-size[A]

```
//Check all children of nodes to find all values < X starting with root
// i=0, x = integer value

printMinimumValues( heap, i, x)

        //Check root first
        if i is greater or equal to heap.size
                STOP, We have checked the entire heap
        if, heap[ i ] greater or equal to x
                STOP, X We don't need to continue looking

        //If we made it here, the value of x < heap's current element. Time to print.
        Print the current index i

        //Check the left and right child
        printMinimumValues( heap, 2 * i + 1 , x)
        printMinimumValues( heap, 2 * i + 2, x)
```

Base Case

**Assignment/Homework #2**
**COP-3530 Data Structures, Summer C 2018 PID: 3816842**

(c) Write (in **Java**) a method **replaceKey** in the **MaxHeap** class with the following signature:

**public void replaceKey(Integer oldKey, Integer newKey)**
The method will replace the first occurrence of **oldKey** with the **newKey**, and restore the **Max-Heap** property after the change. If the **oldKey** does not exist in the heap, the method prints an appropriate message and returns without changing the heap.

Example: Suppose our binary heap object (**bh**) has the following keys:

*** 99 64 42 54 32 28 6 19 7 26 4
Then the method call: **bh.replaceKey (new Integer(54), new Integer(105)** should change the keys to:

*** 105 99 42 64 32 28 6 19 7 26 4
*Note:* You can assume that the methods **perlocateUp** and **perlocateDown** are already implemented in your **MaxHeap** class.

Please review java implementation