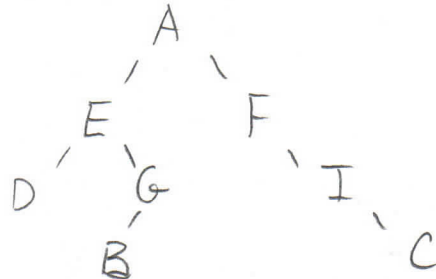


1. [3] Given the following pre-order and in-order traversals, reconstruct the appropriate binary tree. **NOTE: You must draw a single tree that works for both traversals.**

Pre-order: A, E, D, G, B, F, I, C

In-order: D, E, B, G, A, F, I, C



2. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the left subtree when the node to remove is full.

Insert(5), Insert(10), Insert(2), Insert(9), Insert(1), Insert(3), Remove(5).

root \rightarrow nullptr \Rightarrow root \rightarrow 5 \Rightarrow root \rightarrow 5, 10


\Rightarrow root \rightarrow 5, 2, 10 \Rightarrow root \rightarrow 5, 2, 10, 9

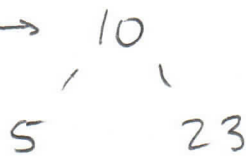
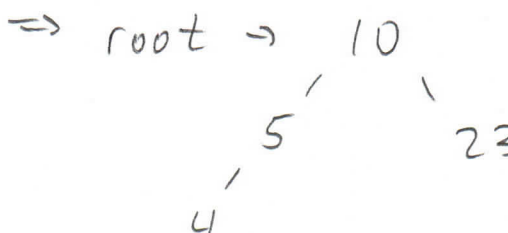
\Rightarrow root \rightarrow 5, 2, 10, 1, 9 \Rightarrow root \rightarrow 5, 2, 10, 1, 3, 9

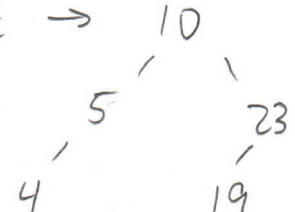
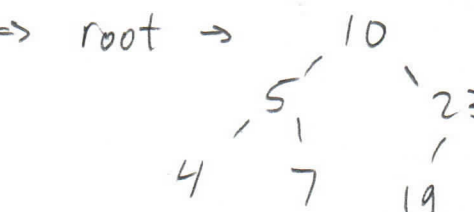
\Rightarrow root \rightarrow 3, 2, 10, 1, 9

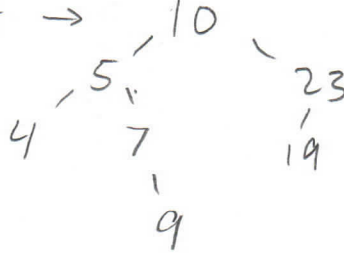
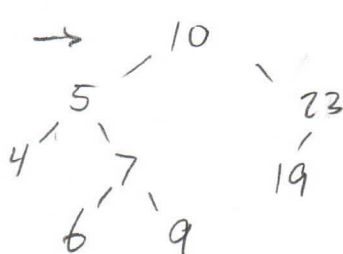
3. [3] Starting with an empty BST, draw each step in the following operation sequence. Assume that all removals come from the right subtree when the node to remove is full.

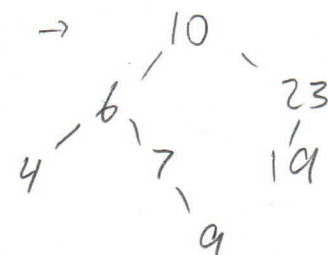
Insert(10), Insert(5), Insert(23), Insert(4), Insert(19), Insert(7), Insert(9), Insert(6), Remove(5).

root \rightarrow nullptr \Rightarrow root \rightarrow 10 \Rightarrow root \rightarrow 

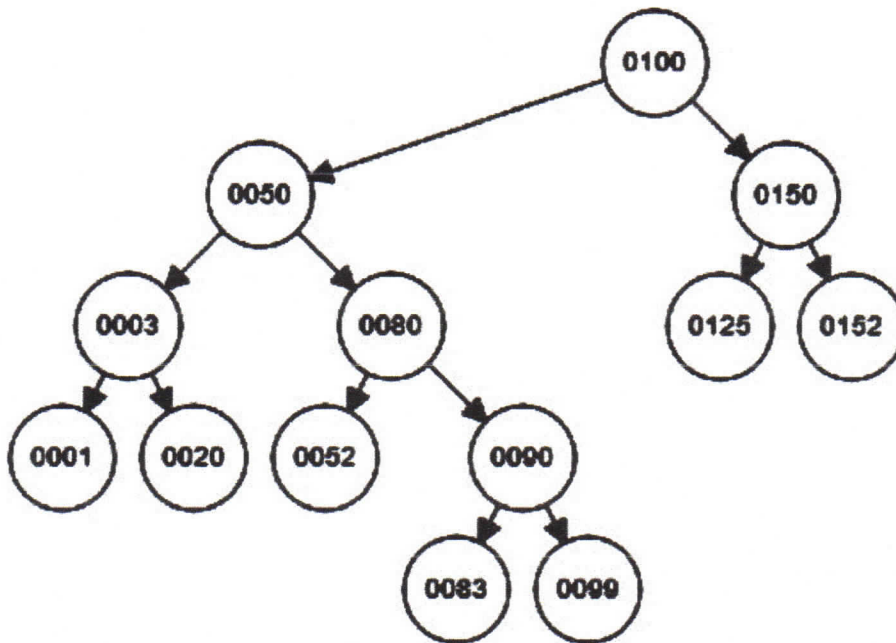
\Rightarrow root \rightarrow  \Rightarrow root \rightarrow 

\Rightarrow root \rightarrow  \Rightarrow root \rightarrow 

\Rightarrow root \rightarrow  \Rightarrow root \rightarrow 

\Rightarrow root \rightarrow 

4. Given the following binary tree (where nullptr height == -1):



A. [1] What is the height of the tree?

4

B. [1] What is the depth of node 90?

3

C. [1] What is the height of node 90?

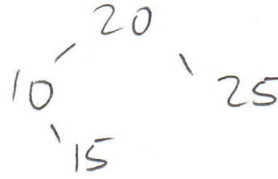
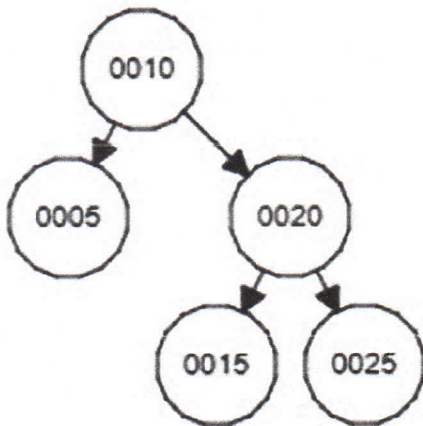
1

D. [3] Give the pre-order, in-order, and post-order traversal of this tree.

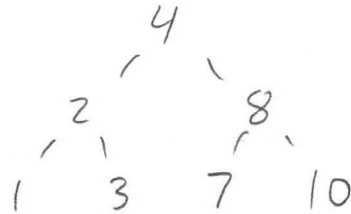
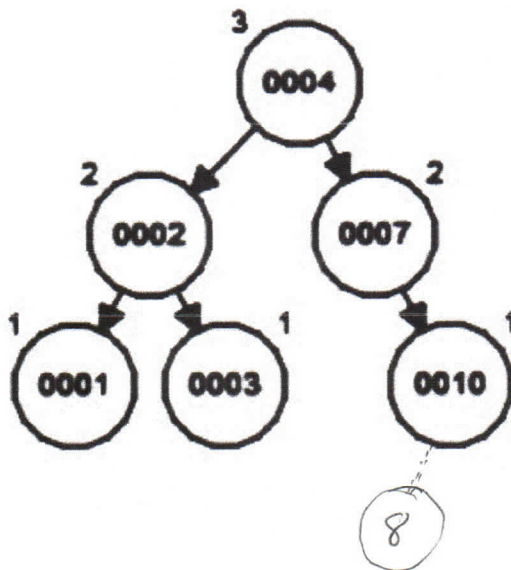
omitting leading zeroes b/c this is a long HW

Pre 100, 50, 3, 1, 20, 80, 52, 90, 83, 99, 150, 125, 152
 In 1, 3, 20, 50, 52, 80, 83, 90, 99, 100, 125, 150, 152
 Post 1, 20, 3, 52, 83, 99, 90, 80, 50, 125, 152, 150, 100

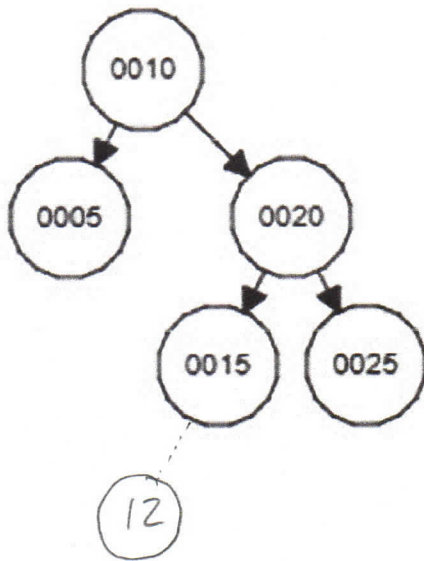
5. [3] Remove 5 from the following AVL tree; draw the results:



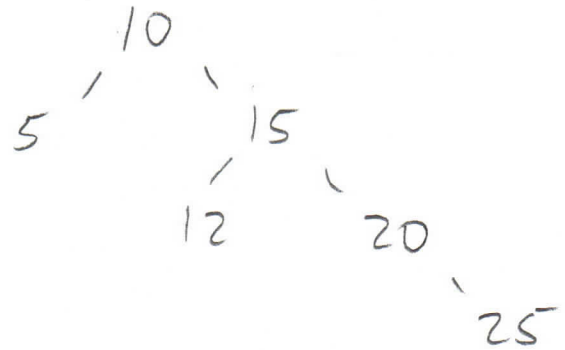
6. [3] Insert the value "8" into the following AVL tree; draw the result:



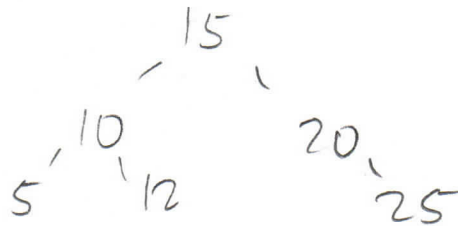
7. [3] Insert the value "12" into the following AVL tree; draw the result:



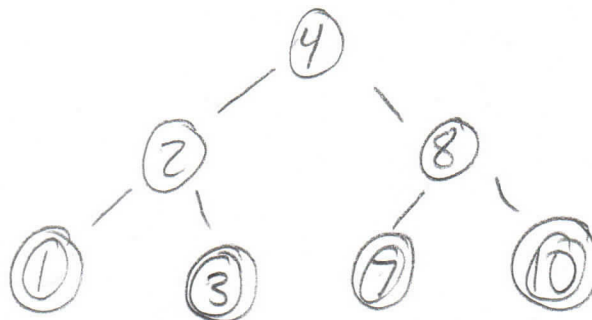
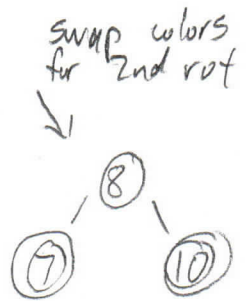
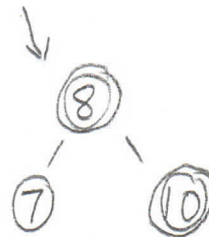
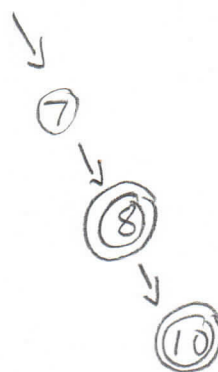
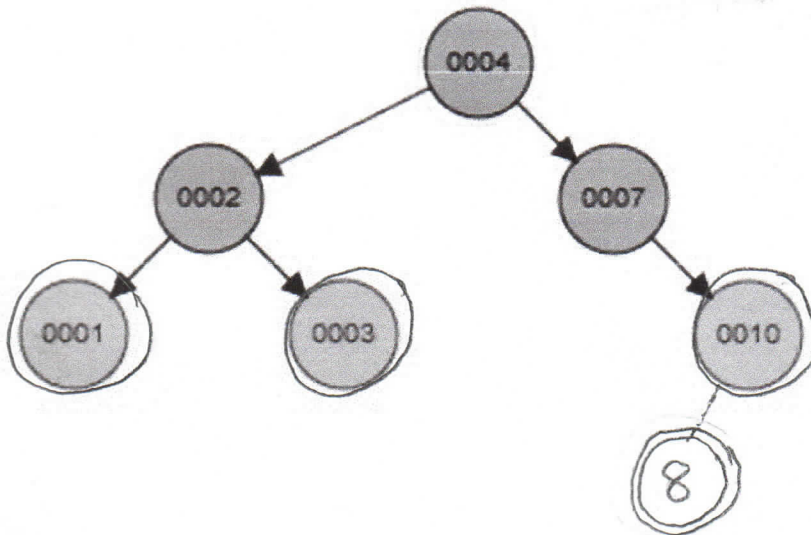
Intermediate step:



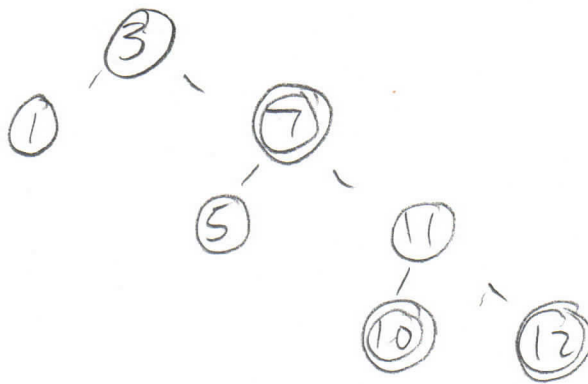
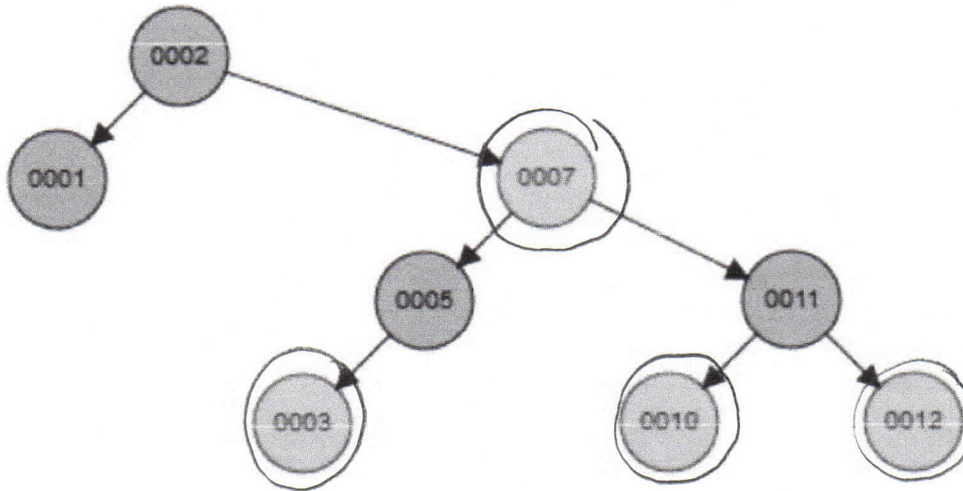
Final:-



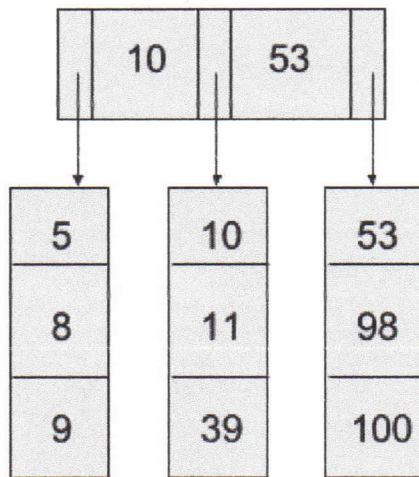
8. [3] Insert the value "8" into the following Red-Black tree; draw the result. Use Double-circle to denote red nodes and single circle to denote black nodes.



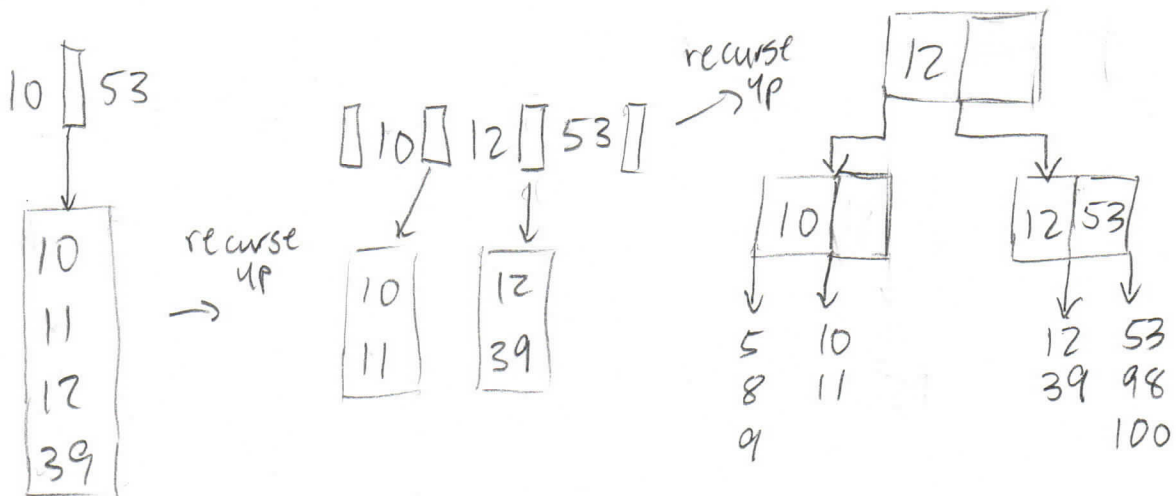
9. [3] Delete the value "2" from the following Red-Black tree; draw the result. Use Double-circle to denote red nodes and single circle to denote black nodes.



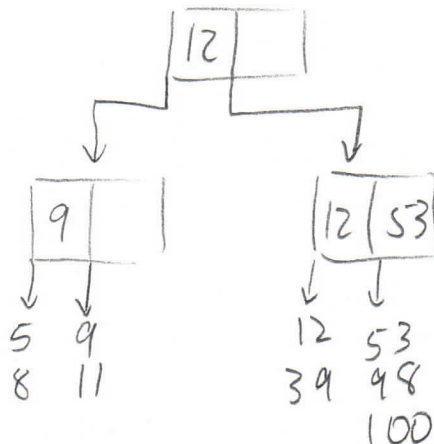
10. [4] Given the following B+ tree ($M = 3$, $L = 3$):



A) Insert 12 into the tree and draw the resulting B+ Tree:



B) Based on the tree resulting from part (A), now remove 10 and draw the new tree:



11. [6] We are going to design our B+ Tree to be as optimal as possible for our old hard drives (since the management won't buy new ones, those cheapskates!). We want to keep the tree as short as we can, and pack each disk block in the filesystem as tightly as possible. We also want to access our data in sorted order for printing out reports, so each leaf node will have a pointer to the next one. See figure #1 on next page for a visualization of our tree.

CPU architecture: Intel Xeon with 64 bit cores $\rightarrow 8$ byte ptrs
 Filesystem: Ext4 with 4KB (4096 byte) blocks
 The customer records are keyed by a random UUID of 128 bits

Customer's Data record definition from the header file:

```
#include <uuid>
struct CustomerData {
    uuid_t uuid;           // Customer 128 bit key            $\rightarrow 16$  bytes
    char[32] name;         // Customer name (char is 1 byte each)  $\rightarrow 32$  bytes
    uint32_t ytd_sales;     // Customer year to date sales  $\rightarrow 4$  bytes
};
```

~~8~~

Calculate the size of the internal nodes (M) for our B-tree:

$$\begin{aligned}
 &16(M-1) \text{ bytes worth of keys} \\
 &8M \text{ bytes worth of ptrs} \\
 &= 24M - 16 \text{ available to stick in } 4096 \text{ byte block} \\
 &M = \lfloor 171.33 \rfloor = 171
 \end{aligned}$$

Calculate the size of the B-tree leaf nodes (L) for this tree make sure to include the pointer (note CPU architecture!) to keep the list of leaf nodes:

$$\begin{aligned}
 &16 + 32 + 4 = 52 \text{ bytes per leaf element} \\
 &\text{leaf node size } 52L + 8 \leq 4096 \\
 &L = \lfloor 78.6 \rfloor = 78
 \end{aligned}$$

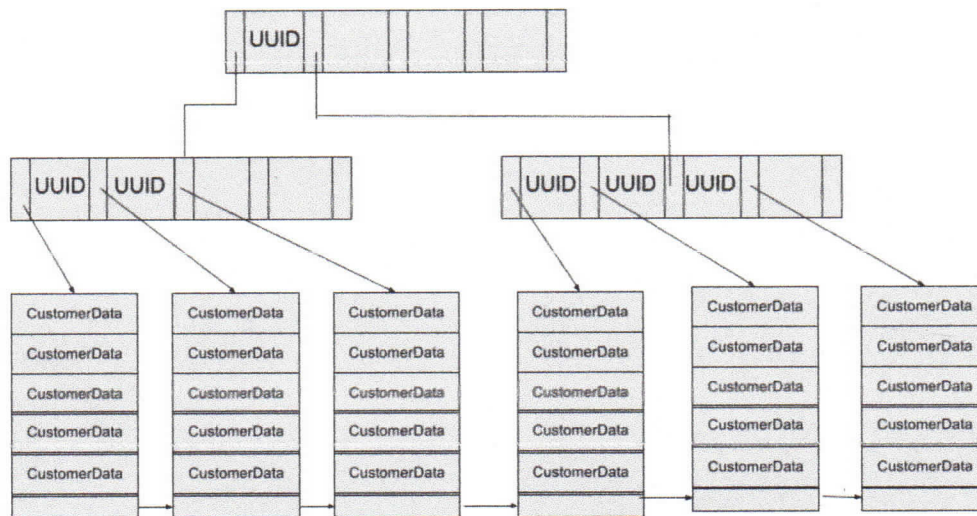


Figure #1: Visualization of our B+ Tree of height 2, customer data records, and pointers between the leaf nodes.

How tall (on average) will our tree be (in terms of M) with N customer records?

$$M = \log_{171} (N)$$

If we insert 30,000 CustomerData records, how tall will be tree be?

$$\log_{171} (30,000) \approx 2$$

If we insert 2,500,000 customers how tall will the tree be?

$$\log_{171} (2.5 \times 10^6) \approx 2.86 = 3$$