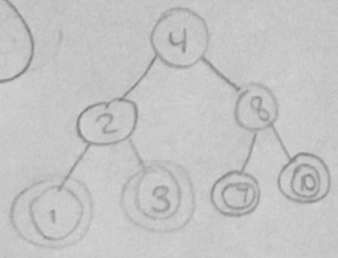
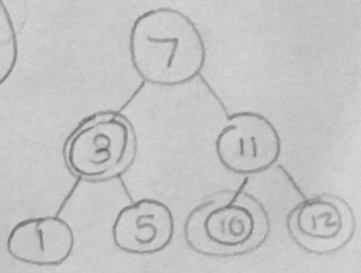
4)

| AVL | Red Black |
|---|---|
| Insert $O(\log_2 n)$<br>AVL is slower since AVL is balanced more strictly than RB trees so more rotations are required | $O(\log_2 n)$<br>Faster than AVL since fewer rotations are done to insert. The black height only has to be the same on each side so the balancing is less strict and less rotations are made. |
| Delete $O(\log_2 n)$<br>Slower than RB for same reason as insertion | $O(\log_2 n)$<br>Faster than AVL for same reason as insert. |
| Search $O(\log_2 n)$<br>AVL is faster because the balancing is more strict. The number of nodes checked to the furthest node is less than RB tree because of that. | $O(\log_2 n)$<br>Since the black nodes are what ensures the tree is balanced, the tree can be more skewed if more red nodes are on one side than the other so searching can take a little longer than AVL. |

5)




6)




7)

```
rbtree insert (rbtree* root, int data){
    if (root == root->root){
        make new node w/ data;
        node color = black;
        return new node;
    }
    else if (root == NULL){
        make new node w/ data
        node color = red;
        return new node;
    }
    else{
        if (root->data > data){
            root->left = insert(root->left, data);
        }
        else{
            root->right = insert (root->right, data);
        }
        // recolor
        if (previous color == red && previous->previous color == black){
            previous is set to black;
            previous sibling is set to black;
            grandparent is set to red;
        }
        if (this is root && also not black){
            set to black);
        }
```

//continued up here

```
        if (sibling is red and root is red){
            parent color = black
            grandparent color = red
            sibling color = black);
        }
        else if (parent is left of grandparent){
            if (root is right of parent){
                parent color = black
                grandparent color = red
                rotate right (grandparent);
            }
        else if (parent = right of grandparent){
            if (root is left of parent)
                set color of parent to black
                set color of parent to red
                rotate left (grandparent);
        }
        return root;
    }
}
```