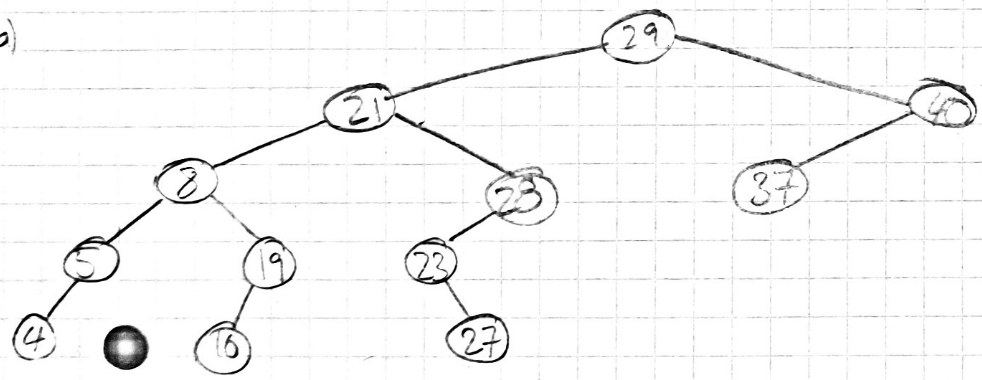
Abraham Marciano Data Structures HW6

1)
```
path longestPath (node* n){
        path p;
        if(n == NULL) return p;
        p.append(n);
        path l= longestPath (n->left);
        path r= longestPath (n->right);
        if (l.head->val < n->val) l = path (NULL);
        if (r.head->val < n->val) r = path (NULL);
        p.append (l.size() >= r.size() ? l : r);
        return p;
}
```

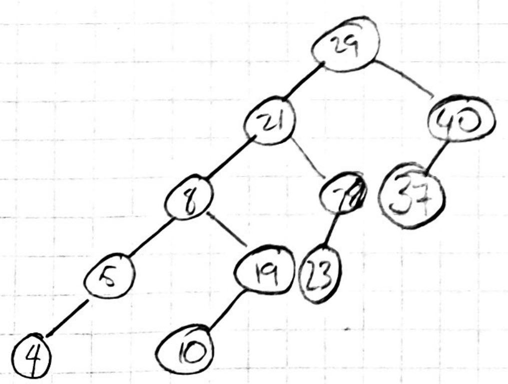
The complexity of function longestPath is $O(n)$ where n is the number of nodes in the binary tree. This is O ∵ the function will run once for each node

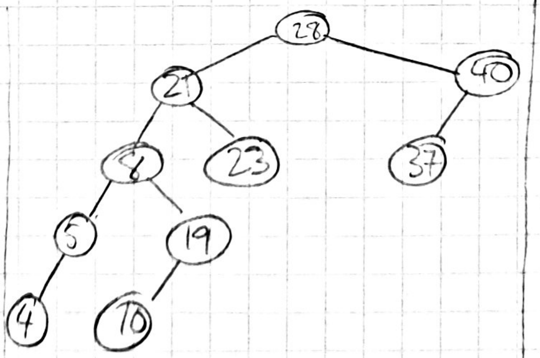
2)
```
int nTrees (int n){
        if(n == 0){ return 0;}
        int sum = 0;
        for(int i = 0, j = n - 1; i < j; i++, j --){
            sum += (nTrees(i) + nTrees(j))* 2;
        }
        if (n % 2 == 1){
            sum += nTrees(n / 2);
        }
        return sum;
}
```

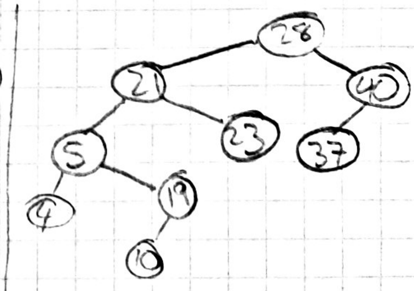
3)a)

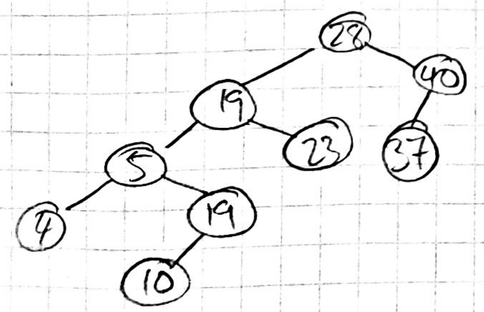


b)









4) a) $O(n^2)$ if you dont balance the tree each time you insert.
However, if you do balance the tree, it's $O(n\lg(n))$

b) the height of the binary tree will not exceed $\lg(n)$. Since the array is sorted the nodes can be inserted from top to bottom, and the tree will always be complete so to insert all nodes complexity is $O(n\lg(n))$.

5) a) the function checks if all the values of root2 are >= all the values of root1

b) Yes. for every node in tree1, it is compared to every node in tree2

c)
```
bool what(Node* root1, Node* root2){
    for(; root1 != NULL; root1 = root1->right){}
    for(; root2 != NULL; root2 = root2->left){}
    return root1->val <= root2->val;
}
```