1. Prove that, when a binary tree with n nodes is implemented by using links to the left and right child, then there will be a total of $n + 1$ null links.

Let $P(n)$ denote the statement that a binary tree with n nodes using links to the left and right child, then there will be a total of $n + 1$ null links. $P(1)$ is true since a binary tree with only one node,the root, will contain two null links. We assume the statement holds for all positive integers i less than or equal to k and show it must hold for $P(k + 1)$.

Case 1: root has one child. WLOG we assume it is the left child. by the IHOP the left subtree has k elements and thus has $k+1$ null links. the additional link from the root right null adds one extra giving us a total of $(k + 1) + 1$ null links which proves the $1st$ case.

Case 2: root has both children. we assume the left subtree has $k-i$ elements and the right subtree has k elements where $0 \leq i \leq k$ by the IHOP the left subtree has $k-i+1$ null links and the right subtree has $i+1$ null links, together there are a total of $(k + 1) + 1$ total null links which proves the $2nd$ case.

Since both cases hold $P(k + 1)$ holds. By the principle of mathematical induction the result holds for all integers $n \geq 1$.

2. Prove that the maximum number of nodes in a binary tree with height h is $2^{h+1} - 1$.

The maximum number of nodes in a binary tree of height h is the case where the binary tree is perfect. In this case ever level is filled completely and there are 2^i nodes for a level where i ranges from 0 to h. Summing all the nodes at every level we obtain the summation $\sum_{i=0}^{h} 2^i$ which evaluates to $2^{h+1} - 1$. We conclude that the maximum number of nodes in a binary tree with height h is $2^{h+1} - 1$.

3. A full node for a binary tree is one that has two children. Prove that the number of full nodes plus one equals the number of leaves of a binary tree.

We prove by induction. If a tree has one node, the root, then it has 0 full nodes which haves 1 leaf, the node itself. we assume that a binary tree has n full nodes then it must have $n + 1$ leaves we prove $P(n + 1)$. We find the leaf that has a full node as a parent. If we cannot find a leaf, we remove a leaf l. This does not reduce the number of leaves or full nodes. We continously do so until we reach the case where the leaf has a full node as a parent. we remove the leaf. there are now n full nodes and by the IHOP $n + 1$ leaves. adding back the leaf we have $n + 1$ full nodes and $(n + 1) + 1$ leaves which is what we wanted to show. Hence by the principle of mathematical induction, the resul holds for all $n \geq 0$.

4. Using the Node class described in lecture. Implement the function boolean is_balance(Node n).

```
 1: balanced_and_height helper(node* n){
 2:         if(!n) return {true, -1};
 3:
 4:         auto left = helper(n->left);
 5:         if(!left.balanced) return {false, -1};
 6:
 7:         auto right = helper(n->right);
 8:         if(!right.balanced) return {false, -1};
 9:
10:         int height = max(right.height,left.height) + 1;
11:
12:         return {abs(left.height - right.height) <= 1, height};
13: }
14:
15: bool is_balanced(node* n){
16:         return helper(n).balanced;
17: }
```

5. Use the Node class described in lecture, and assuming and additional integer value attribute, implement the function boolean is_bst(Node n).

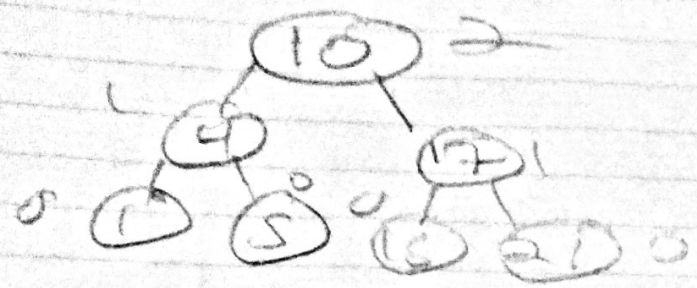
```cpp
bool helper(node* n, int max_left, int max_right){
        if(!n) return true;
        int val = n->val;
        return helper(n->left, max_left, val)
                && helper(n->right, val, max_right)
                && max_left < val && val < max_right;
}

bool is_bst(node* n){
        return is_bst(n, INT_MIN, INT_MAX);
}
```

6. Using the Node class described in lecture, and assuming an integer value attribute, implement the function void print(n) which has the effect of printing all the integer values stored in the tree rooted at n.

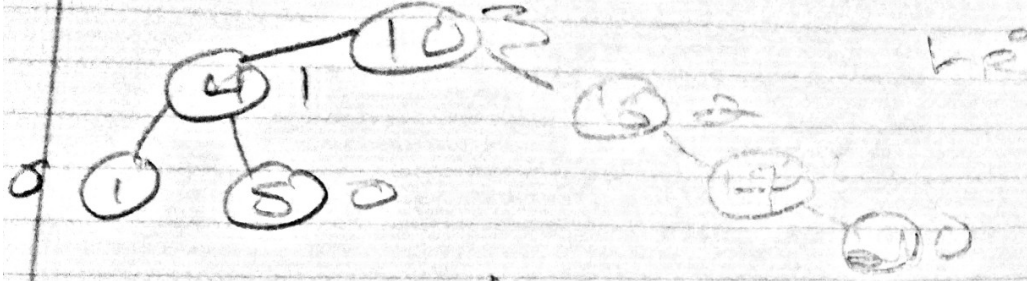
```
1: void print(Node* n){
2:         if(!n) return;
3:         print(n->left);
4:         cout << n->val << " ";
5:         print(n->right);
6: }
```

7. Suppose a binary search tree is to hold keys $1, 4, 5, 10, 16, 17, 21$. Draw possible binary search tree for these keys, and having height $2, 3, 4, 5$, and 6. For each tree, compute its internal path length.

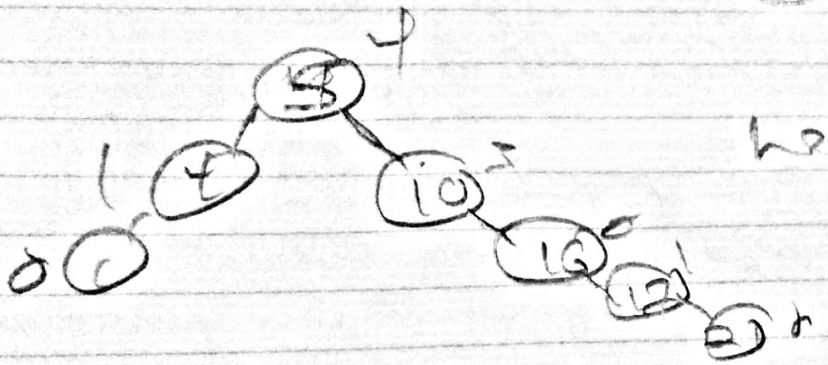
1, 4, 5, 10, 16, 17, 21



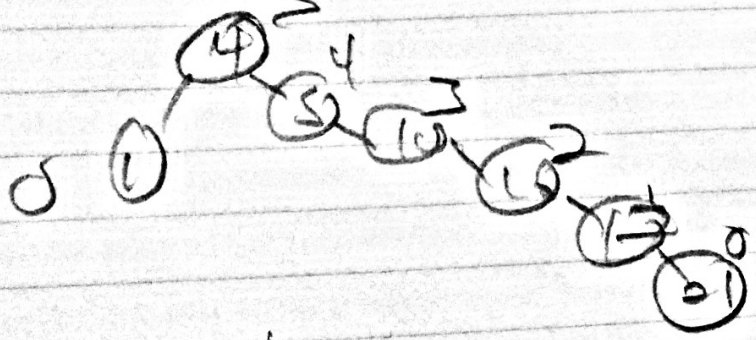
height 3



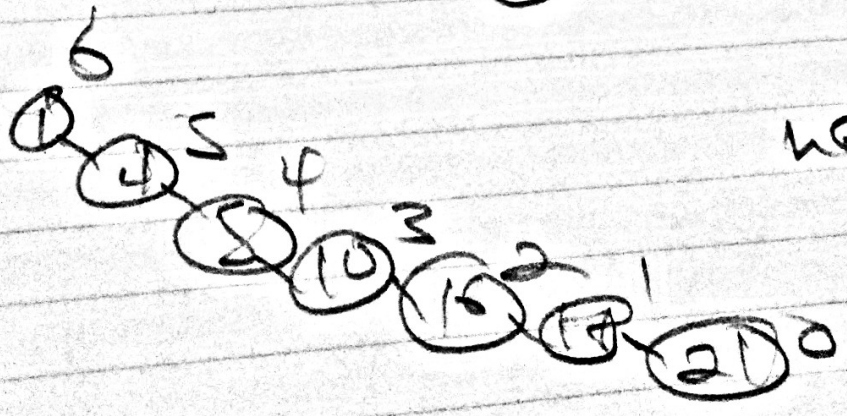
height 4



height 5



height 6



8. Prove that it takes $\Omega(nlog(n))$ steps in the best case to build a binary search tree having n keys.

In the best case the tree at the kth insertion has height $log(k)$. We obtain $\sum_{i=1}^{n} log(i) = \Omega(\int_1^n log(x)dx) = \Omega(n \cdot log(n))$.

9. Suppose we have a bst that stores keys between 1 and 1000. If we perform a find on key 363, then which of the followign sequences could not be the sequence of nodes examined when performing using a successful find() method provided in lecture.

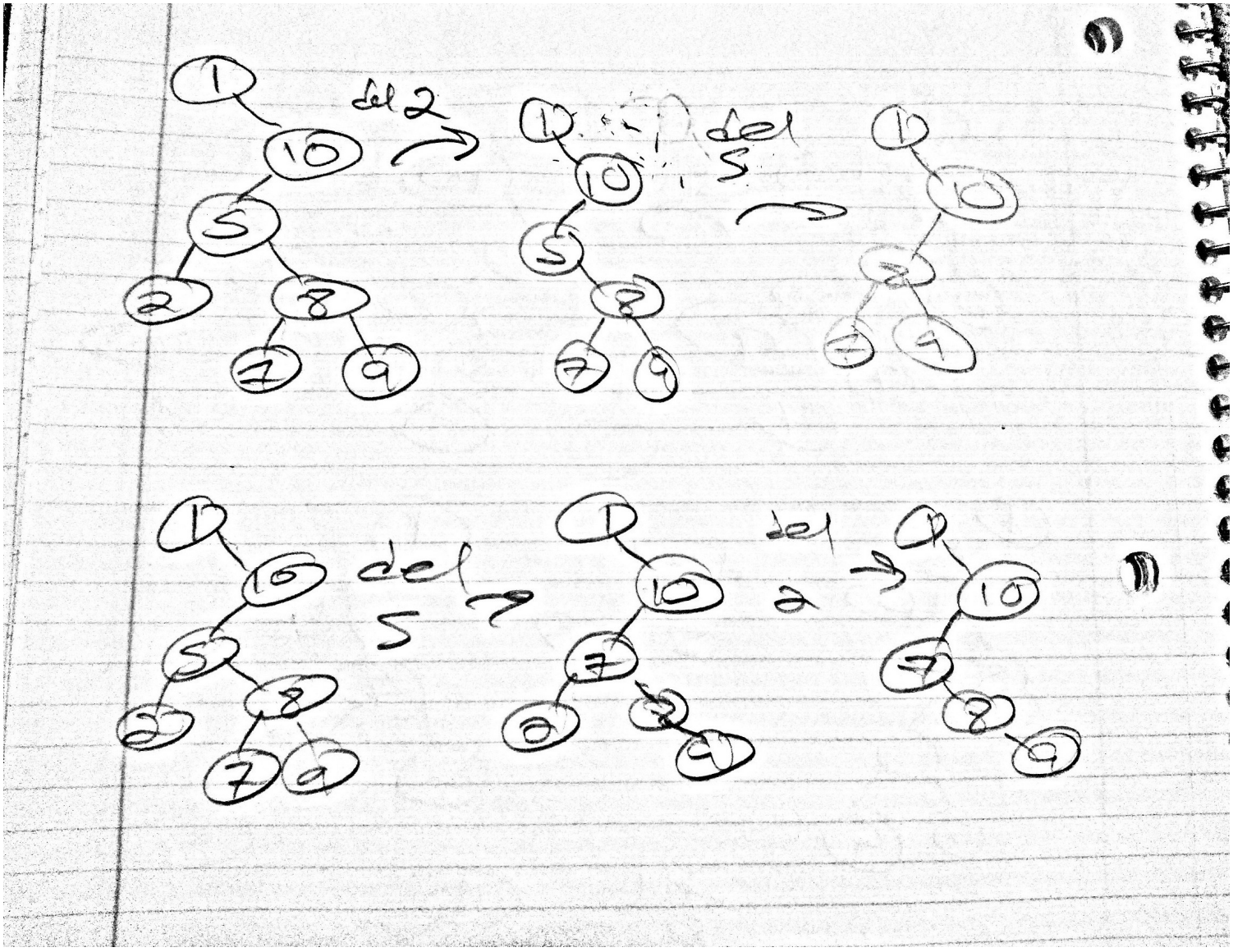
1. 2,252,401,398,330,344, 397
2. 924,220,911,244,898,258,362,363
3. 925,202,911,240,912,245,363
4. 2,399,387,219,266,382,381,278,363
5. 935,278,347,621,392,358,363

1 is not correct because 363 was not find. 3 is also incorrect because it is not a valid bst.

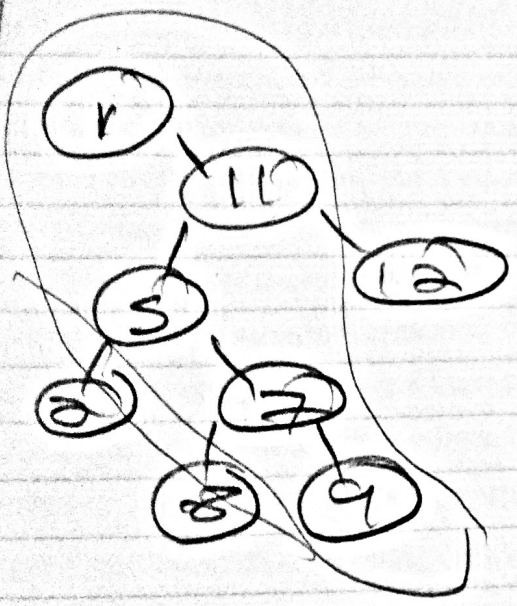
10. Suppose a bst is constructed by repeatedly inserting distinct keys into the tree. Argue that the number of nodes examined when searching for a key is equal to one more than the number examined when inserting that key.

Suppose the search has $k + 1$ path length. The search prior to the insertion of the node was of length k. The nodes was found and thus was inserted. It is the case then that the number of nodes examined when searching for a key is equal to one more than the number examined when inserting that key.

11. Prove or disprove: Deleting keys x and y from a bst is commutative. In other words, it does not matter which order the keys are deleted. The final trees will be identical. If true, provide a proof. If false, provide a counterexample.


del 2
del 5
del 5
del 2

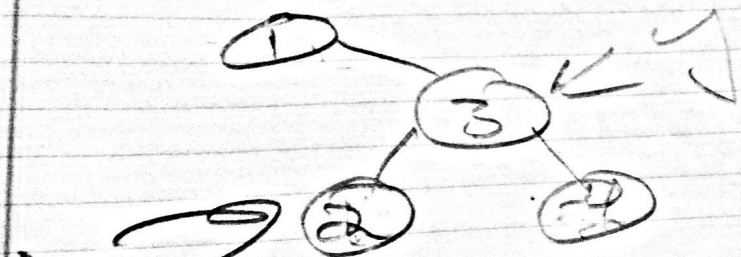
12. Consider a branch of a binary search tree. this branch divides the tree into three sets: set A, the elements to the left of the branch; set B, the elements on the branch; and set C; the elements to the right of the branch. Provide a small counterexample to the statement "for all $a \in A, b \in B$ and $c \in C$, $a \leq b \leq c$."


r
11
12
3
2
7
8
9

13. Consider a bst T whose keys are distinct. We call key y a **successor** of key x iff $y > x$ and there is no key z for which $y > z > x$. Show that if the right subtree of the node storing key x in T is empty and x has a successor y, then the node that stores y must be the lowest ancestor of x whose left child is also an ancestor of x. Show two examples of this, one in which y is the parent of x, and one for which y is not the parent of x. Label both x and y.

examples

y is the parent of x



y is not the parent of x