COP3502 - 10/24/23

(0) Junior Knights - Spr 2024, volunteers

(1) Binary Trees

Paper - definition
traversals (preorder, inorder, postorder)
binary search tree
search
insert PAPER

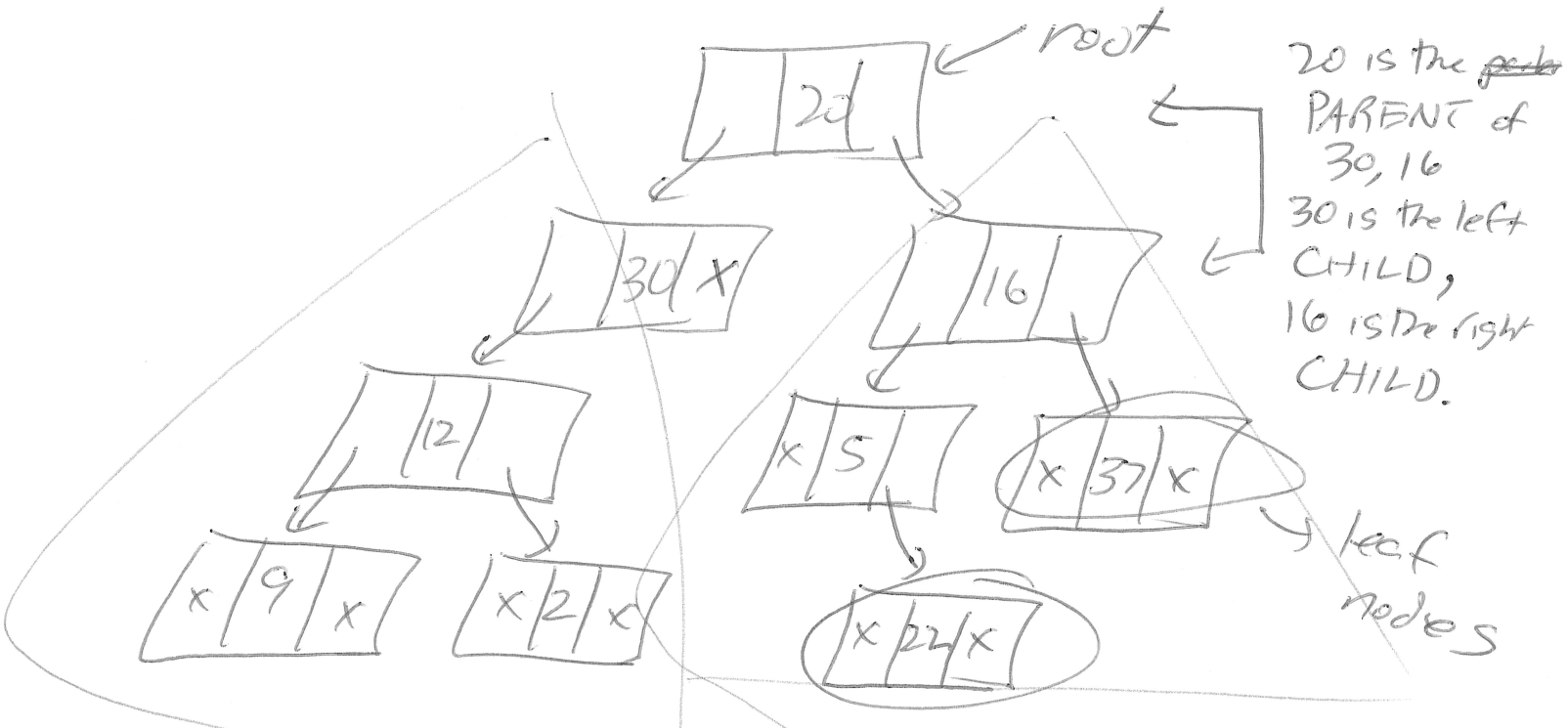
Live vde: Ceiling Function (Kattis)

2 main data structs

dyn. sized array
linked list
} search, insert, delete] O(n) runtime n = # items

Some sort of tradeoff where 1 item O(1) but others O(n).

```
struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
};
```



nodes are all connected

unique simple path btw any 2 nodes.
(no repeated vertices) - no cycle

links = n-1, n = # nodes

height of a tree is the longest distances from root to any leaf node.

n	height
0	-1
1	0
2	1

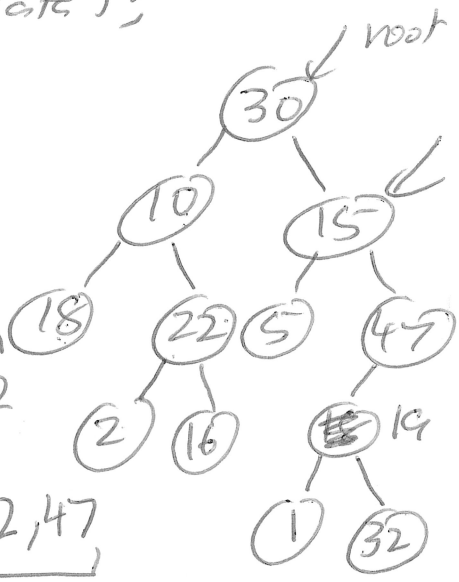
Binary Tree Traversals

1. root
2. left subtree
3. right subtree

Possible orders w/ L < R

preorder	(A)	1, 2, 3
Inorder	(B)	2, 1, 3
postorder	(C)	2, 3, 1

```
void preorder(bintreenode* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
```



pre left 30 | pre right 30

Pre: 30, 10, 18, 22, 2, 16, 15, 5, 47, 19, 1, 32

In: 18, 10, 2, 22, 16, **30**, 5, 15, 1, 19, 32, 47

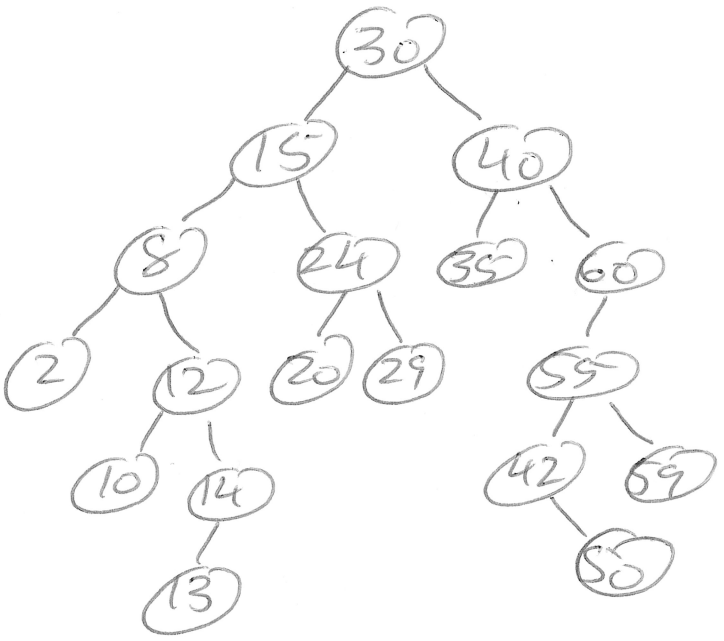
in left 30 | in right 30

Post: 18, 2, 16, 22, 10, 5, 1, 32, 19, 47, 15, **30**

post left 30 | post right 30

$O(n)$ run-time n = # nodes

Binary Search Tree

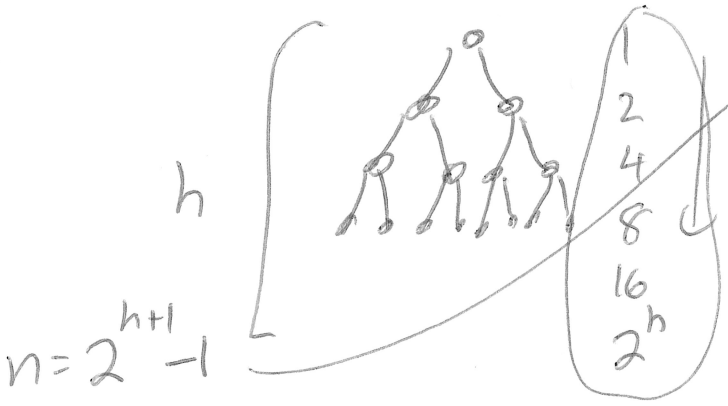


BST node property:
every node in a left subtree ≤ root.
every node in right subtree is ≥ root.

```
int search(bintreenode * root, int value) {
    if (root == NULL) return 0;
    if (value < root->data)
        return search(root->left, value);
    else if (value > root->data)
        return search(root->right, value);
    else
        return 1;
}
```

$O(h)$, h = height

best case



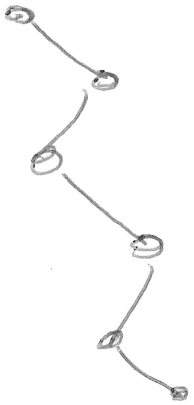
$n+1 = 2^{h+1}$

$\log_2(n+1) = h+1$

$h = \log_2(n+1) - 1$

$h = O(\log n)$

Worst



jassed linked list

$O(n)$

$h = n-1$

avg height $O(\lg n)$

height h

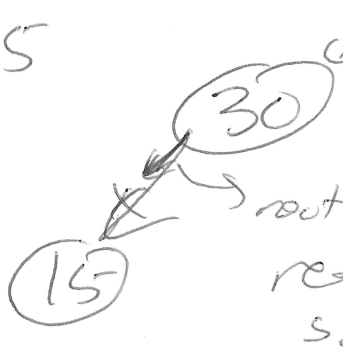
Insert, Search, Delete BST $O(h)$

Insert

B.C. if NULL → create node return it

```
if value < root->data
    root->left = insert(root->left, value);
else
    root->right = insert(root->right, value);
return root;
```

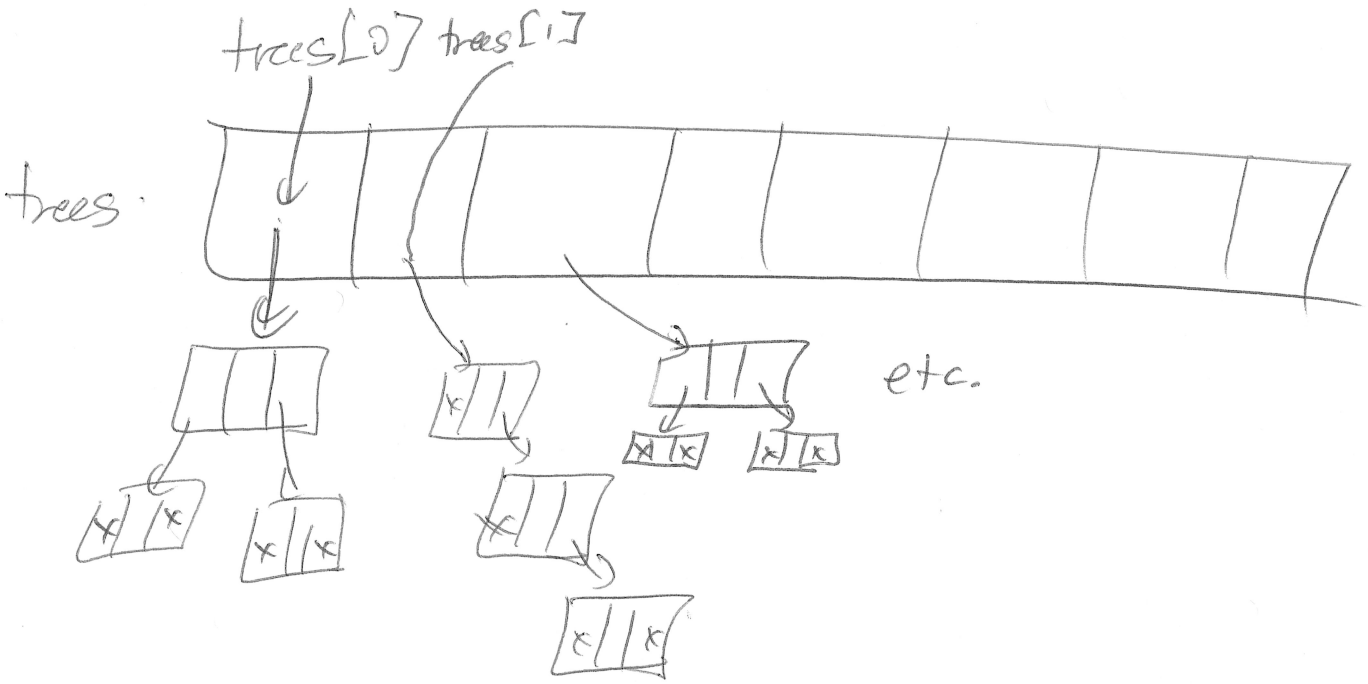
insert 15

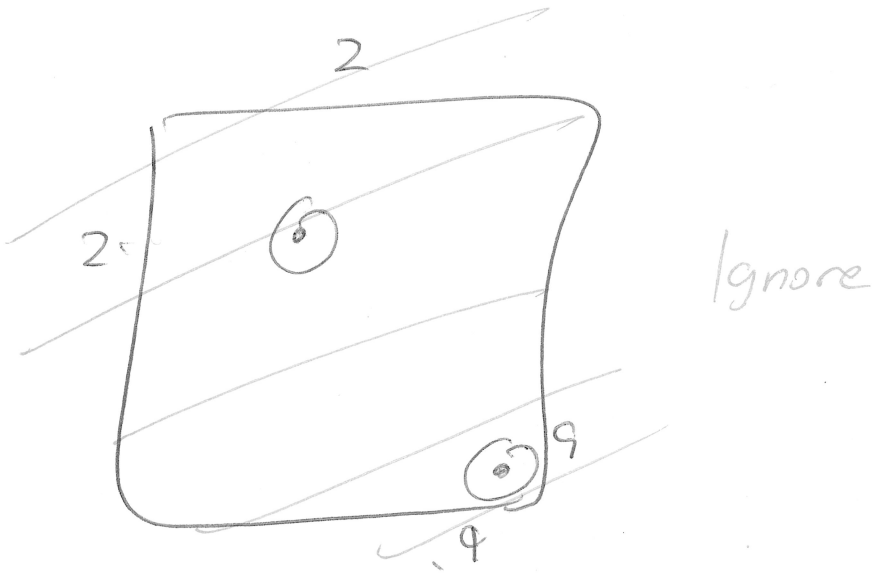


root->left has to be reassigned to new left subtree root

root->left = insert(root->left, value);

Ceiling Function Picture





Ignore