# COP 3502 2/27

(1) BFS - application of queue (Elevator Trouble) Kattis

(2) Binary Trees

q: ~~10~~, ~~15~~, 8, 20, 13, etc.

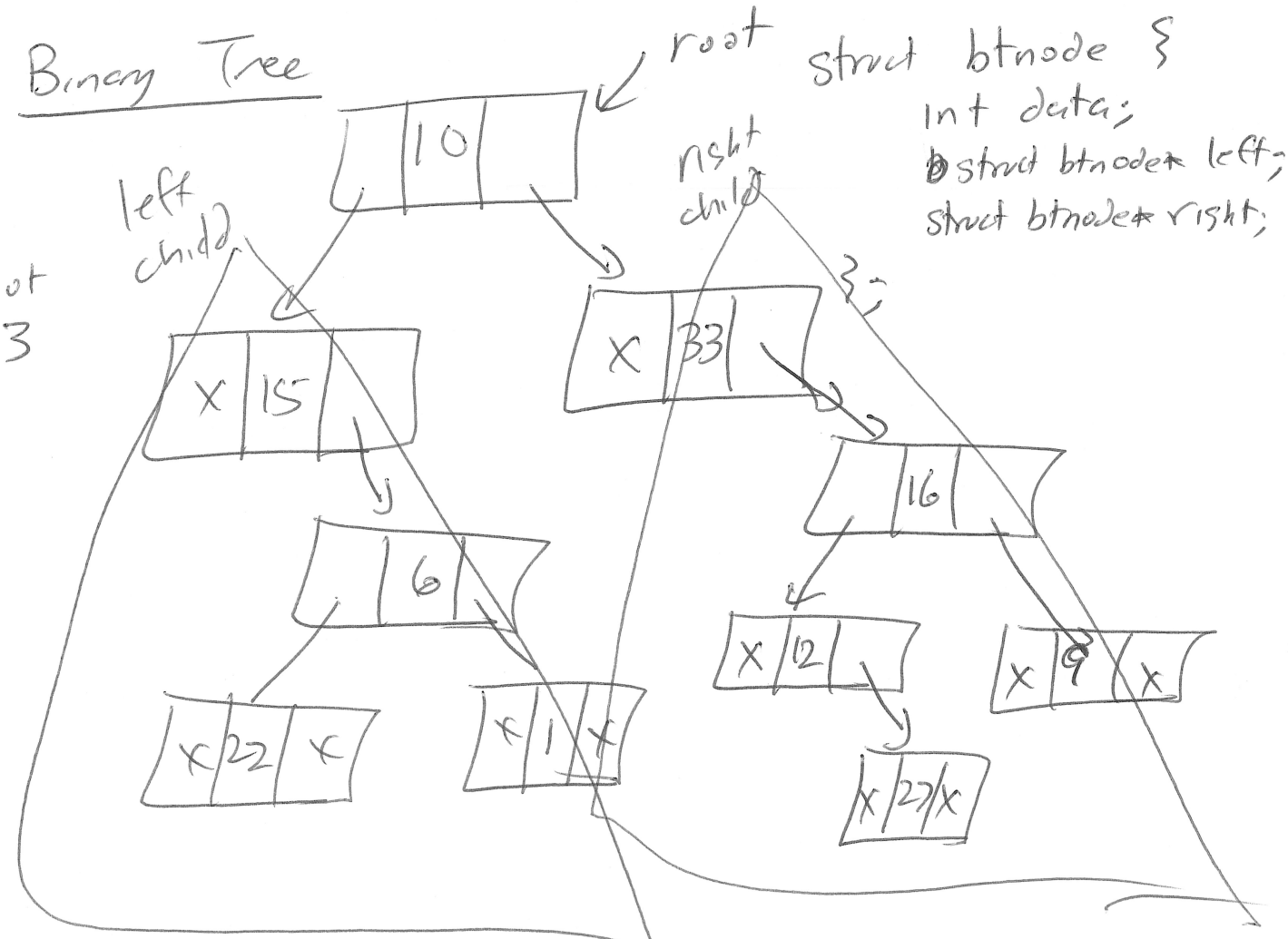
f=30

u=5 d=2

s=10

| ind | 8 | 10 | 13 | 15 | 20 | dist |
|---|---|---|---|---|---|---|
| val | 1 | 0 | 2 | 1 | 2 | |

## Binary Tree




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

tree does NOT have cycles

height = longest distance from root to any leaf node.

# Binary Tree Traversals

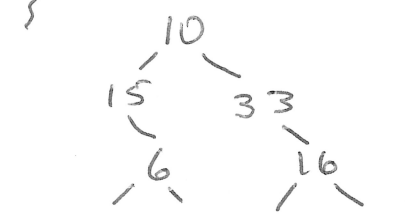
① Preorder [ROOT, LEFT, RIGHT]

② Inorder [LEFT, ROOT, RIGHT]

③ Postorder [LEFT, RIGHT, ROOT]

ROOT
LEFT
RIGHT

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




left: 10, 15, 6, 22, 1

right: 33, 16, 12, 27, 9

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

15, 22, 6, 1 , 10, 33, 12, 27, 16, 9

left | right

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

22, 1, 6, 15, 27, 12, 9, 16, 33, 10

left | right

# Binary Search Tree

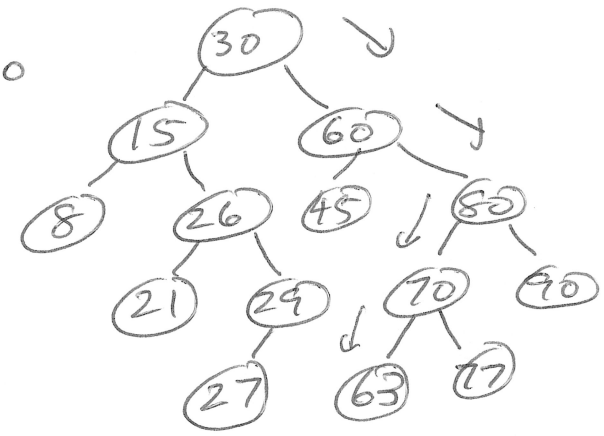
Binary Tree

Search node property: all nodes in the left subtree of a node store value less than it. All values in right subtree store values greater than it.

find 63

Pre: 30, 15, 8, 26, 21, 29, 27
60, 45, 80, 70, 63, 77, 90

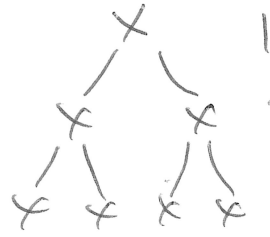
In: 8, 15, ...




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

Traversals: $O(n)$, n nodes in tree

Search: $O(h)$, h = height of tree



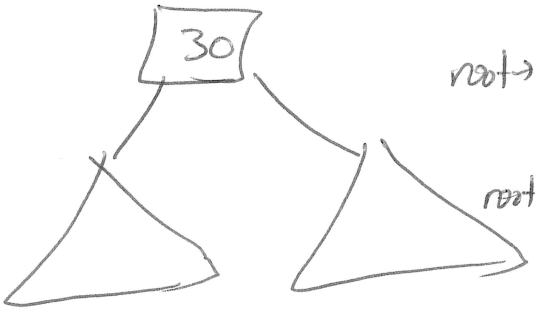

Tree of height h has at most $2^{h+1}-1$ nodes. It has at least $h+1$ nodes. roughly

$$\log_2 n < h < n$$

$$1+2+4+8+\dots+2^k = 2^{k+1}-1$$

# Insert

NULL ~~root~~ special case




```
Insert X
If  X < root->data
root->left = Insert(root->left
   else
root->right=Insert(root->right)
return root;
```