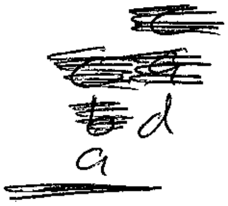
3/7/17 ①

① Exam 2 + Next 2 Weeks

② Stacks

③ binary trees

Stacks - Abstact Data Type
- Last In First Out
- push (puts item on top)
- pop (removes top item
- size
- top (returns top w/o removing)

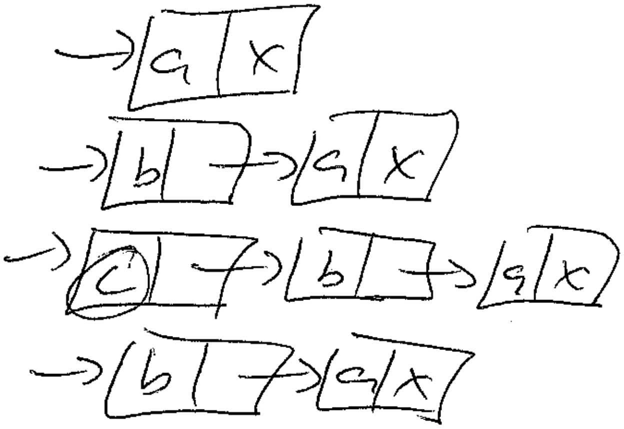


① LL

② Stack

→ push - add to Front, pop - delete first front

push(a)
push(b)
push(c)
x = pop()



```
struct stack {
  struct node* front,
  int size;
};
```

① Call Stack

~~f(3)~~ ~~f(2)~~
~~f(4)~~ ~~f(3)~~
~~f(5)~~

3/7/17 ②

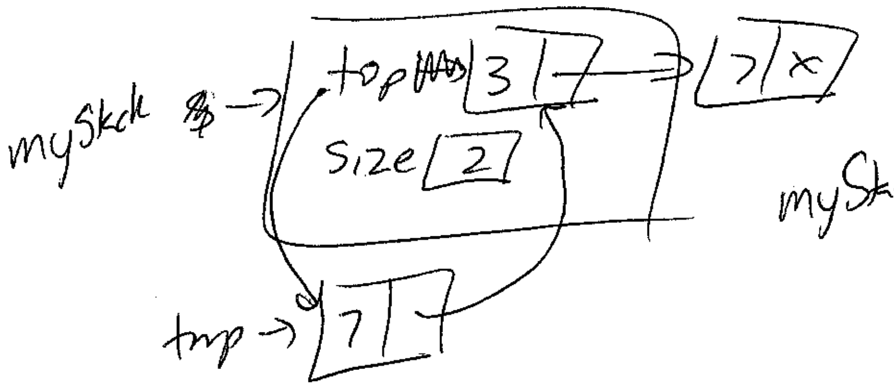
Array Implementation

items	3	7	~~8~~ 9	7	8	9	3	6

→ if stack fills up, realloc OR fail → don't execute push.

top ~~0~~ ~~1~~ ~~2~~ 3

push(3)
push(7)
push(5)
x = pop() → ① $\overset{\rightarrow}{s.}$items[$\overset{\rightarrow}{s.}$top -1]
② s.top --;



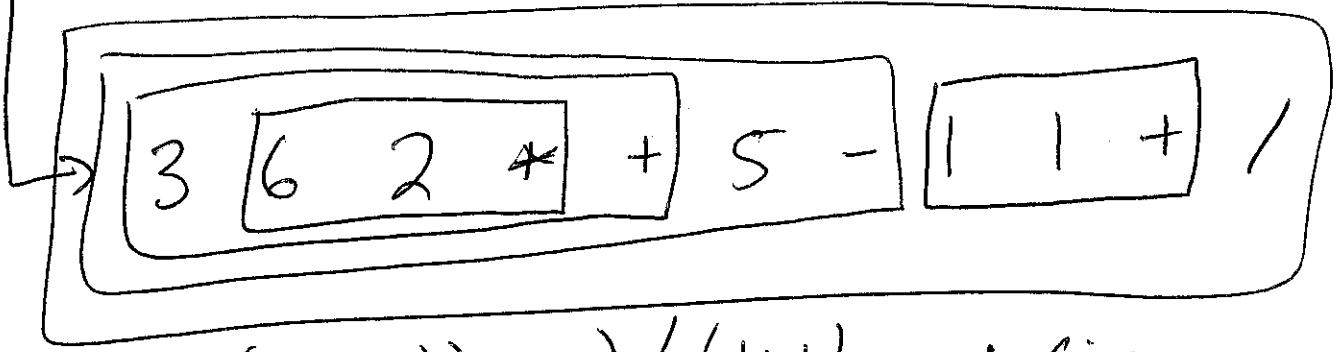
myStack->top = tmp;

3/7/17 ③

Two Stack Algorithms

① Evaluating a postfix expression

② Converting infix to postfix



((3+(6*2))-5)/(1+1)   Infix

(3+4)*5

order of ops

Read expression L→R

Use a <u>operand</u> stack (starts empty)

① When you read an operand push it onto the stack

② When you read an operator (op), pop the last 2 items off the stack op2, followed by op1. Calculate op1 op op2. Push this value back onto the stack

⟹ if ~~your stack~~ you ever try to pop an empty stack, the expression is invalid. if you end up w/a stack size >1, it's also invalid.

3/7/17 ④

Infix → Postfix

Stack → <u>Operator</u> stack

$((3+6*2)-5)/(1+1)$

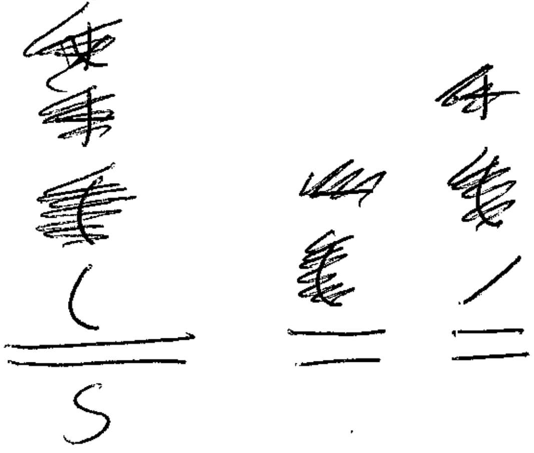
① Open paren → push onto stack

② Operand → Place into expression

③ Close paren → pop items off stack placing each in the expression until we hit the 1st open paren.

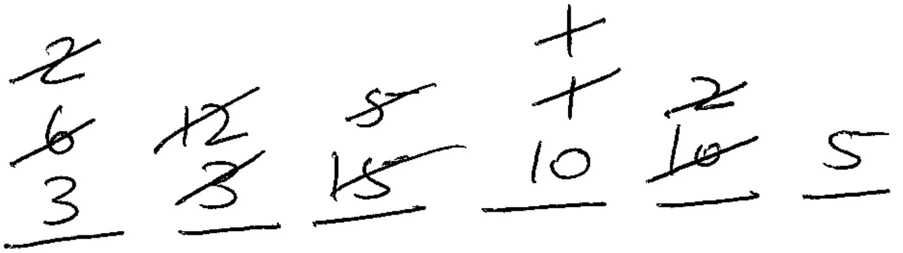
④ Operator → Pop off the stack each operator of equal OR higher precedence, placing each into the expression. Stop popping when you reach a operator of lower precedence OR a parenthesis OR the end of the stack. Push this operator onto the stack.

END Pop off remaining operators + place in expr.



Exp: 3 6 2 * + 5 − 1 1 + /

$3 \quad 6 \quad 2 \quad * \quad + \quad 5 \quad - \quad 1 \quad 1 \quad + \quad /$



$6 * 2 = 12$ $\qquad ((3+(6*2))-5)/(1+1)$

$3 + 12 = 15$ $\qquad ((3+12) -5) / 2$

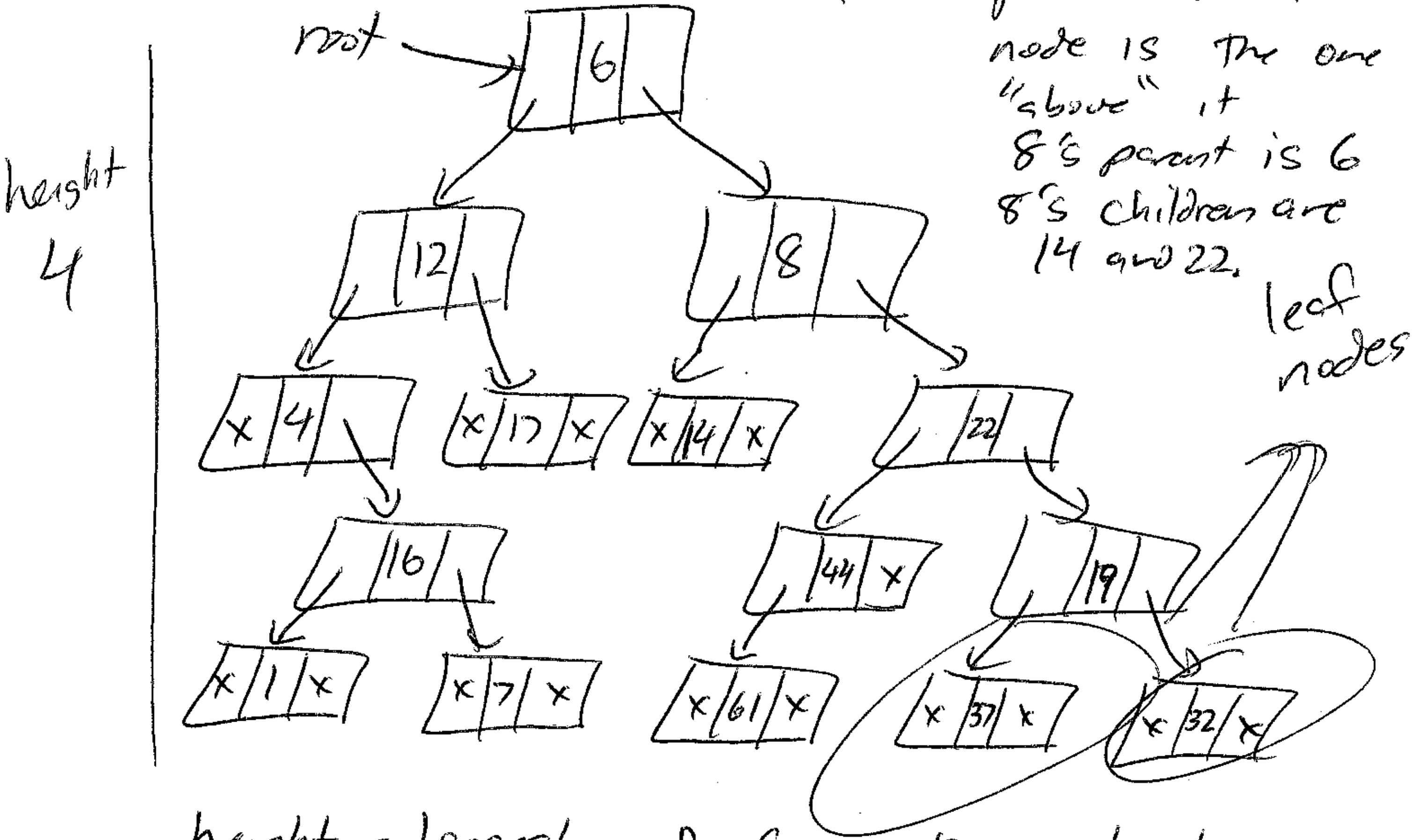
$15 - 5 = 10$ $\qquad (15 - 5) /2$

$1 + 1 = 2$ $\qquad 10 /2$

$10 / 2 = 5$ $\qquad 5$

3/9/17

Binary Trees



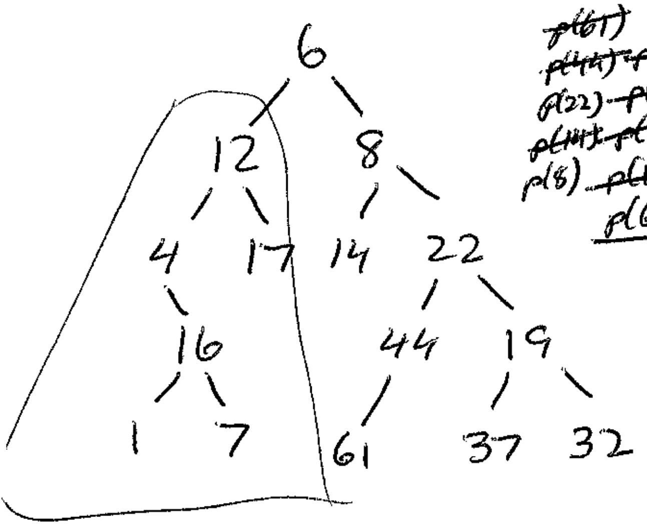
parent of a node is the one "above" it
8's parent is 6
8's children are 14 and 22.

height - longest path from the root to any leaf node

tree of 0 nodes has height -1
tree of 1 node has height 0
tree of 2 nodes has height 1

How do I traverse a tree?

3/9/17 ②



~~p(61)~~
~~p(44)~~ ~~p(1)~~ ~~p(7)~~
p(22) ~~p(16)~~
~~p(14)~~ ~~p(4)~~ ~~p(17)~~
p(8) ~~p(12)~~
p(6)

```
typedef struct treenode {
        int data;
        struct treenode* left;
        struct treenode* right;
} treenode;
```

6, 12, 4, 16, 1, 7, 17,
8, 14, 22, 44, 61, 19,
37, 32

```
void preorder(treenode* root) {
    if (root != NULL) {
①   →   printf("%d ", root->data);
②   →   preorder(root->left);
③   →   preorder(root->right);
    }
}
```

4, 1, 16, 7, 12, 17, 6
14, 8, 61, 44, 22, 37, 19, 32

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

1, 7, 16, 4, 17, 12, 14
61, 44, 37, 32, 19, 22,
8, 6

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

Couple Example Functions on binary trees

① Sum of all leaf nodes in a binary tree

```
int sumleaf (treenode* root) {
        if (root == NULL) return 0;
        if (root->left ==NULL &&
            root->right == NULL) return root->date;
        return sumleaf(root->left) + sumleaf(root->right);
}
```

② height of a tree

```
int height (treenode* root) {
    if (root == NULL) return -1;
    int lheight = height (root -> left);
    int rheight = height (root -> right);
    if (lheight > rheight)
        return 1 + lheight;
    return 1+ rheight;
}
```

Binary Search Tree

→ BST property: for any node, all values in its left subtree are less than it, all values in the right are greater than it.

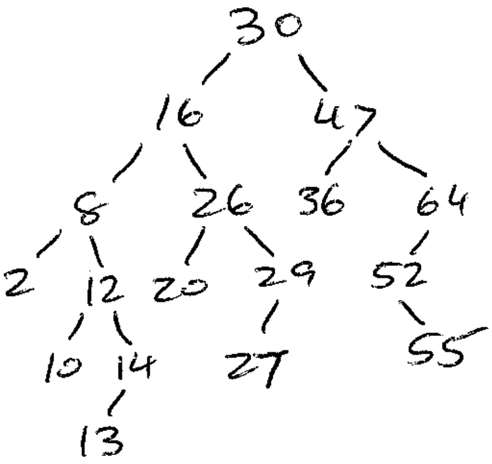
Runtimes of each traversal are $O(n)$ for a tree of n nodes.

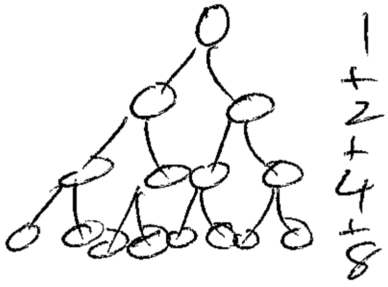
A search in a Binary Search Tree takes $O(h)$ time, where h = height of the tree.

→ h is the worst case is $O(n)$.

→ h in the best case is $O(\lg n)$.

→ h is the avg case in $O(\lg n)$.





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

let $n = 2^{k+1}-1$

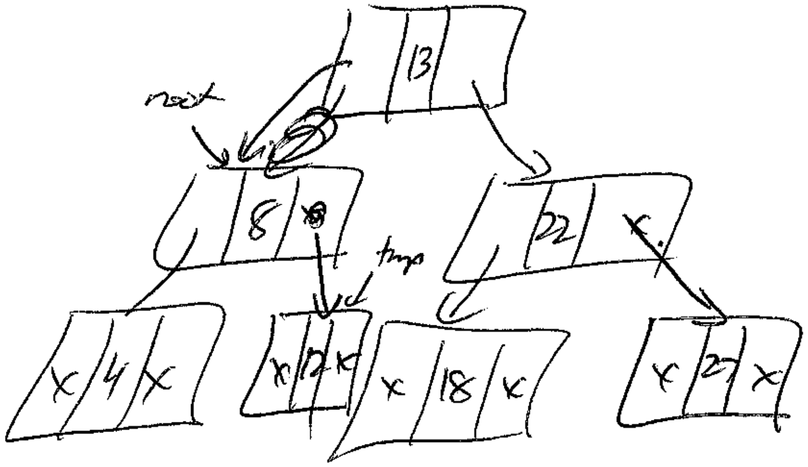
$n+1 = 2^{k+1}$

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

$k = \log_2(n+1)-1 = O(\lg n)$

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

Inserting into a BST



```
treenode*  insertrec (treenode* root, int value) {
    if ( root == NULL) {
        treenode* tmp = malloc(sizeof(treenode));
        tmp->data = value;
        tmp->left = NULL;
        tmp->right= NULL;
        return tmp;
    }
    if  (value <= root->data)
          root->left = insertrec(root->left , value);
    else
          root->right=insertrec(root->right, value);
    return root;
}
```