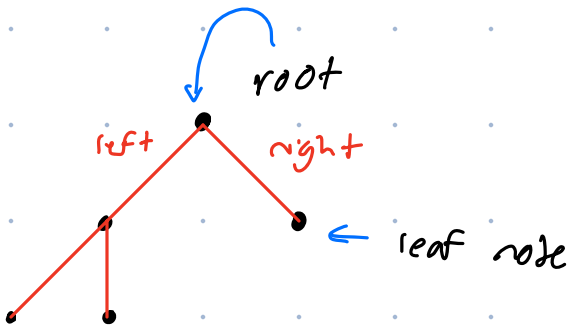
CS 21 LECTURE #9

Assignment 7 hint: for comments – take in whole string, find '#' symbol, clear string, repeat.

Binary Trees!



- A linked structure.
- Follows tree structure.
  - Binary search tree property.
- Does many operations.

Examples

- Build a P.Q., is N in there? → inefficient...
- Build hash table. output in ascending order... → inefficient...

Use a binary search tree!

Operations for BST:

- Insert
- Delete
- Search

} Basic Dictionary functions...

- min/max

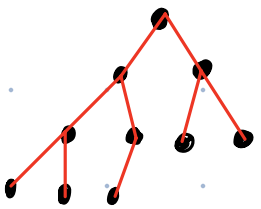
- Predecessor / Successor
- Traversals

Typical performance of BST function is $O(h)$

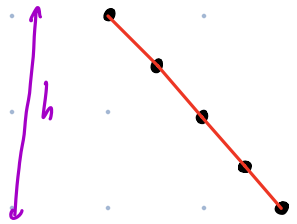
h = height of tree.

- hope for $h \cong O(\log(n))$
- worst case $h \cong O(n)$
  - Happens w/ sorted data
  - COMMON! Sadly...

Best case:



Worst Case



How do we stay away from sorted data?

- Randomize the data!
- But if input one at a time, there's nothing we can do...
- Use another data structure (Balancing BST)

BST Property

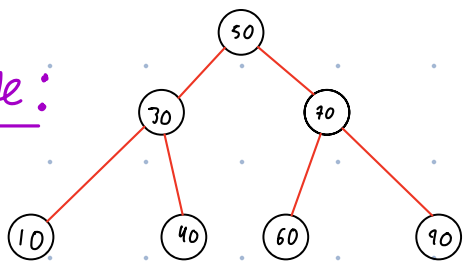
key(x.right) > key(x)

key(x.left) < key(x)

* Assume unique keys

(Note for assignment) ↑

For Example:



Traversal: O(n) → 'touch' n nodes

- A 'walk' of the nodes that touches each node once in some order.

3 traversals:

- pre order: this, left, right
- in order: left, this, right
- post order: left, right, this

Memorize!

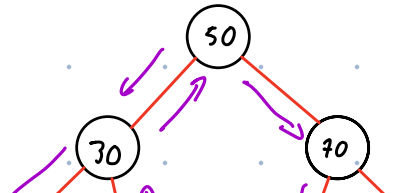
Left always comes before right & "this" is in the expected spot of the 3 based on the name of the traversal

Pseudo code of in order traversal

→ MUST BE A PUBLIC WRAPPER FUNCTION BC USER DOESNT HAVE ACCESS TO SENSITIVE TREE DATA! (to root I believe)

node reference / pointer

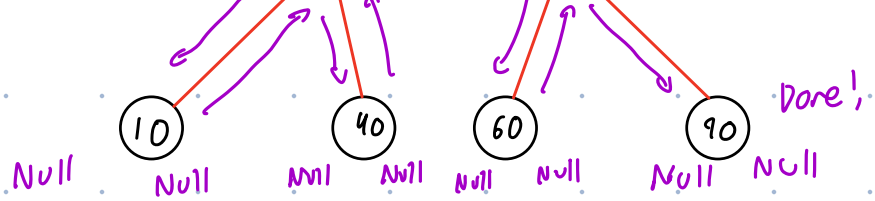
```
inorder (x) {
    if (x == null) { return }
    inorder (x.left);
    print x;                    // or whatever
    inorder (x.right);
}
```

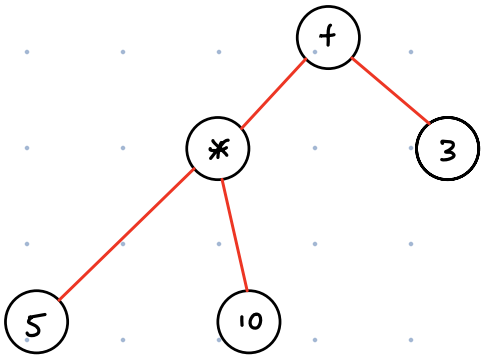


Pre order Printed: 50, 30, 10, 40, 70, 60, 80

In Order printed: 10, 30, 40, 50, 60, 70, 80

Post order Printed: 10, 40, 30, 60, 80, 70, 50





prefix Pre order Printed: + * 5 10 3

infix In Order printed: 5 * 10 + 3

postfix Post order Printed: 5 10 * 3 +

DONT MAKE THIS MISTAKE: NOTICE IN ORDER HAD IN ORDER CALL. IN POST ORDER & PRE ORDER FUNCTIONS, CALL THEMSELVES RECURSIVELY DONT CALL IN ODER IN THE OTHER FUNCTIONS LOL...

Search → MUST BE A PUBLIC WRAPPER FUNCTION BC USER DOESNT HAVE ACCESS TO SENSITIVE TREE DATA!

recursive version

node reference

key

```
Search (x, k) {

    if (x == null || x.key == k) return x;
```

```
        if (k < x.key) {
            return search(x.left, k);
        }
        return search(x.right, k);
    }
```

Iterative version

```
search(x, k) {
    while (x != null && x.key != k) {
        if (k < x.key) {
            x = x.left;
        } else {
            x = x.right;
        }
    }
    return x;
}
```

Min

MUST BE A PUBLIC WRAPPER FUNCTION BC USER DOESNT HAVE ACCESS TO SENSITIVE TREE DATA!

O(h)

CHECK IF X IS NULL

Note ref

CHECK IF X IS NULL IN WRAPPER FUNCTION!

```
Min (x) {
    while (x.left != null) {
        x = x.left;
    }
    return x;
}
```

Max

MUST BE A PUBLIC WRAPPER FUNCTION BC USER DOESNT HAVE ACCESS TO SENSITIVE TREE DATA!

O(h)

CHECK IF X IS NULL IN WRAPPER FUNCTION!

```
Max (x)
    while (x.right != null) {
        x = x.right;
    }
    return x;
}
```

Successor

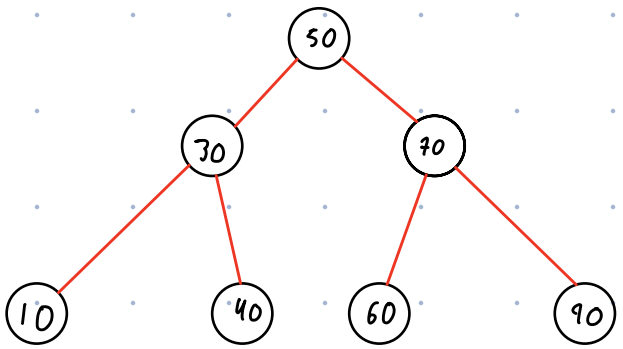
MUST BE A PUBLIC WRAPPER FUNCTION BC USER DOESNT HAVE ACCESS TO SENSITIVE TREE DATA!

Note

CHECK IF X IS NULL IN WRAPPER FUNCTION!

```
Successor (x) {
```

```
if (x.right != null) {
    return min(x.right);
}
t = x.parent;
while (t != null && x == t.parent) {
    x = t;
    t = t.parent;
}
return t;
}
```



S(30) -> 40

S(10) -> 30

S(50) -> 60

S(90) -> null...

Insert
O(h)

key value

```
insert (k) {
    n = new node(k)    // node constructor left right & parent
                       // objects to null
```

```
p = null;          // previous
t = root;          // temp

while (t != null) {
    p = t;
    if (k < t.key) {
        t = t.left;
    } else {
        t = t.right;
    }
} // t always null here

n.parent = p

if (p == null) {
    root = n;
    return;
}

if (k < p.key) {
    p.left = n;
} else {
    p.right = n;
}
}
```