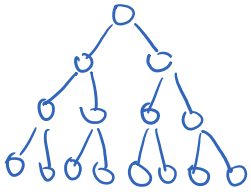
BST

Thursday, November 5, 2020 5:00 PM

Reminder: HW 7 and labs are due this Sunday

Q14 HW7 $h \rightarrow \left\lceil \frac{n}{2^{h+1}} \right\rceil$ induction / Q13



basis step: $h=0$ → leaves ⇒ based on Q13 ⇒ $\left\lceil \frac{n}{2} \right\rceil$

Q14: $\left\lceil \frac{n}{2^{0+1}} \right\rceil = \left\lceil \frac{n}{2} \right\rceil$

Note: #nodes $= k \Leftrightarrow$ #leaves $= \left\lceil \frac{k}{2} \right\rceil$ ← Q13

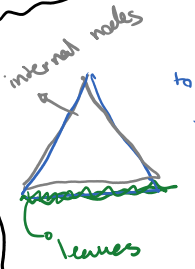
Note 2: #leaves $= \left\lceil \frac{k}{2} \right\rceil \Leftrightarrow$ #internal $= \left\lfloor \frac{k}{2} \right\rfloor$

inductive step:

(I) assumption: $h=k$ ⇒ total nr of nodes having height $k = \left\lceil \frac{n}{2^{k+1}} \right\rceil$

(II) proof: $h=k+1$ ⇒ prove total nr of nodes at height $k+1$ is $\left\lceil \frac{n}{2^{k+2}} \right\rceil$

Note 3:



total nodes $= n$

leaves $= \left\lceil \frac{n}{2} \right\rceil$

internal nodes $= n - \left\lceil \frac{n}{2} \right\rceil = \left\lfloor \frac{n}{2} \right\rfloor$

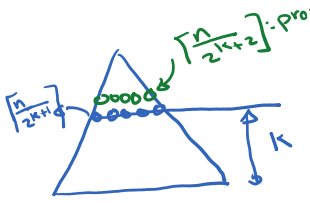
Remove all the leaves

total nodes $= \left\lfloor \frac{n}{2} \right\rfloor$

#leaves $= \left\lceil \frac{n}{2^2} \right\rceil$

internal nodes $= \left\lfloor \frac{n}{2^2} \right\rfloor$

(I) keep removing the leaves k times ⇒



⇒

#nodes $= \left\lfloor \frac{n}{2^k} \right\rfloor$

#leaves $= \left\lceil \frac{n}{2^{k+1}} \right\rceil$ = from my assumption

#internal $= \left\lfloor \frac{n}{2^{k+1}} \right\rfloor$ = note 2

⇒ (II) Remove the leaves one more time

#nodes $= \left\lfloor \frac{n}{2^{k+1}} \right\rfloor$

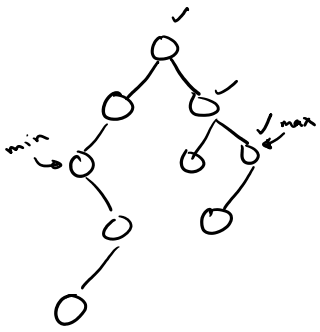
#leaves $= \left\lceil \frac{n}{2^{k+2}} \right\rceil$ ← total nr of nodes at height $k+1$ in the original tree :)

Find_min & Find_max: $O(h)$ $\log n \leq h < n$

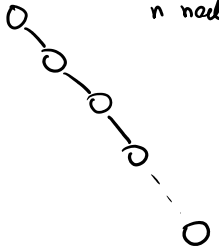
Johnathon

```
Find_max(node node)
    current = node
    while(current.right != null)
        current = current.right;
    return current.val;

Find_min(node node)
    if(node.left == null)
        return(node.val);
    else
        return find_min(node.left)
```



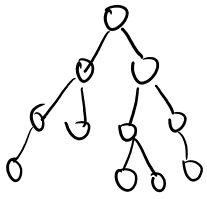
worst-case : $O(n)$



n nodes $\Rightarrow h = n-1$

best-case : $\Omega(1)$

avg-case : $O(h) = O(\log n)$

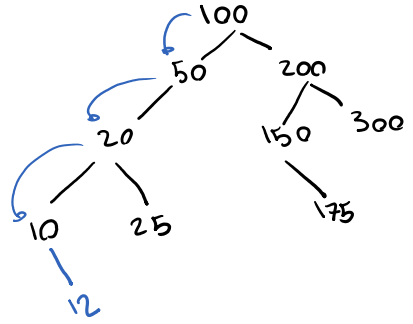


n nodes $\Rightarrow h = \log n$

best-case
avg-case

Insertion : $O(h)$

$\rightarrow$ Start from root



Search : $O(h)$

khang

Search : O(h)

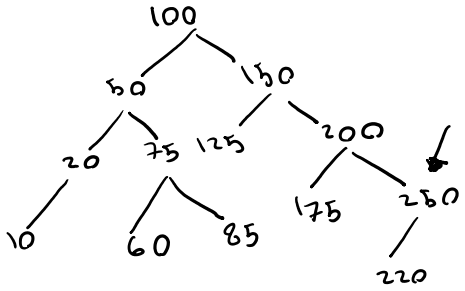
Khang

root

```
search(root, key)
if (root == null || root.key == key)
    return root;
if (root.key < key)
    return search(root.right, key)
return search(root.left, key)
```

Find Successor: O(h)

```
Find_Successor (node)
if node.right
    return a
end
parent = node.p
while a
    a.
```



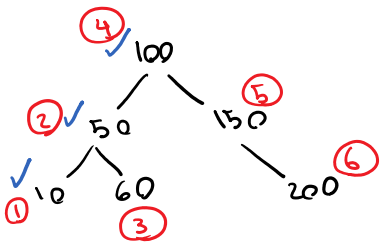
Josh :)

```
successor(node)
    if (node.right != null) {
        return findMin(node.right)
    }
    parent = node.p
    while (parent != null && node == parent.right) {
        node = parent
        parent = parent.p
    }
    return parent
```

O(h)

O(h)

Sorting : O(n)




```
Tree_walk (node)
    if node != null
        Tree_walk (node.left)
```

```
Tree-walk (node)
if node != null
    Tree-walk (node.left)
    print (node.value)
    Tree-walk (node.right)
end
```

(3)

10 50 60 100 150 200

Build BST

avg-case / best-case : k element → h = log k

$$\sum_{i=1}^{n} \log i = O(n \lg n)$$

worst-case :

$$\sum_{i=0}^{n-1} i = \frac{(n-1)(n)}{2} = O(n^2)$$

1, 2, 3, 4, 5