CS314H CSB Lecture 23: Top-Down Splay Tree

1. Recap: `Delete(e)` algorithm (bottom-up way, not top-down)
   - Step 1: Perform standard BST search for *e*. O(h)
   - Step 2: Splay *e* to the root O(h)
   - Step 3: Remove *e* (Let L, R be the left and right subtree respectively) O(1)
   - Step 4: Search for *Lmax* (Max of L), O(h)
   - Step 5: Splay *Lmax* to the root O(h)
     Since it is max of L, it should have no right subtree) and attach R as the right child of *Lmax. O(1)*

What is the time complexity for each step above? 4 x O(h) + O(1)
Amortized cost is 4x O(log n), which is O(log n)

2. **(True/False)** Another possible way to perform `Delete(e)` would be to take the predecessor of `e` and place it directly at the root, without splaying it up through each level.
   Why?

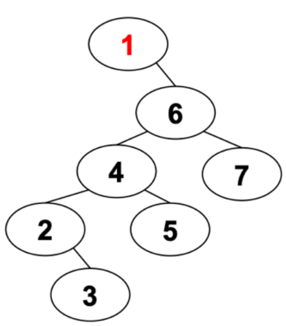
This is because the *proof* of splay trees' amortized complexity relies on the fact that **every access ends with a splay rotation sequence** that redistributes weight and reduces potential. If you skip splaying, you break the proof.

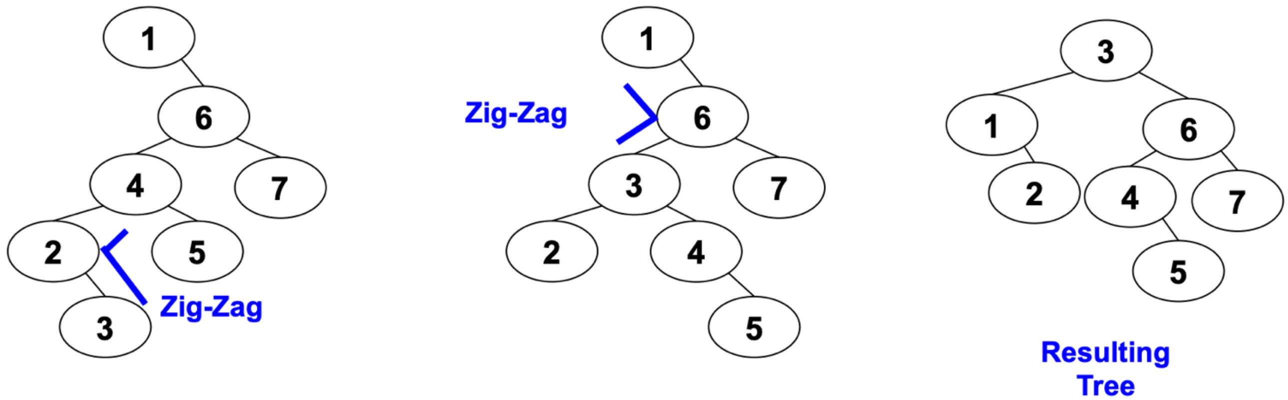
3. `Find(e)` operation in a splay tree can be performed in two ways:
   (1) use the **bottom-up** splay method, which performs a standard BST search for *e* and then splay *e* to the root.
   (2) use the **top-down** splay method, which performs splaying while descending the search path.

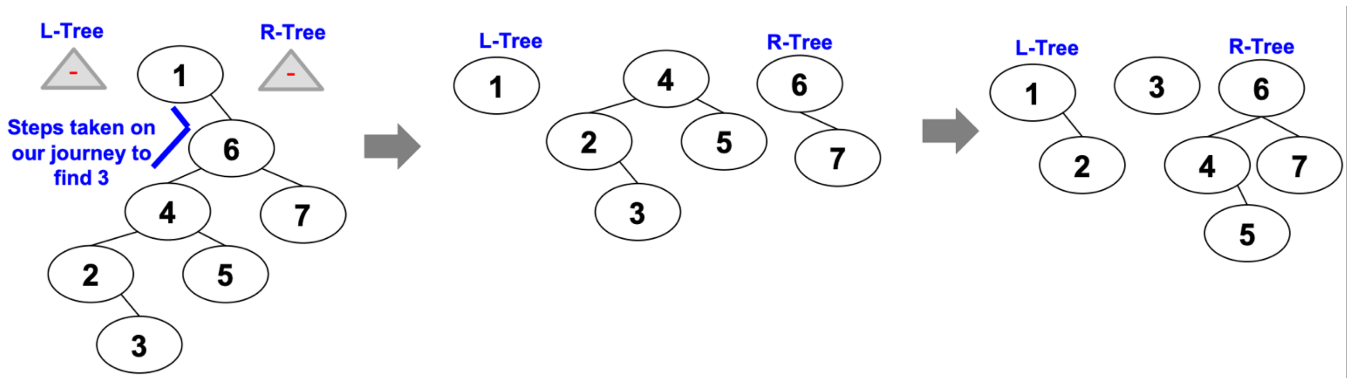
(True/False) The resulting trees from these two methods will be identical.

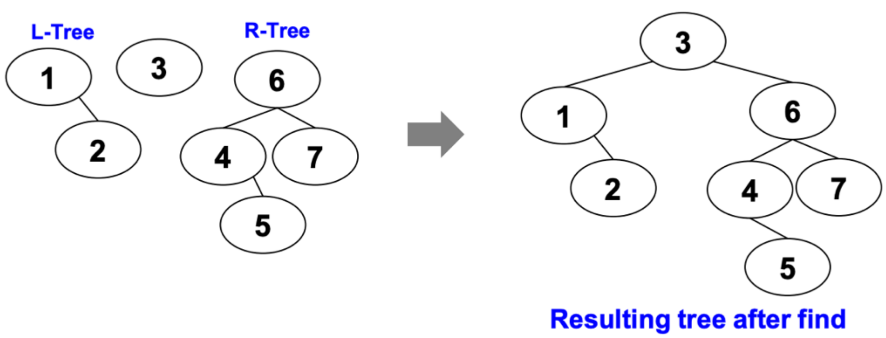
False!

Ex 1) `Find(3)`









Ex 2) `Find(0)`

```
      5
     / \
    3   7
   / \
  1   4
 /   \
```

```
0   2
```

Bottom up

```
Step 1: zig-zig                          Step 2: zig
(Lift 1 first) → Lift (0 next)
     5               5                        0
    / \             /   \                      \
   1   7           0     7                      5
  / \               \                          / \
 0   3               1                        1   7
    / \               \                        \
   2   4               3                        3
                      / \                      / \
                     2   4                    2   4
```

Top down

Initially,

```
L = null,     M =        5          R =null
                        / \
                       3   7
                      / \
                     1   4
                    / \
                   0   2
```

Step 1: zig-zig

```
L = null,     M =        1          R =  3
                        / \               \
                       0   2               5
                                          / \
                                         4   7     (Rmin = 3)
```

Step 2: zig

```
L = null,     M =        0          R =   3
                                        /   \
                                       1     5
                                        \   / \
                                         2 4   7     (new Rmin = 1)
```

```
Attached 1      as the left child of Rmin (3)
          \
```

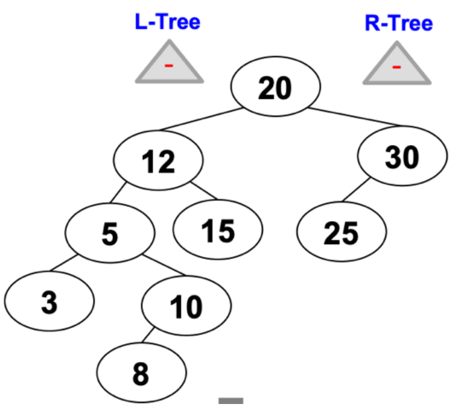
```
                2
```

M is equal to target, thus stop.
Step 3: Final tree - Attach t with L and R.

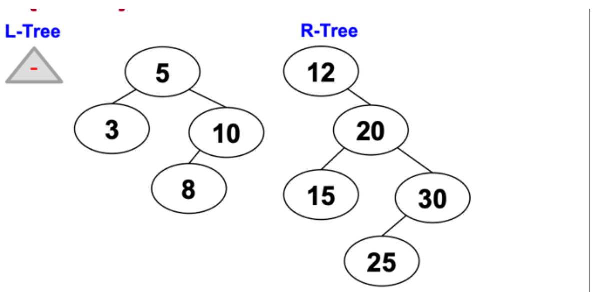
```
     0
      \
       3
     /   \
    1     5
     \   / \
      2 4   7
```

As you can see the two resulting trees from top-down and bottom-up splay is different

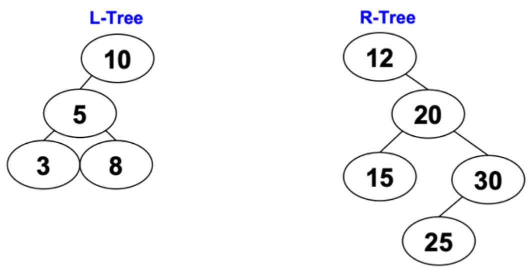
4. Perform `Insert(11)` in top-down manner. When to stop and construct a new tree with root 11?



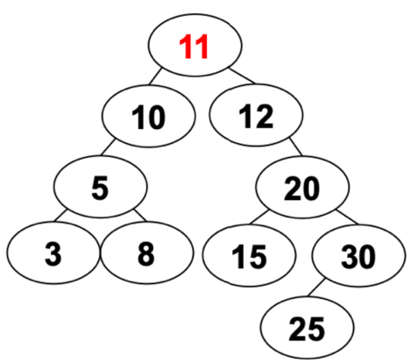
Step 1: top-down zig-zig



Step 2: top-down zig-zig (symmetric)



Step 3: M is empty. (This is the stopping condition)
Create a new node 11 and attach L and R.



5. Perform delete 5 in top-down manner from above tree.
Initially L = null, R = null, M = T

Step 1. Top down zig-zig.
L = null,   M =   5   R= 10

```
       /  \                 \
      3    8                 11
                               \
                                12
                                  \
                                   20
                                  /  \
                                15    30
                                     /
                                   25           Rmin = 10
```

Attach L and R (case 2 when t has subtrees)

```
     5
    /  \
   3    10
       /  \
      8    11
             \
              12
                \
                 20
                /  \
              15    30
                   /
                 25
```

Step 2: Remove 5 (L = 3 and R = the subtree with root of 10)
Step 3: Let's do successor replacement. Perform Top-down zig.
Initally L' = null, M = R(Red tree above) , R' = null,
Finally tree:

```
     8
    /  \
   3    R
```

6. Compare time complexity:
`Delete(e)` vs `Delete-top-down(e)`
4 x O(h) + O(1) vs 2 x O(h) + O(1)