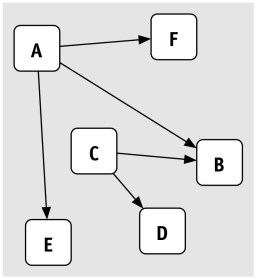
Lecture 7a: Trees!

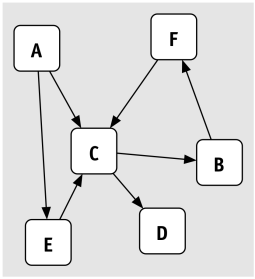
CS 70: Data Structures and Program Development

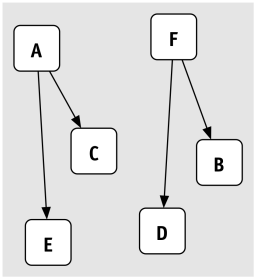
Tuesday, March 3, 2020

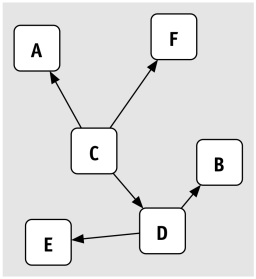
What is a tree?

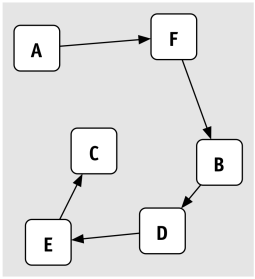
Which of these are trees?

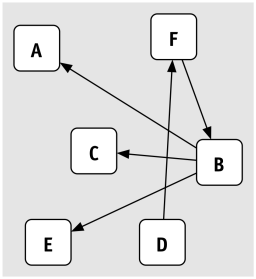






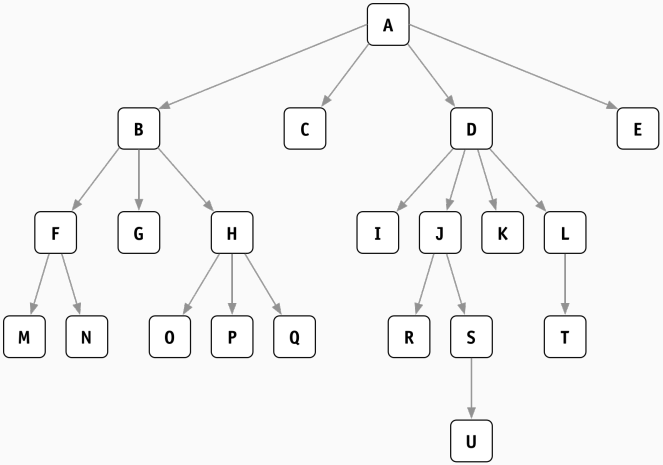




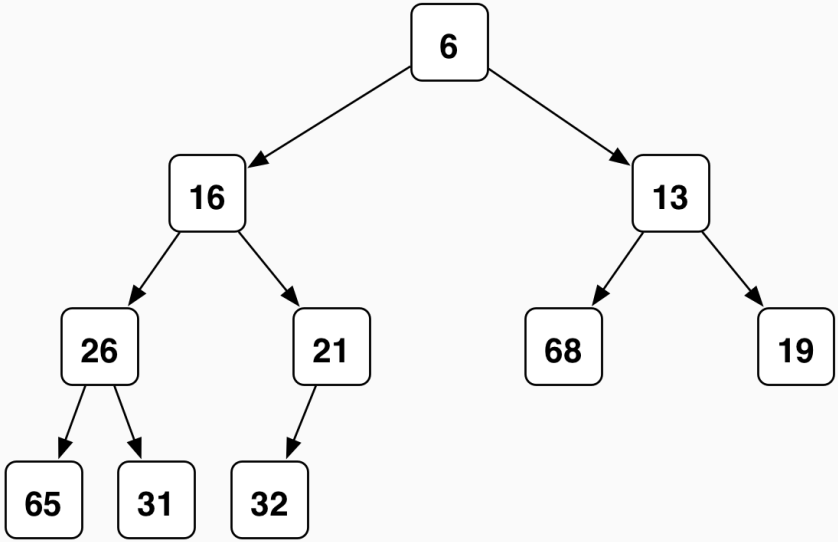


Terminology

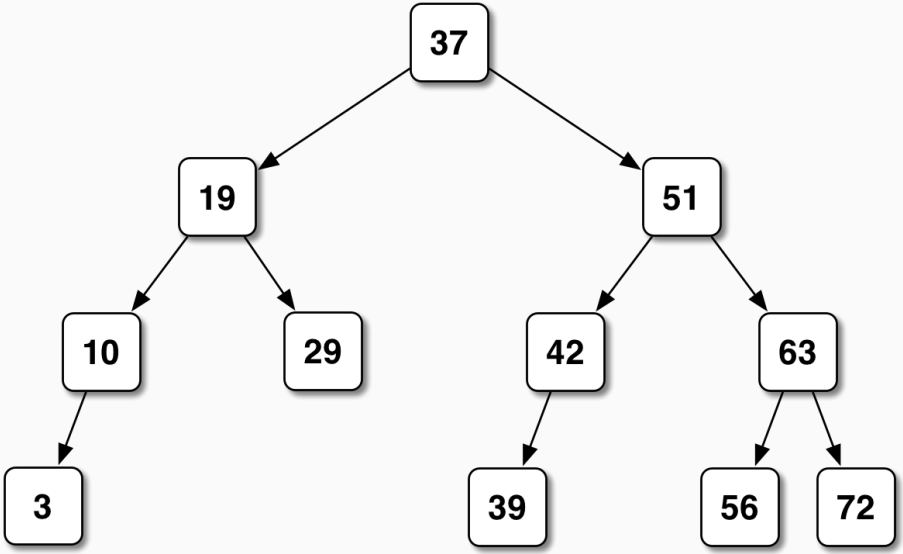
node, edge; root, leaf, tree, subtree; parent, child, ancestor; height, balance; binary



Binary Tree

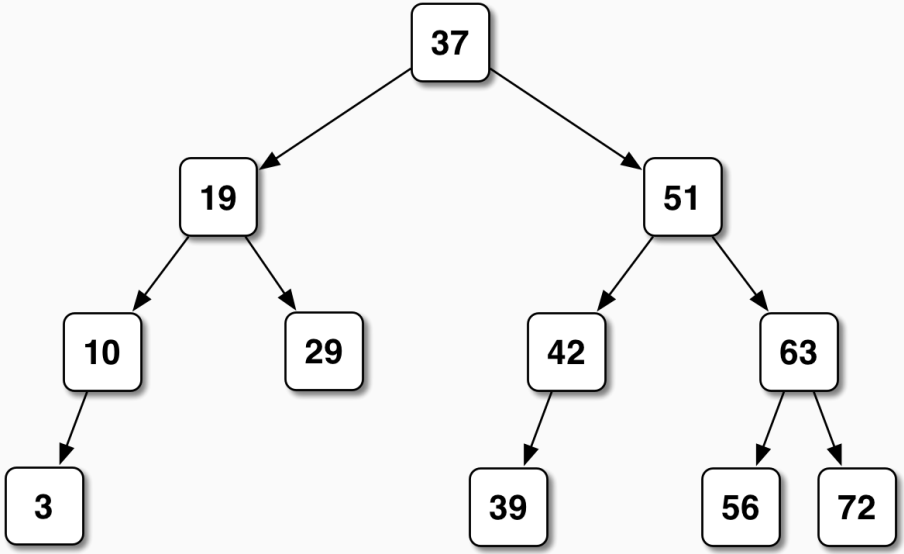


Binary Search Tree (a.k.a. Ordered Binary Tree)



Basic Algorithms

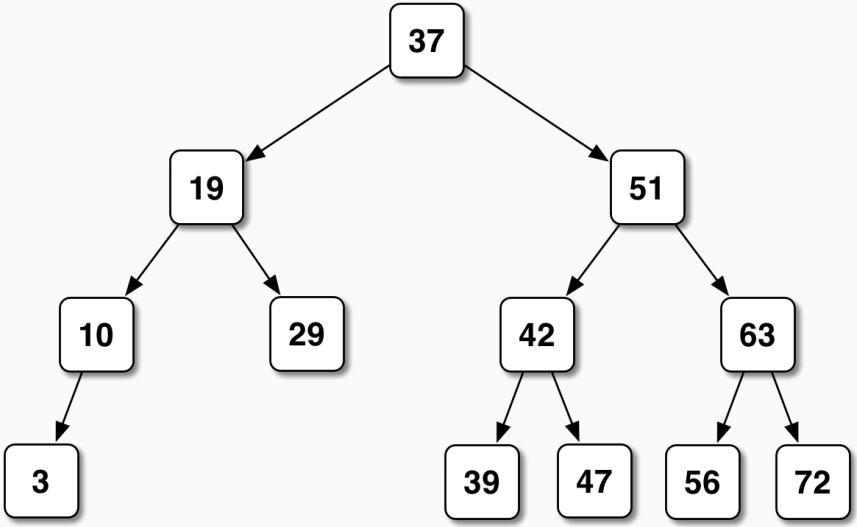
To do: find 56; find 35; insert 47



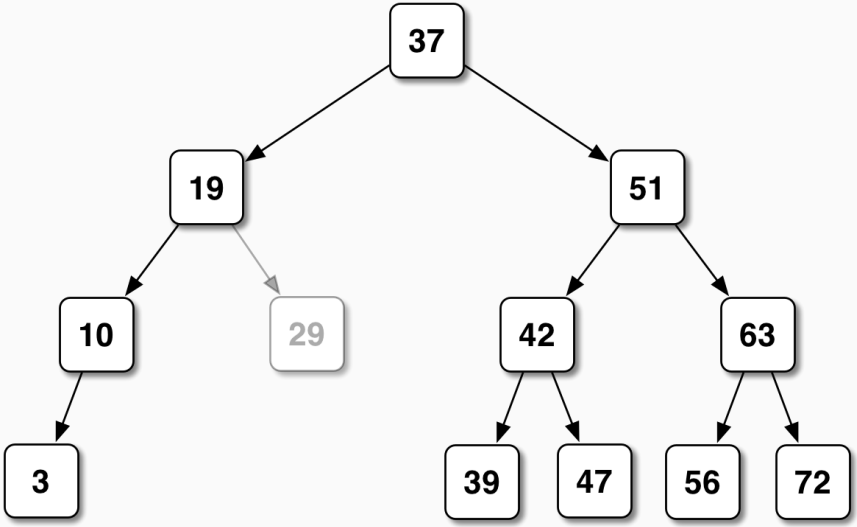
`insert` pseudocode

```
insert(tree, x):
   if tree is empty:
      make x its new root.

   else if x < tree's root:
      insert(left subtree, x)

   else if tree's root < x:
      insert(right subtree, x)
```

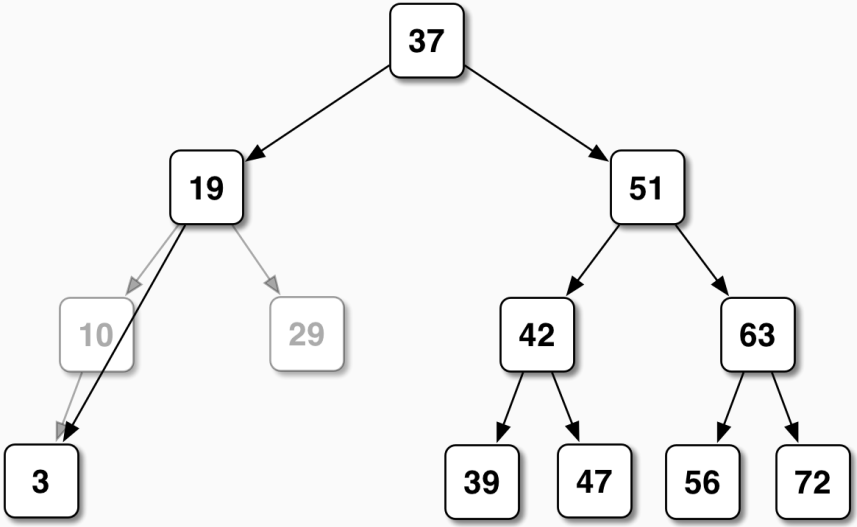
To do: delete 29



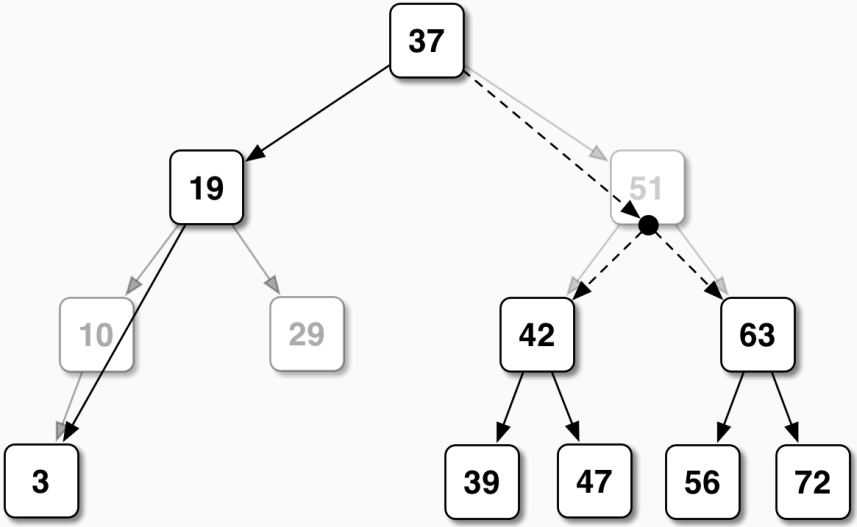
To do: delete 10



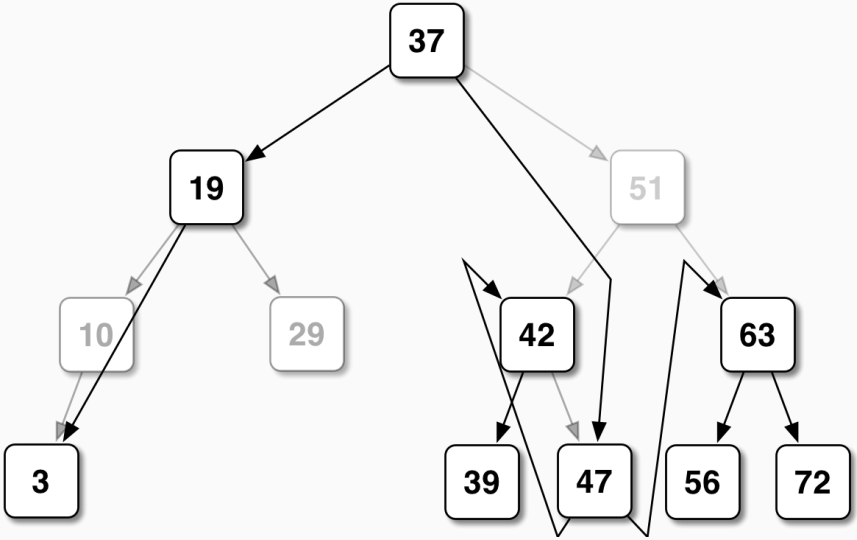
To do: delete 51



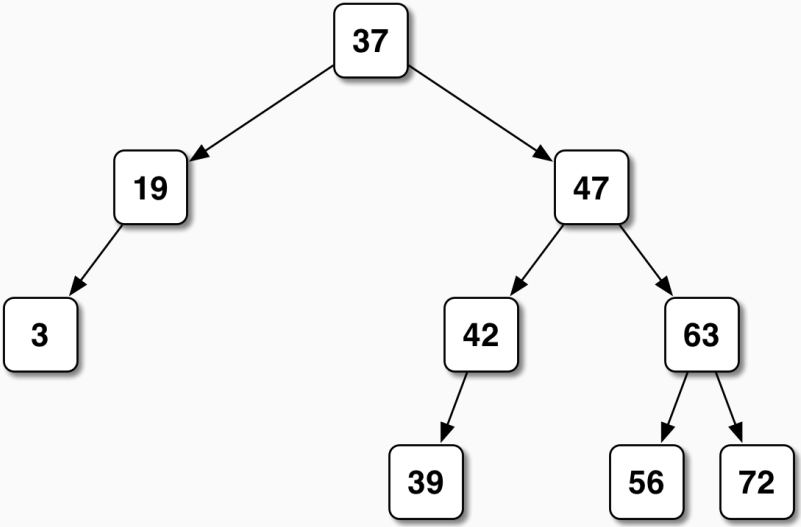
Oops



Better (why?)



Redrawing the same tree...



Exercise

Quick Test: Doing Insert

Choose one of these sequences, and insert it into an empty tree.

D, B, A, F, E, C, G	D, B, F, E, A, C, G	D, B, F, C, E, G, A	D, F, B, C, A, E, G
D, F, E, G, B, A, C	D, F, G, E, B, C, A	D, B, F, C, E, A, G	D, B, C, F, E, A, G
D, B, C, A, F, G, E	D, B, F, A, E, C, G	D, F, G, B, C, E, A	D, B, F, C, G, E, A
D, F, B, G, A, E, C	D, B, C, F, G, A, E	D, B, F, G, E, C, A	D, F, G, B, A, E, C
D, B, F, C, A, G, E	D, F, B, A, E, C, G	D, B, C, F, G, E, A	D, B, F, A, G, C, E
D, F, B, G, C, A, E	D, B, F, E, A, G, C	D, F, B, E, A, G, C	D, B, F, G, A, C, E
D, F, G, B, A, C, E	D, F, B, C, G, E, A	D, F, B, C, G, A, E	D, F, G, E, B, A, C
D, F, B, A, G, E, C	D, B, F, A, G, E, C	D, F, B, E, G, C, A	D, F, G, B, C, A, E
D, F, E, G, B, C, A	D, B, A, F, G, E, C	D, B, F, A, C, G, E	D, B, A, F, G, C, E
D, F, B, C, E, G, A	D, B, A, F, C, G, E	D, F, B, G, C, E, A	D, B, F, A, C, E, G
D, F, B, E, A, C, G	D, F, E, B, G, A, C	D, F, B, E, C, G, A	D, F, E, B, A, C, G
D, B, F, A, E, G, C	D, F, E, B, C, A, G	D, F, E, B, A, G, C	D, F, E, B, G, C, A
D, B, F, G, E, A, C	D, F, B, E, C, A, G	D, F, B, A, C, G, E	D, B, F, E, G, A, C
D, B, C, F, A, E, G	D, B, A, F, C, E, G	D, F, B, C, A, G, E	D, F, B, A, E, G, C
D, F, B, G, E, A, C	D, B, F, E, C, G, A	D, F, E, B, C, G, A	D, B, F, G, C, E, A
D, F, B, A, C, E, G	D, B, F, C, G, A, E	D, B, F, G, C, A, E	D, F, B, A, G, C, E
D, B, A, C, F, G, E	D, B, C, F, E, G, A	D, F, B, G, E, C, A	D, F, G, B, E, C, A

48

Exercise (continued)

What tree results from the following sequences of inserts?

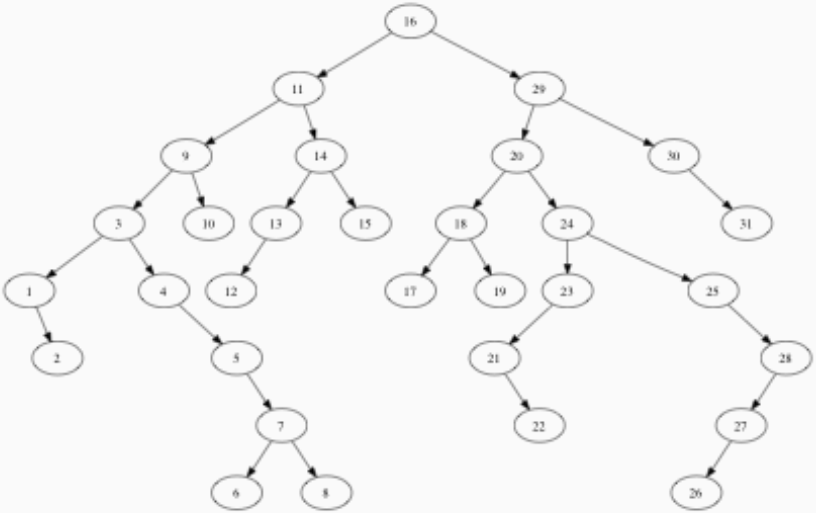
- A, B, C, D, E, F, G
- D, C, A, B, E, F, G

Suppose we have a BST with n nodes.

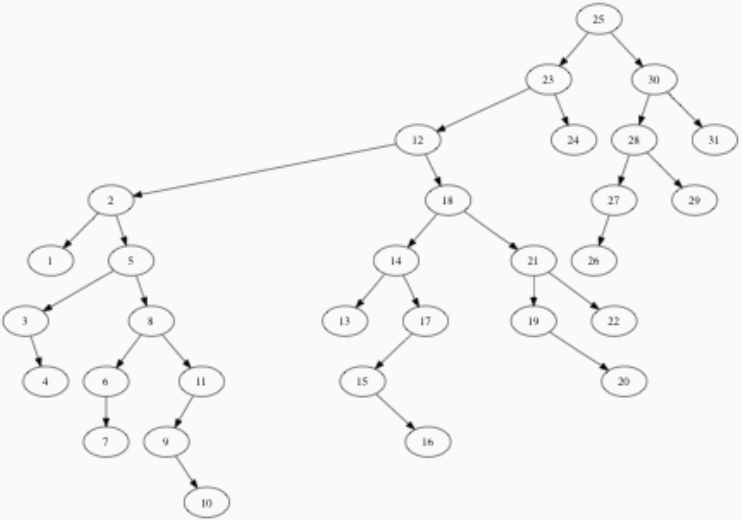
What is the worst-case running time for `find` (and `insert`)

- if we have a really terrible tree?
- if we have a really nice tree?
- if we have a "random" tree?

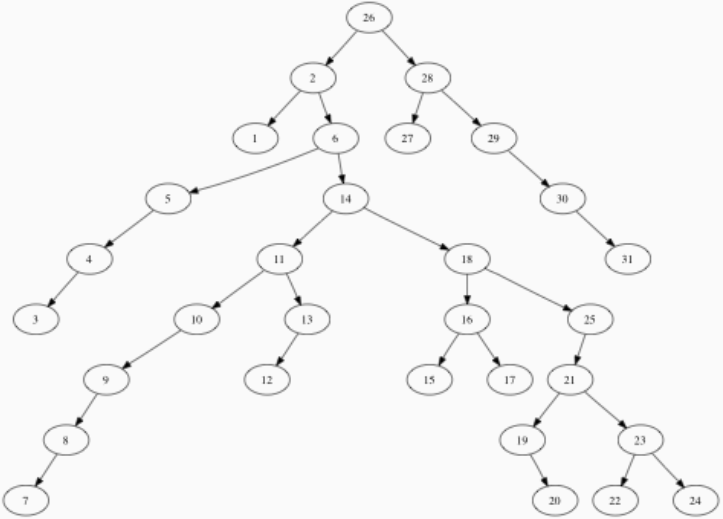
Random trees average 39% worse than perfect



Random trees average 39% worse than perfect



Random Tree average 39% worse than perfect



Building better trees: Off-line algorithm

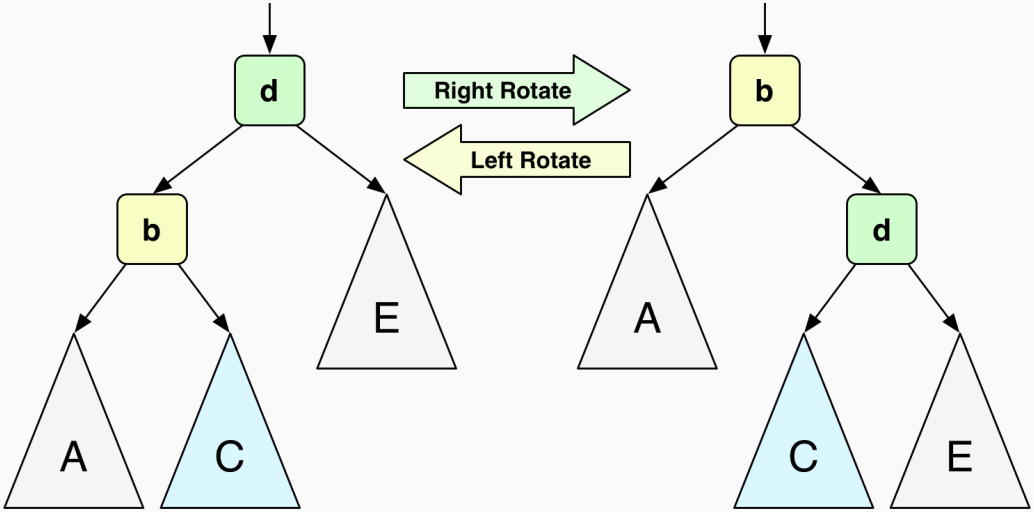
1. Take the inputs we want to put in the tree.
2. Randomly shuffle them.
3. Build tree by inserting in *shuffled* order.

Building better trees: Off-line algorithm

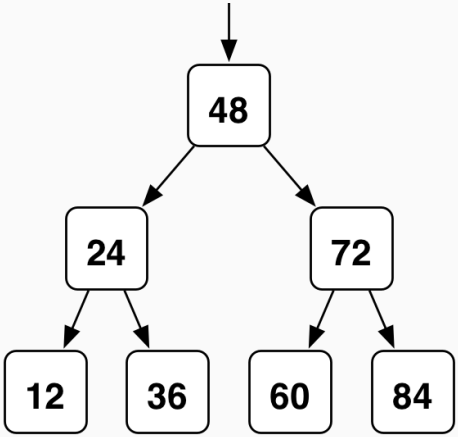
1. Take the inputs we want to put in the tree.
2. Randomly shuffle them.
3. Build tree by inserting in *shuffled* order.

What if we want to keep the tree balanced as new data comes in?

Tree Rotations



Insert 40. Rotate left at 36, left at 24, right at 48.



`insertAtRoot` pseudocode

`insertAtRoot` pseudocode

```
insertAtRoot(tree, x):
   if tree is empty:
      make x its new root.

   else if x < tree's root:
      insertAtRoot(left subtree, x)
      do right rotation at tree's root.

   else if tree's root < x:
      insertAtRoot(right subtree, x)
      do left rotation at tree's root.
```

Building better trees: Randomized Binary Trees

Idea: insert each new key "randomly" into the tree-so-far

- Maybe it should become the new root
- Maybe put it somewhere below the existing root

But how often to do each?

Building better trees: Randomized Binary Trees

Idea: insert each new key "randomly" into the tree-so-far

- Maybe it should become the new root
- Maybe put it somewhere below the existing root

But how often to do each?

Answer: If the tree has n nodes **before** the insert,

- do insert-at-root with probability $1/(n+1)$
- otherwise, insert randomly into the appropriate child.

Learning Targets

1. Given a tree, I can tell whether it's a valid BST.
2. I can simulate BST lookup, insert, and delete (on paper)
3. I can simulate left and right rotations (on paper)
4. I can simulate `insertAtRoot` (on paper)
5. I can simulate Randomized Binary Tree insertion.