Binary Search Trees (Continued)

- Study Project 3 Solution
- Balanced Binary Search Trees
- Balancing Operations
- Reading: L&C 11.1 – 11.4

  `http://algs4.cs.princeton.edu/33balanced`

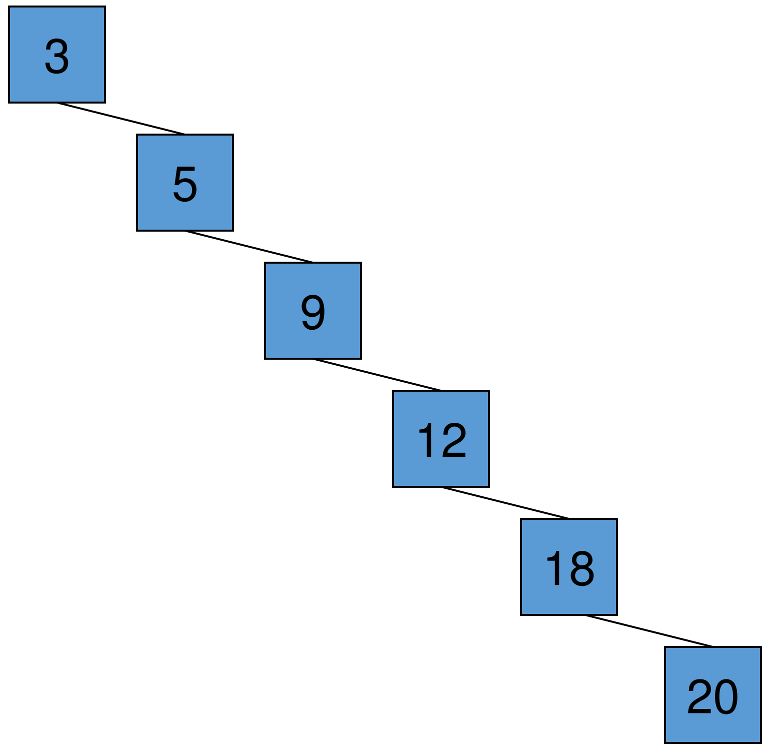
DRAFT

Balanced Binary Search Trees

- The balance of a binary search tree is important for obtaining its efficiency
- If we add 3, 5, 9, 12, 18, and 20 to a binary search tree, we get a degenerate tree
- This is less efficient than a singly linked list because our code needs to check the null left pointer at each level while traversing it
- Operations are O(n) instead of O(log n)
- We want our binary search trees balanced

Balanced Binary Search Trees

- Degenerate tree for a binary search tree



Balancing Operations

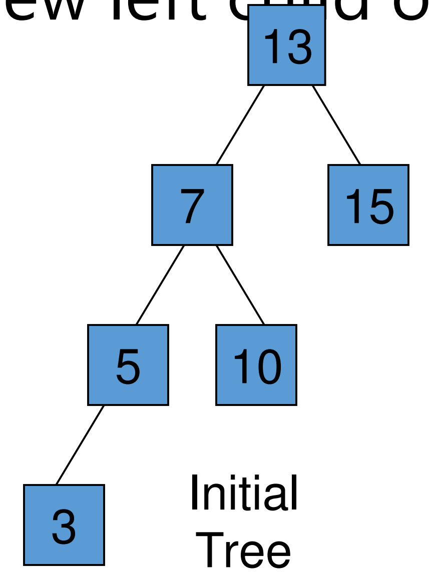
- Brute force balancing methods work but are unnecessarily time consuming
- We could use an in-order traversal of the tree and move everything out to an array
- Then, we could use a recursive method to insert the middle element of the array as the root and subdivide the array into two halves
- Eventually, we will rebuild a balanced tree

Balancing Operations

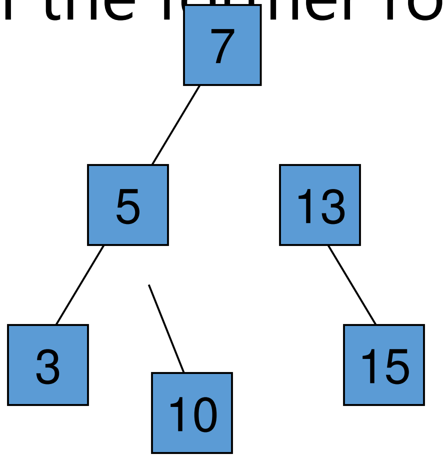
- We prefer to use balancing operations after each add or remove element operation
- Semantics of balancing operations
  - Right rotation
  - Left rotation
  - Rightleft rotation
  - Leftright rotation

Balancing Operations

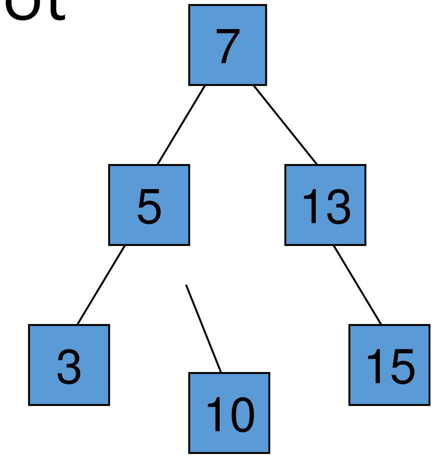
- Semantics of Right Rotation
  A. Make the left child of the root the new root
  B. Make former root the right child of the new root
  C. Make right child of the former left child of the former root the new left child of the former root



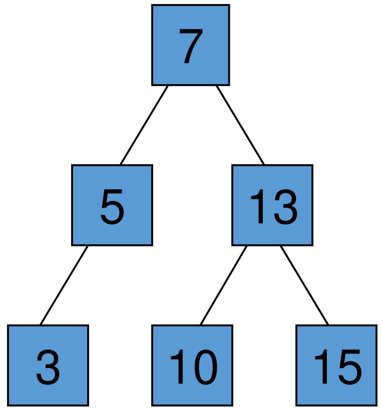
Initial Tree



Step A



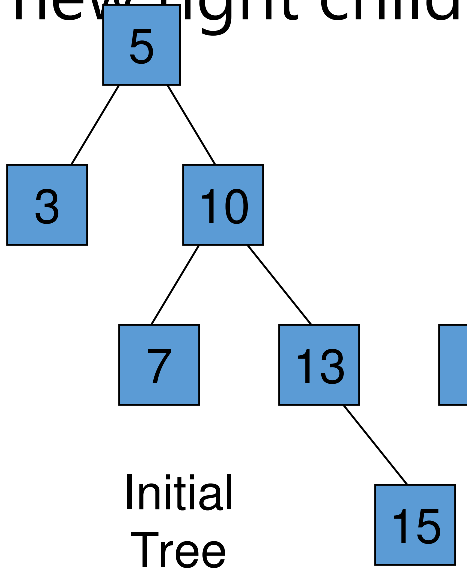
Step B



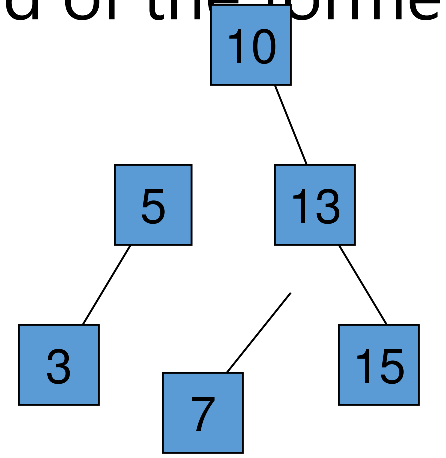
Step C

Balancing Operations

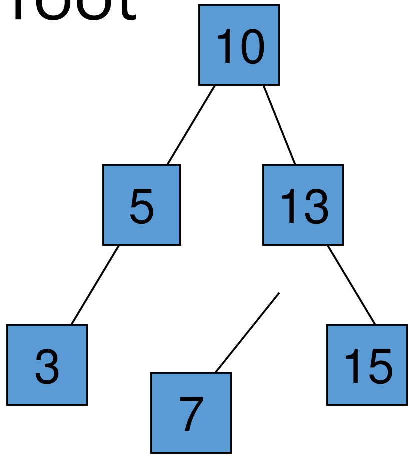
- Semantics of Left Rotation
  A. Make the right child of the root the new root
  B. Make former root the left child of the new root
  C. Make left child of the former right child of the former root the new right child of the former root



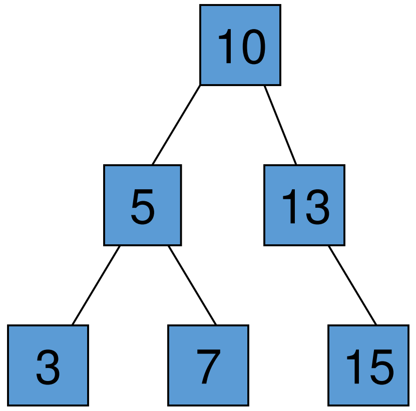
Initial Tree



Step A



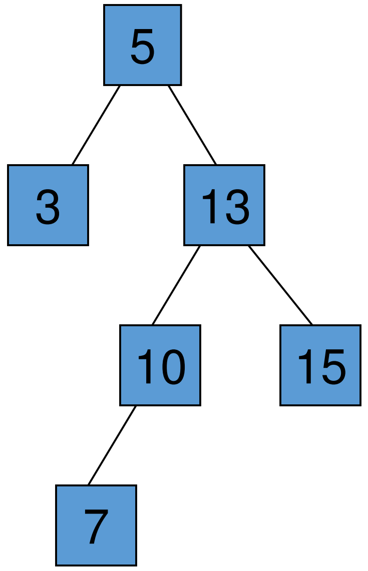
Step B



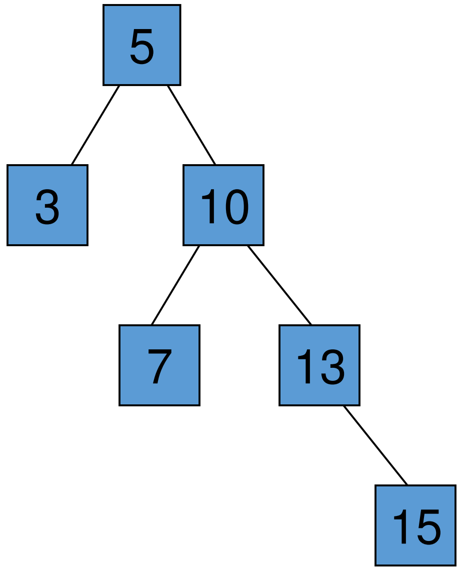
Step C

Balancing Operations

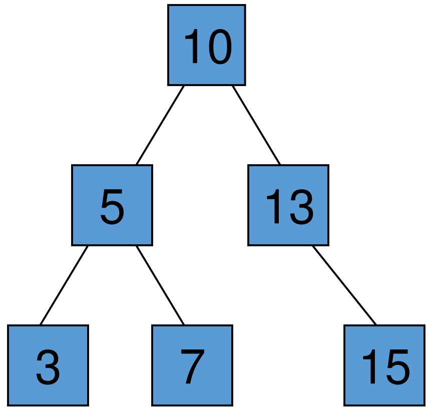
- Semantics of Rightleft Rotation
  A. Right rotation around right child of root
  B. Left rotation around root



Initial
Tree



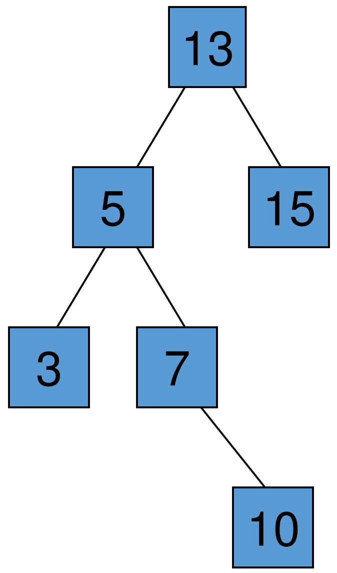
After Right
Rotation



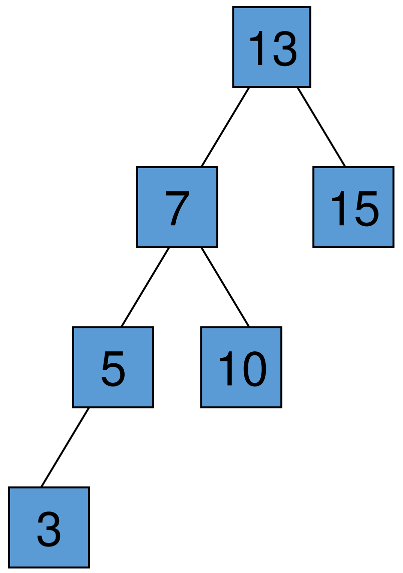
After Left
Rotation

Balancing Operations

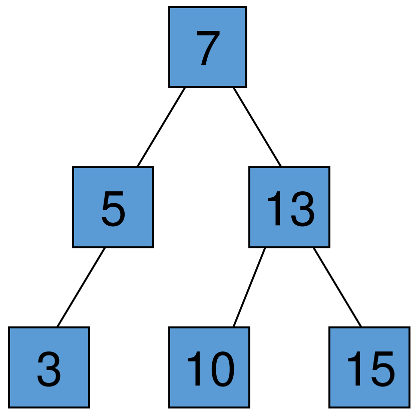
- Semantics of Leftright Rotation
  A. Left rotation around left child of root
  B. Right rotation around root



Initial Tree



After Left Rotation



After Right Rotation

Introduction to Project 4

- In the next lecture we will cover the AVL strategy for determining which rotation to perform on a binary search tree after an add or remove operation
- In project 4, I provide you a framework for an AVL tree implementation
- However, it is missing a few key pieces:
  - Determining which of the rotations to perform
  - Updating of the balance factors after a rotation

Introduction to Project 4

- You need to add code for the missing pieces
- I provide junit test cases to determine if your code works correctly for samples of all of the different situations that can occur
- Your code must pass the test cases