SELF-BALANCING SEARCH TREES (AVL TREES)

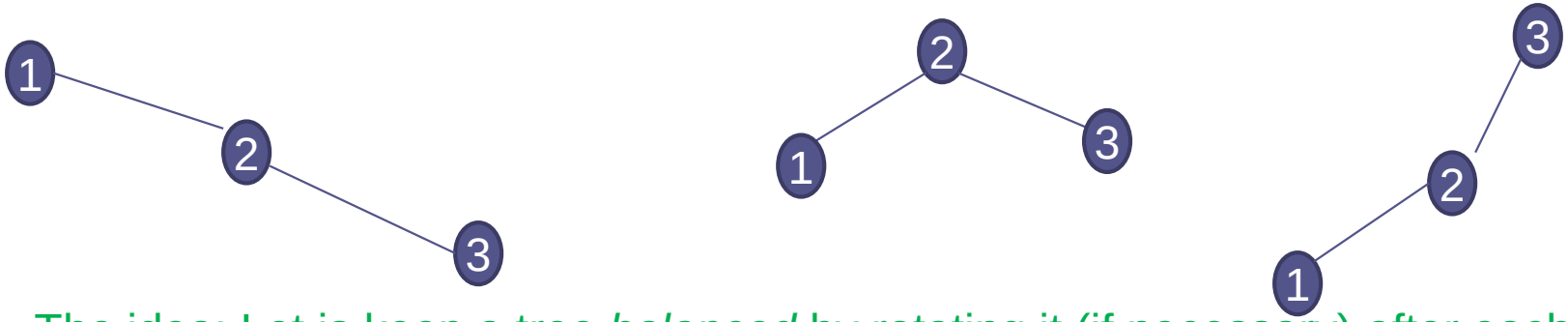
Lecture 35

Assignment

- Start reading Chapter 9
- Be able to repeat what we have discussed *without* looking at the notes or in the book

Self-Balancing Search Trees

- The performance of a binary search tree is proportional to the *height of the tree*
- A full binary tree of height *k* can hold $n = 2^k - 1$ items
- If a binary search tree is full and contains *n* items, each an insertion and a search takes O(log *n*) comparisons
- If a binary tree is not full, these operations can take *O(n)* comparisons
- A given sorted sequence A, however, can be stored in different trees:



- The idea: Let is keep a tree *balanced* by rotating it (if necessary) after each insertion

AVL Trees

- In 1962 G.M. Adelson-Velskii and E.M. Landis developed a self-balancing-tree algorithm.
- The actual development was in the Artificial Intelligence research (spearheaded in the USSR by A. S. Kronrod and, in the US, by John McCarthy [MIT])
- Playing chess was the experiment
- See https://chessprogramming.wikispaces.com/Georgy+Adelson-Velsky

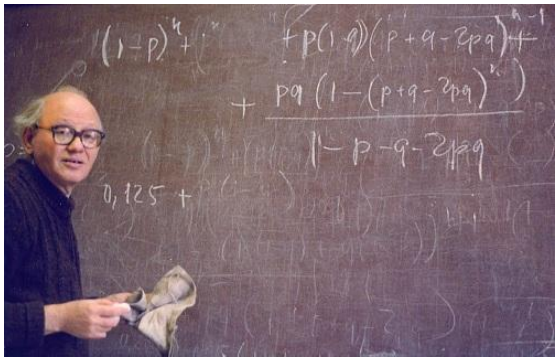
About computer chess competion

- In1966 a four game match began between the Kotok-McCarthy-Program, running on an IBM 7090 computer, and the ITEP Chess Program on a Soviet M-2 computer.
- The match played over nine months was won 3-1 by the ITEP program.
- In 1971, Georgy Adelson-Velsky became the primary author of Kaissa, winner of the first computer chess championship 1974 in Stockholm.

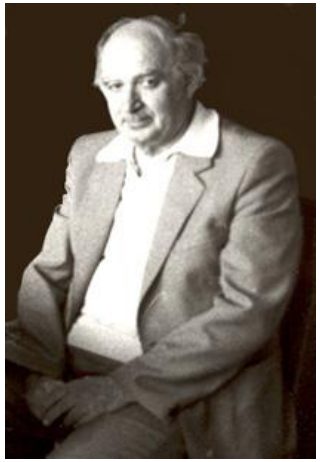
1974 International Chess Competition

Adelson-Velski and Landis

Georgy Maximovich Adelson-Velsky (1922-2014)



Evgenii Mikhailovich Adelson-Velsky (1921-1997)



Alexandr Semyonovich Kronrod (1921 –1986)

John McCarthy and Alan Kotok

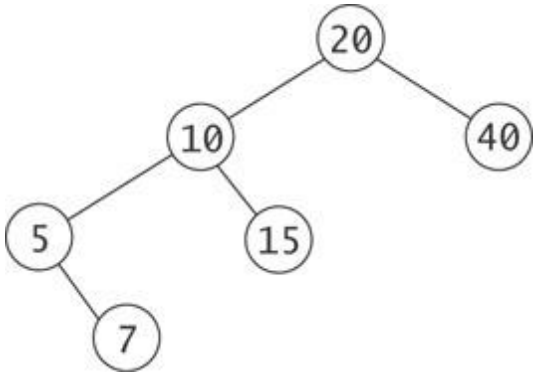
John McCarthy (1927 – 2011)

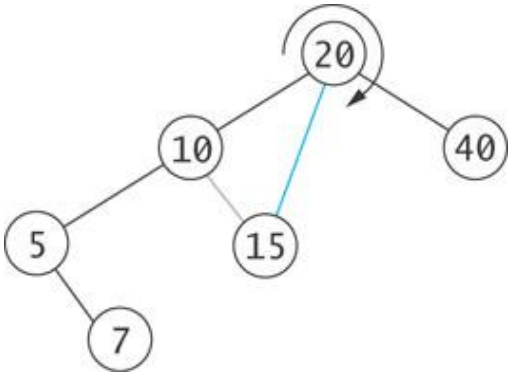
Alan Kotok (1941 – 2006)

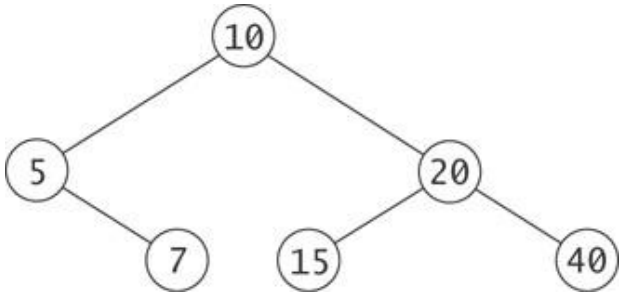
Adelson-Velski playing chess with John McCarthy (1979)

Rotation

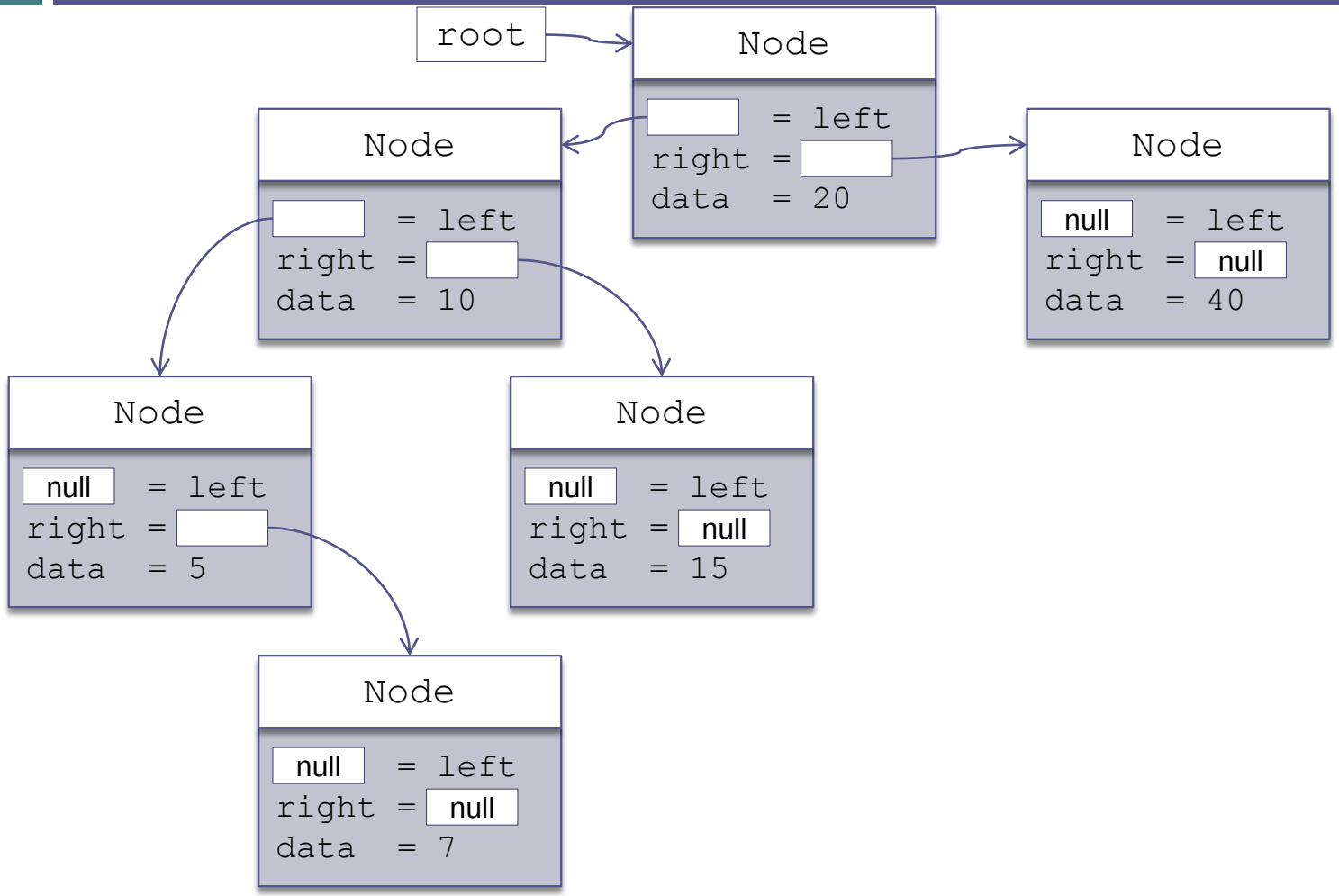
We need an operation on a binary tree that changes the relative heights of left and right subtrees, but preserves the binary search tree property



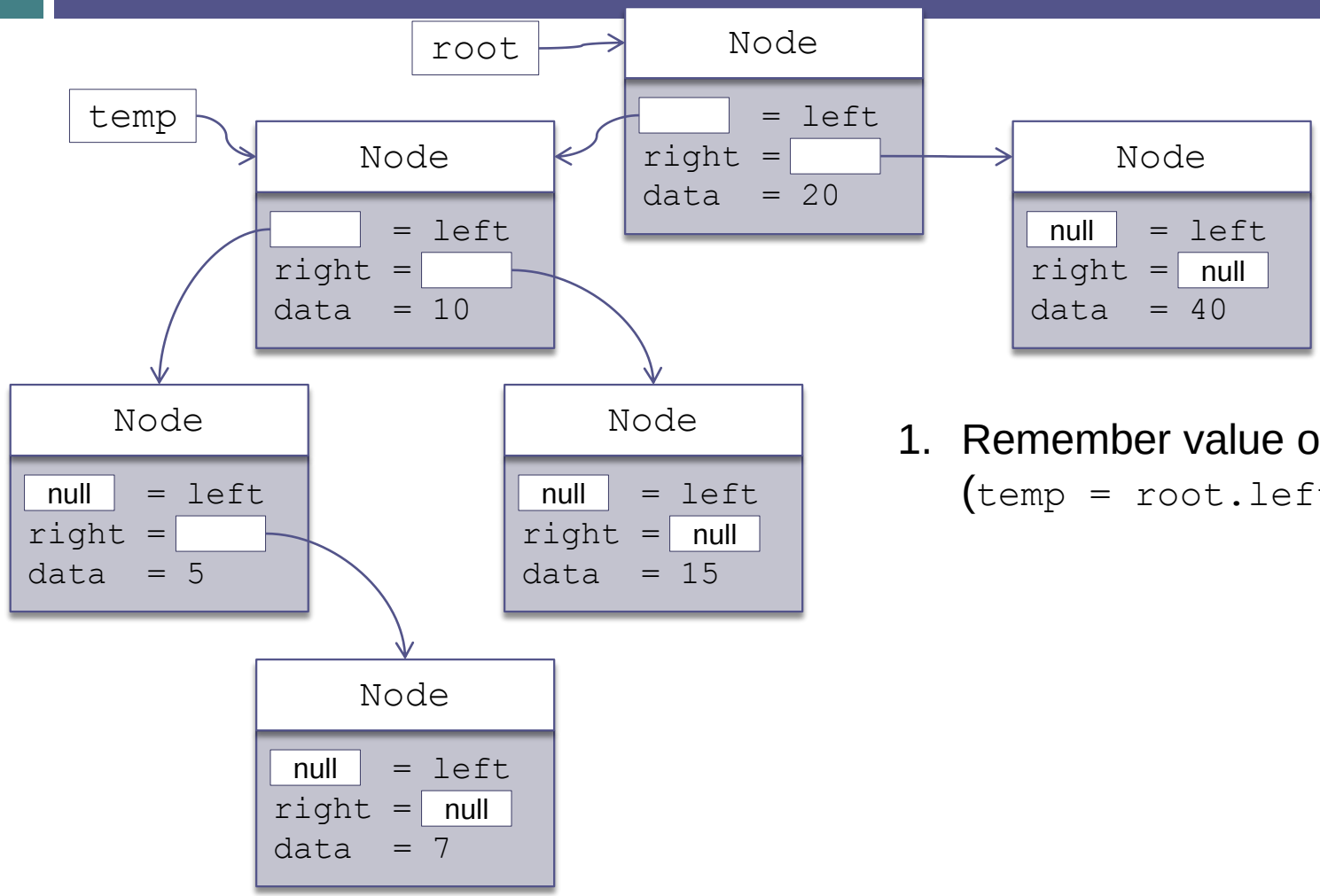




Algorithm for Rotation

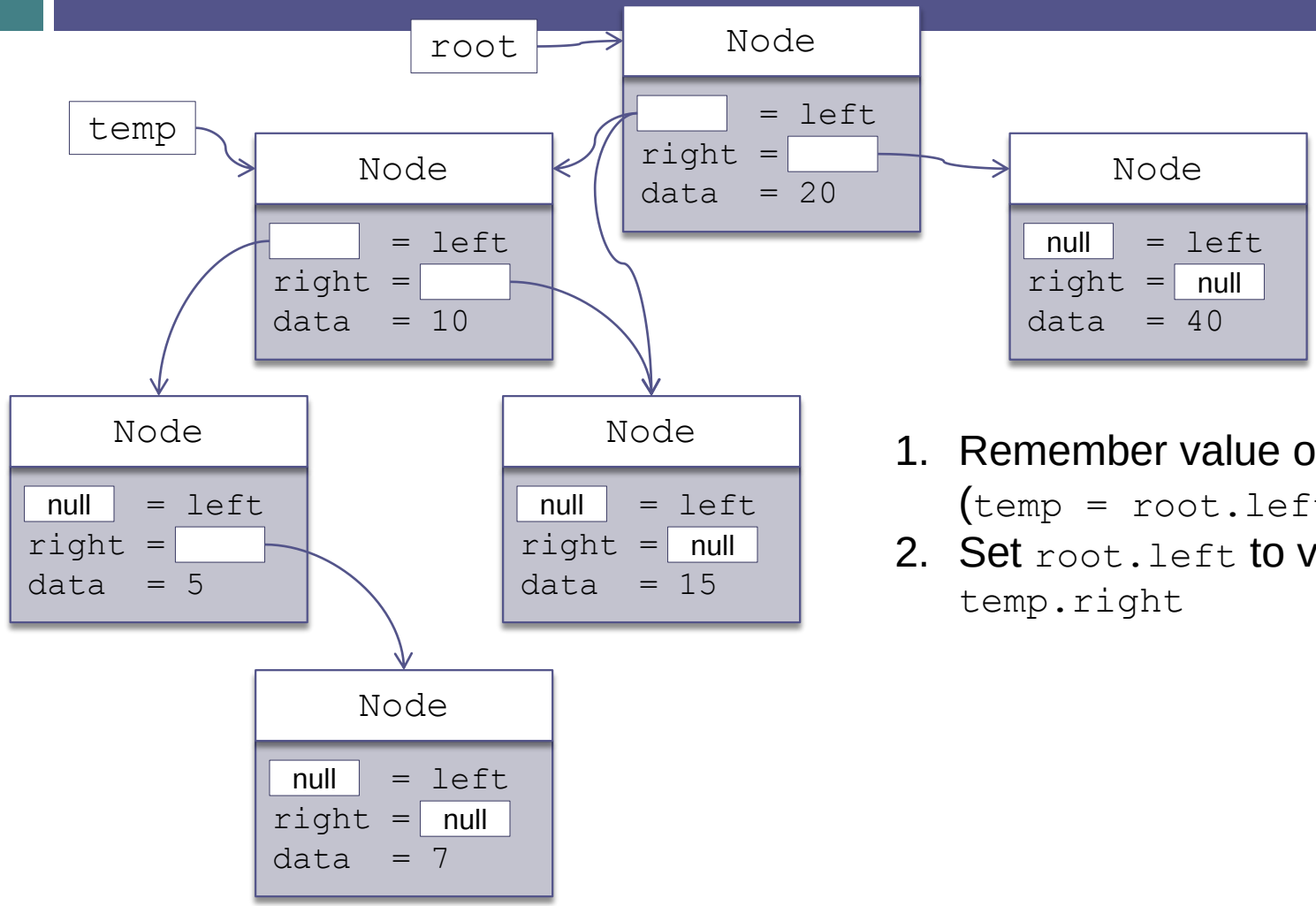


Algorithm for Rotation (cont.)



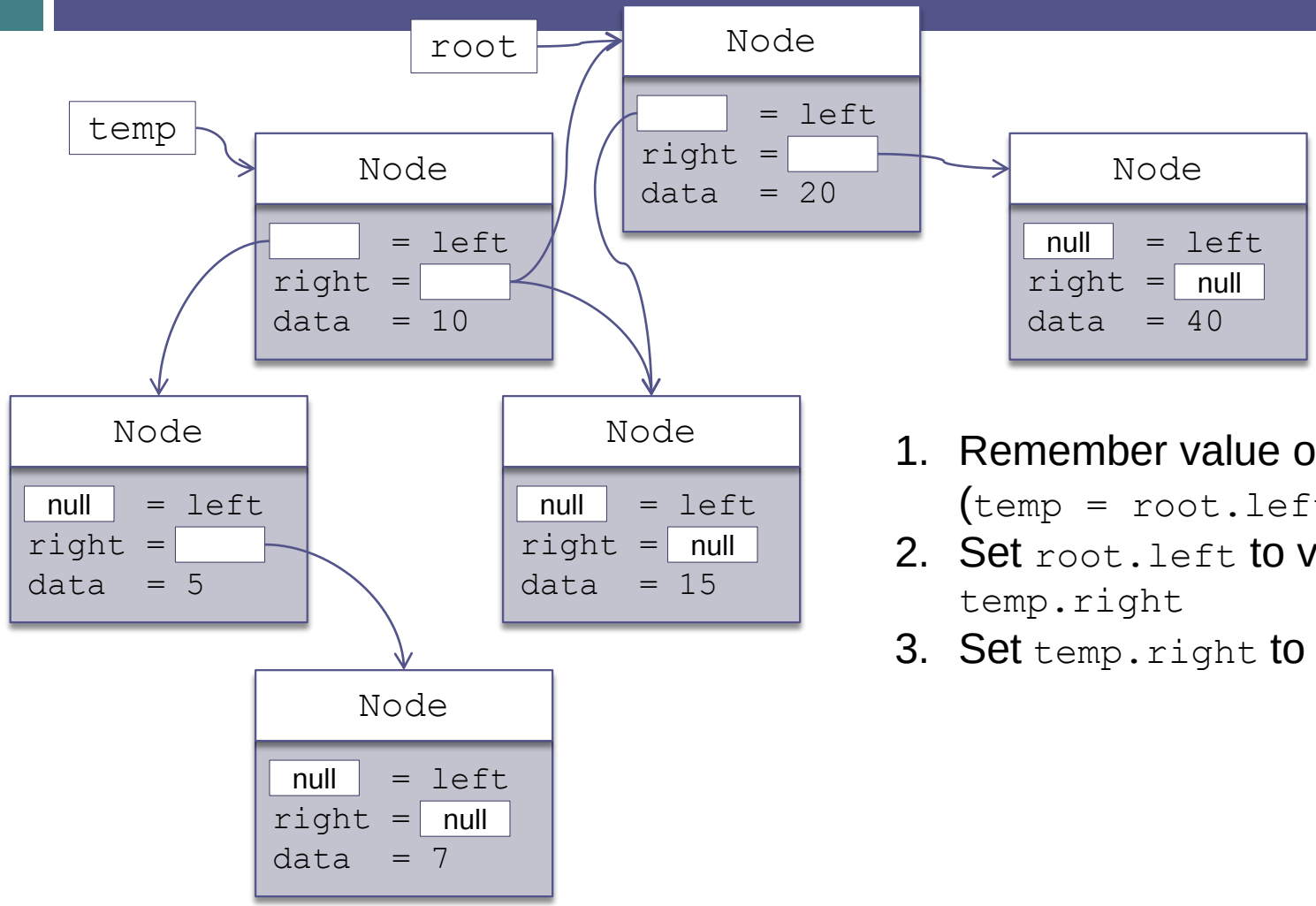
1. Remember value of `root.left` (`temp = root.left`)

Algorithm for Rotation (cont.)



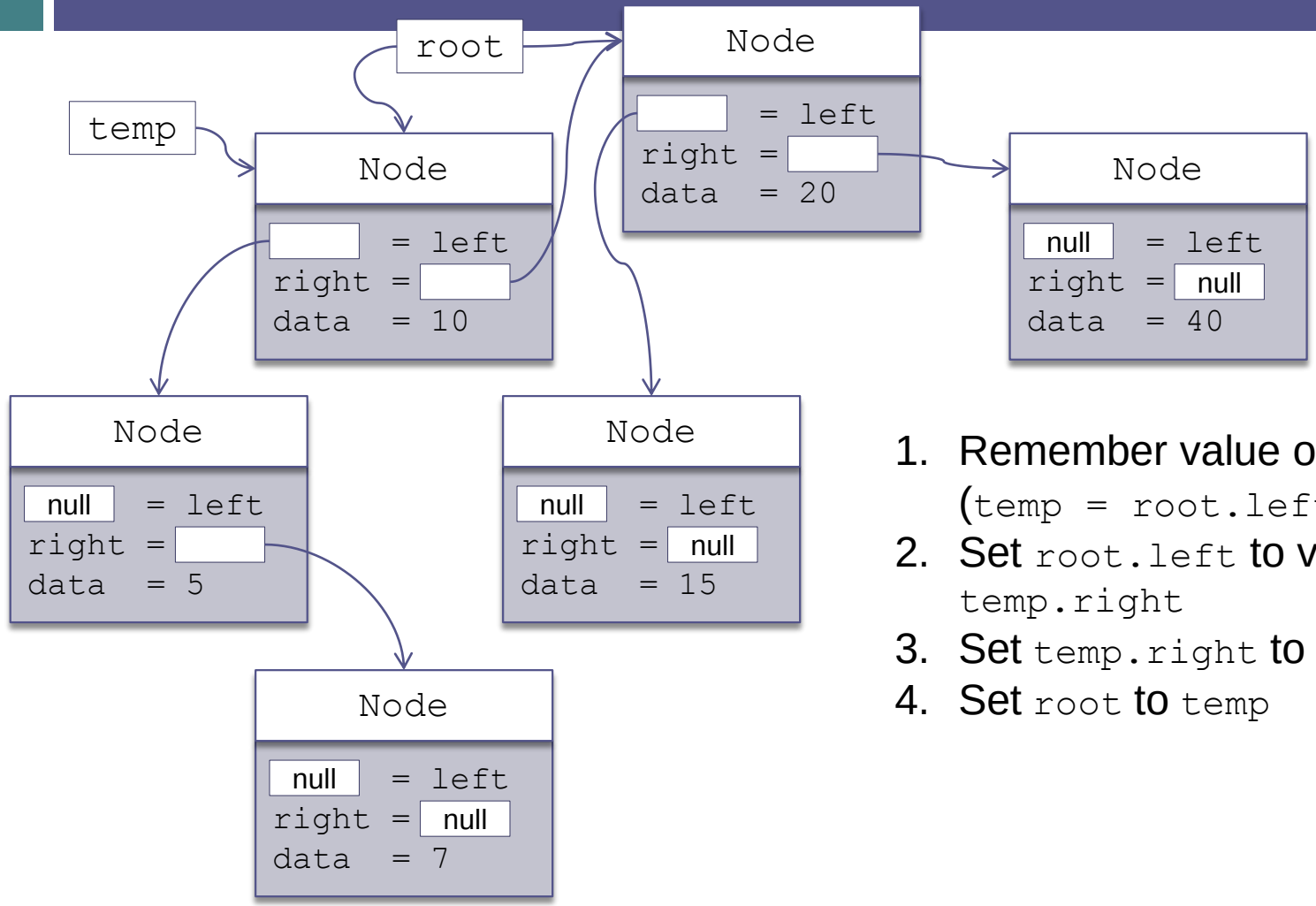
1. Remember value of `root.left` (`temp = root.left`)
2. Set `root.left` to value of `temp.right`

Algorithm for Rotation (cont.)



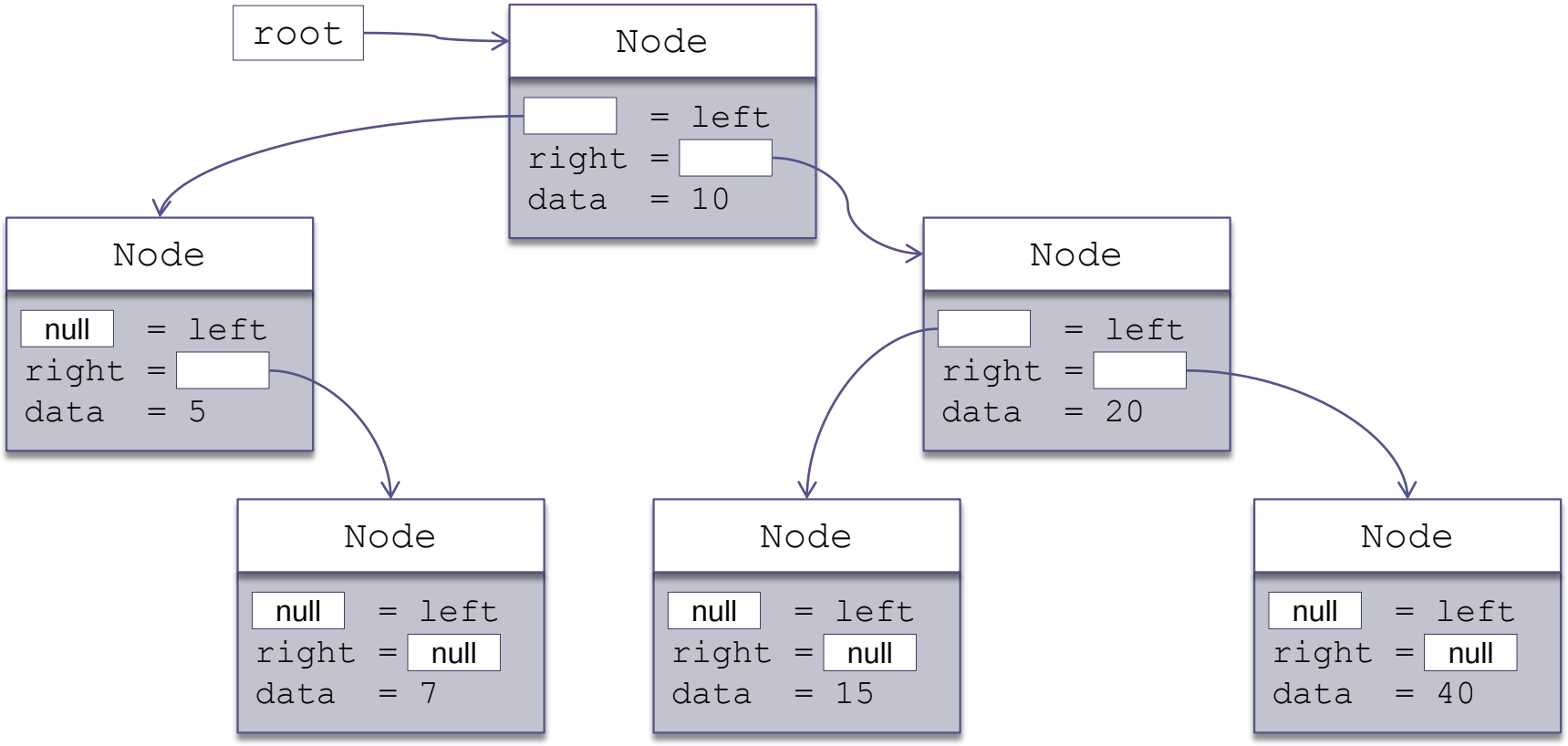
1. Remember value of `root.left` (`temp = root.left`)
2. Set `root.left` to value of `temp.right`
3. Set `temp.right` to `root`

Algorithm for Rotation (cont.)

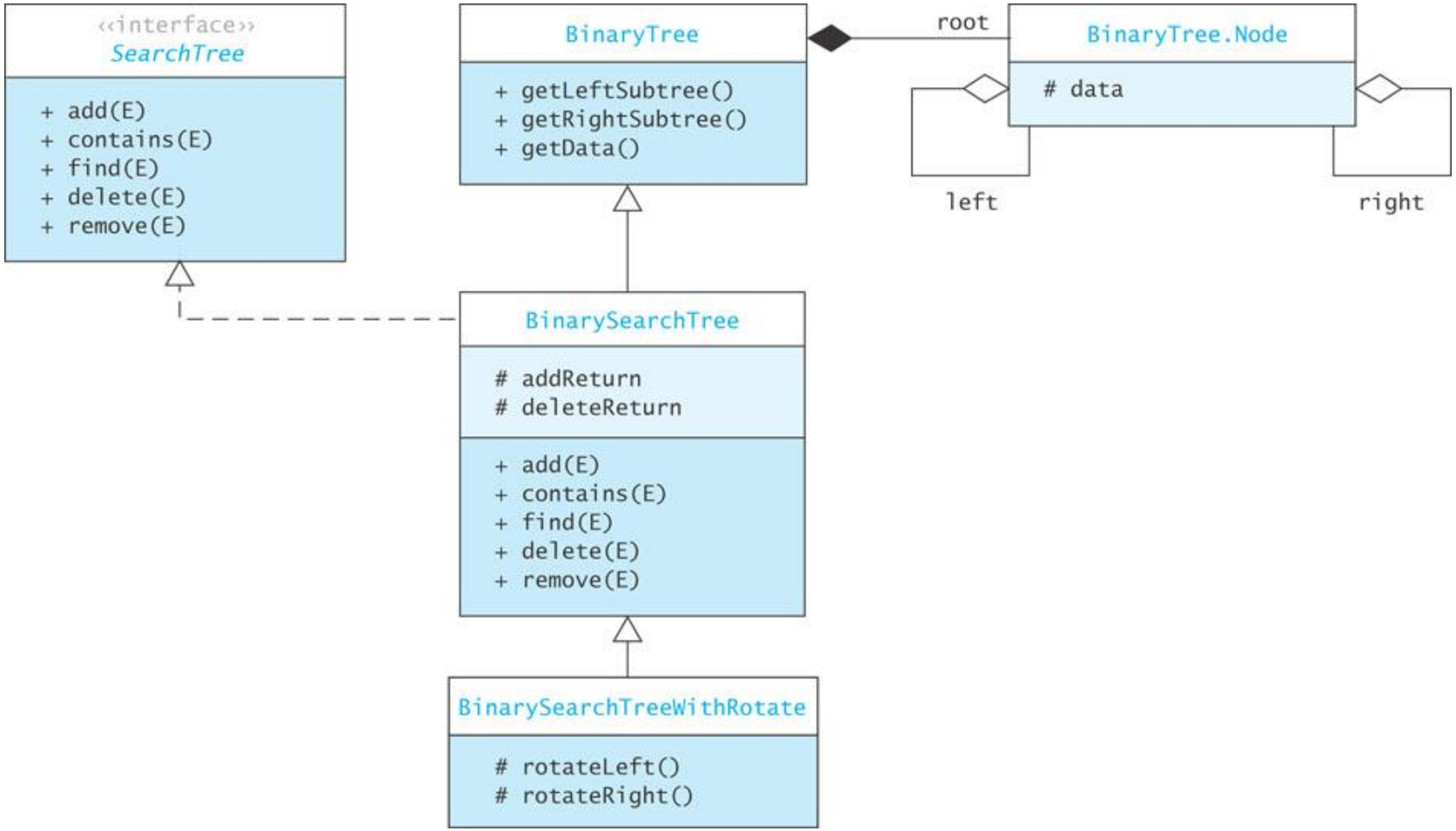


1. Remember value of `root.left` (`temp = root.left`)
2. Set `root.left` to value of `temp.right`
3. Set `temp.right` to `root`
4. Set `root` to `temp`

Algorithm for Rotation (cont.)



Implementing Rotation



Implementing Rotation (cont.)

- Listing 9.1 (`BinarySearchTreeWithRotate.java`, page 476)