School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

AVL Trees

Review - Binary Search Tree

- A binary search tree (BST) is a binary tree with the following properties:
  - Node values are distinct and comparable
  - The left subtree of every node contains only values that are *less than* the node's own value.
  - The right subtree of every node contains only values that are *greater than* the node's own value.
- Basic Operations:
  - Search
  - Min and Max
  - Insert
  - We will see Remove today

Searching

- Problem: Search whether a value exists in a dataset.
- One suitable data structure for this problem is sorted array (assuming the values are orderable).
  - Searching takes logarithmic time instead of linear time of linked list.
  - However, insertion and deletion are expensive. (Shifting array elements often takes linear time.)
- Ordered tree or Binary search tree is an easy-to-implement data structure, under which searching, insertion, and deletion **all take logarithmic time on average**.
  - All are done in O(height), but height can be $\Omega(n)$ in worst case

Binary Search Tree

- Basic Operations:

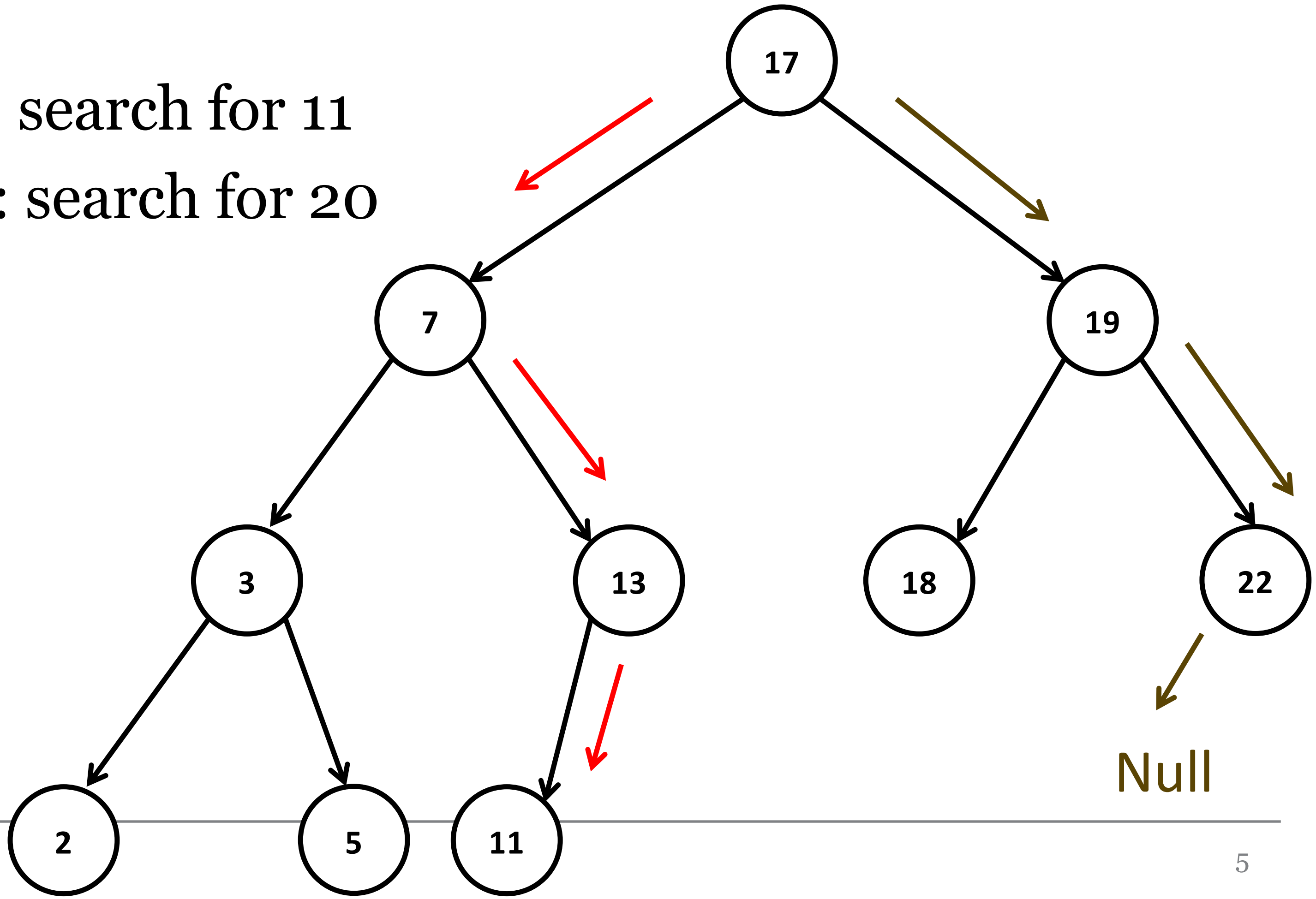
- Search
- Min and Max
- Insert
- Remove

- Given n items, how much will it take to build the whole binary search tree?

BST - Searching

- This operation returns true if there is a node in tree T that has value X, or false if there is no such node.

- Example 1: search for 11
- Example 2: search for 20



BST - Searching

- This operation returns true if there is a node in tree T that has value X, or false if there is no such node.

- Start from root
- If current subtree is empty, return not found
- If target value = current value, return found
- If target value < current value, go left
- If target value > current value, go right

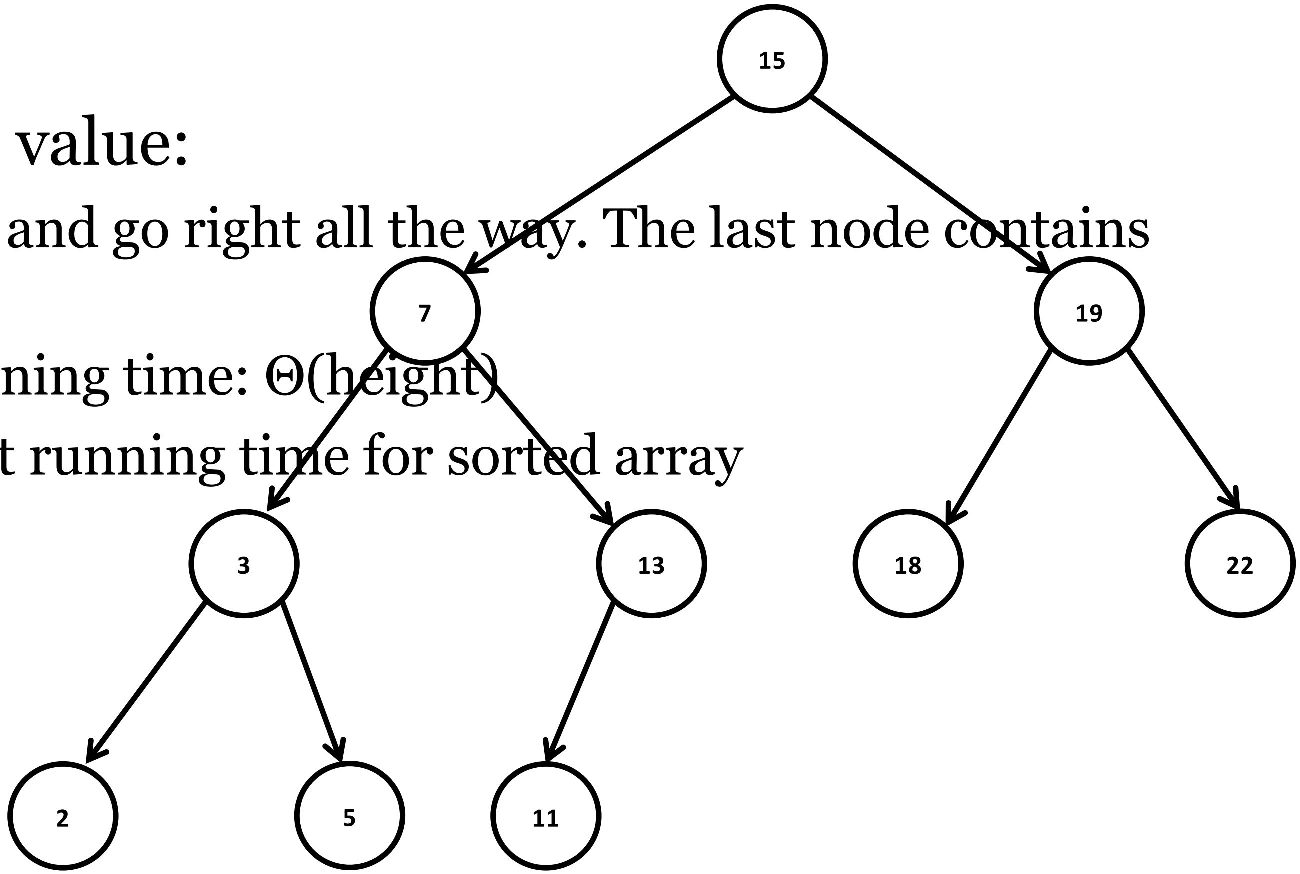
BST - Searching

- Which of the following best describes the worst-case running time of searching under a BST with n nodes?
  - $\Theta(n)$
  - $\Theta(\log(n))$
  - $\Theta(\text{height})$
  - $\Theta(1)$

BST – Min and Max

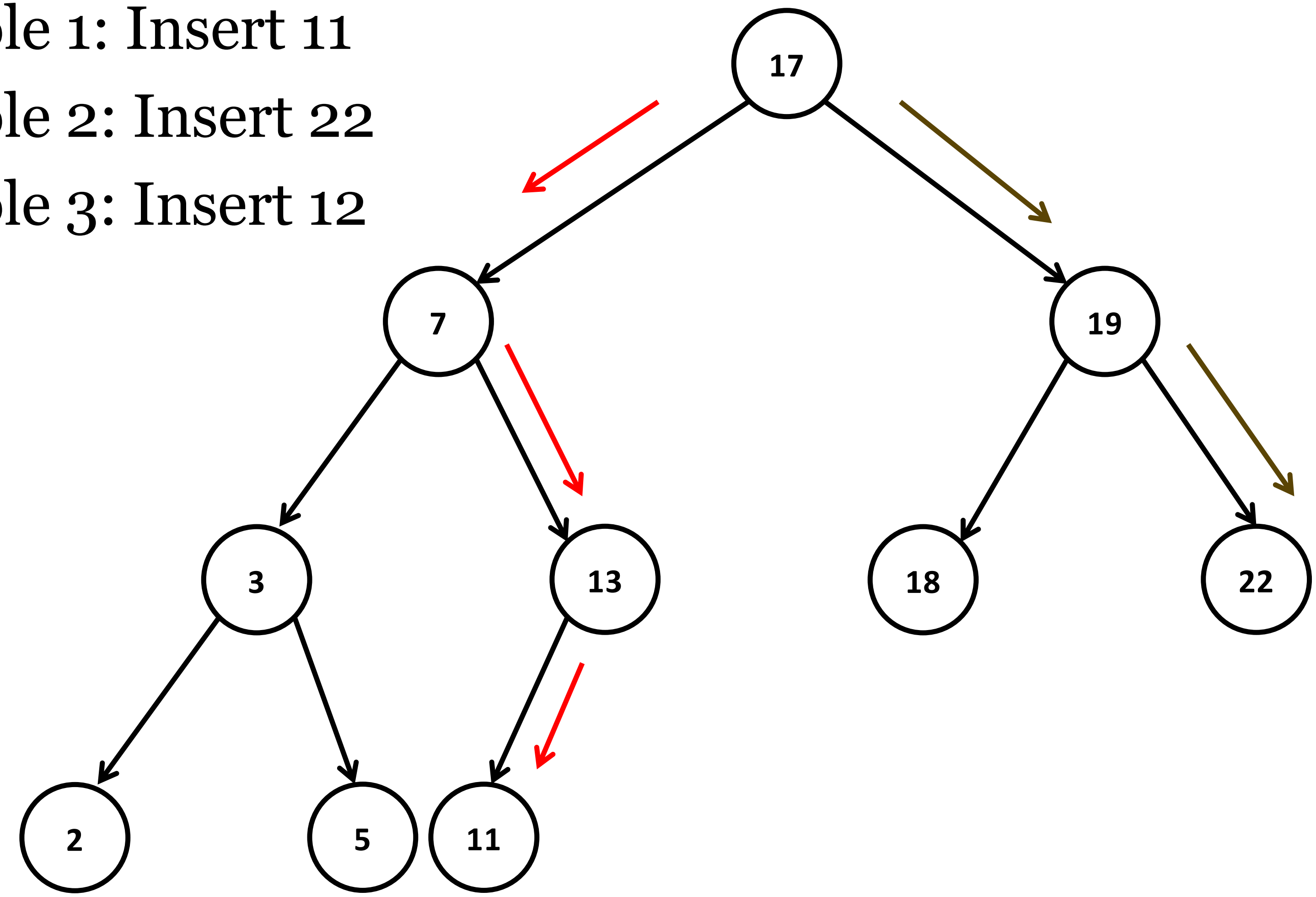
- The operation returns the node containing the smallest or largest elements in the tree.

- To find the max value:
  - Start from root and go right all the way. The last node contains the max value.
  - Worst-case running time: Θ(height)
  - Versus constant running time for sorted array
- Similar for min.



BST - Insertion

- Example 1: Insert 11
- Example 2: Insert 22
- Example 3: Insert 12



BST - Insertion

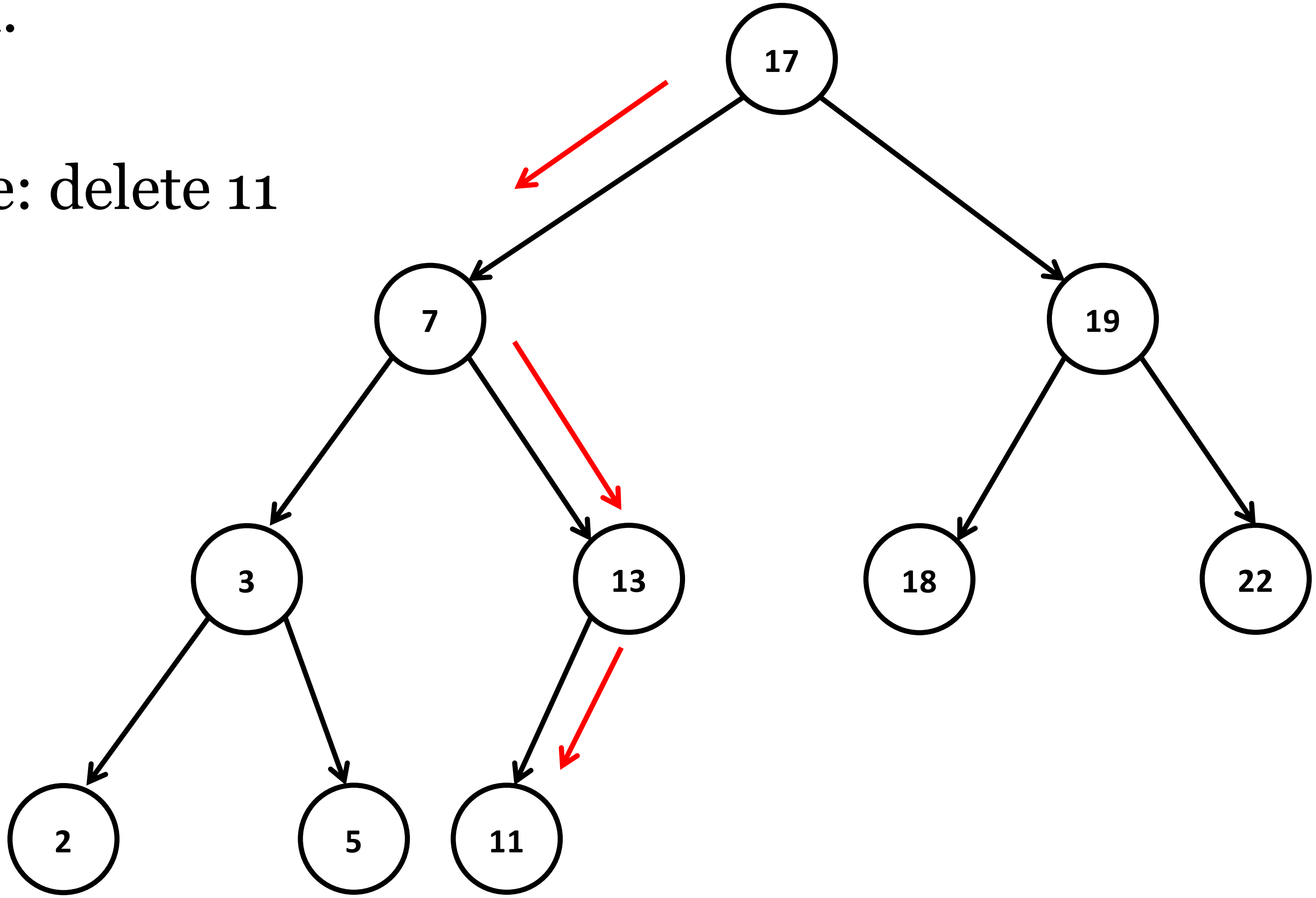
- Start from root
- If current subtree is empty, create new node here.
- If target value = current value, terminate.
- If target value < current value, go left.
- If target value > current value, go right.

- What is the worst-case running time of insertion under a BST with n nodes?
  - Θ(height)

BST - Deletion

- Case 1: the node to be deleted does not have any children.

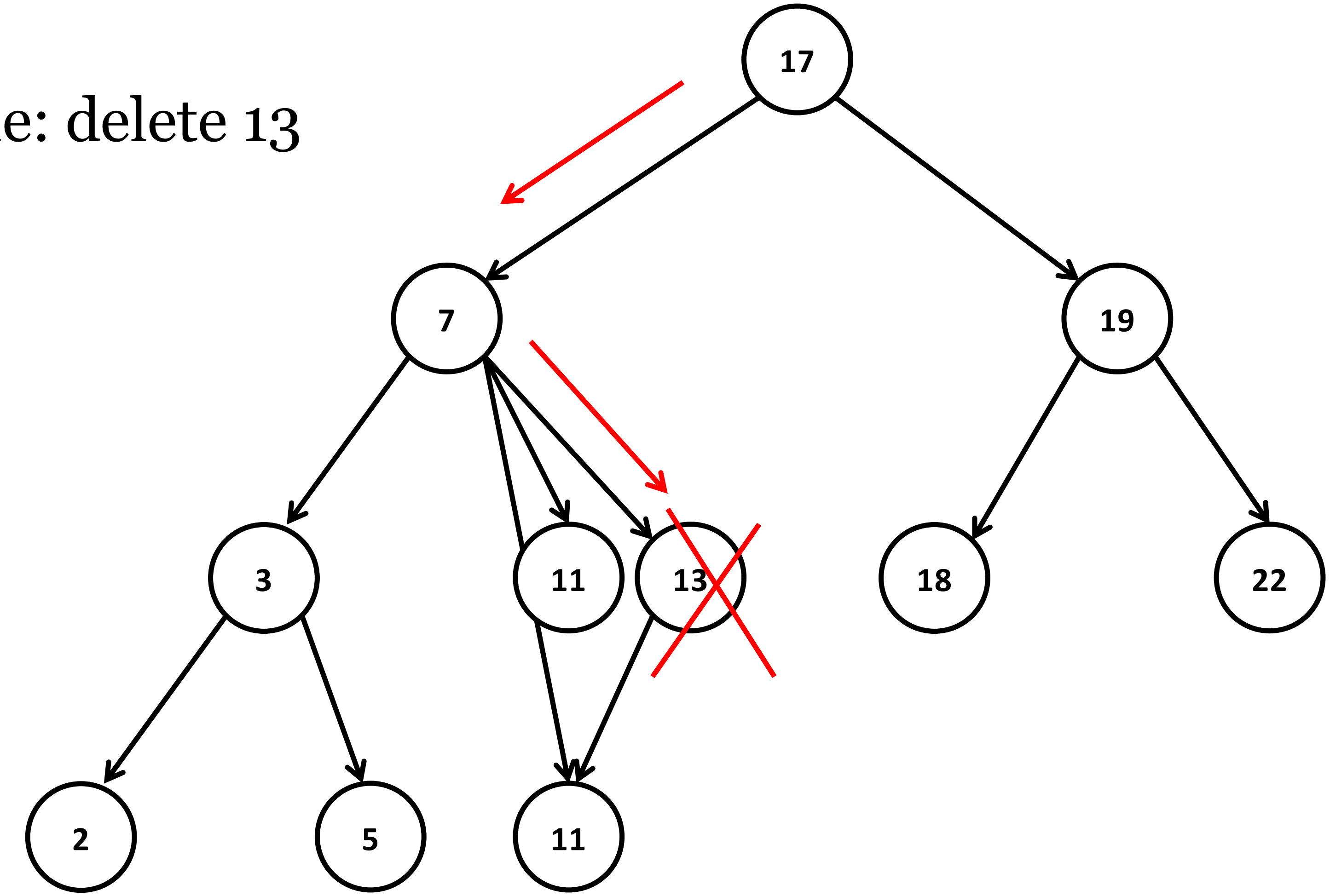
- Example: delete 11



BST - Deletion

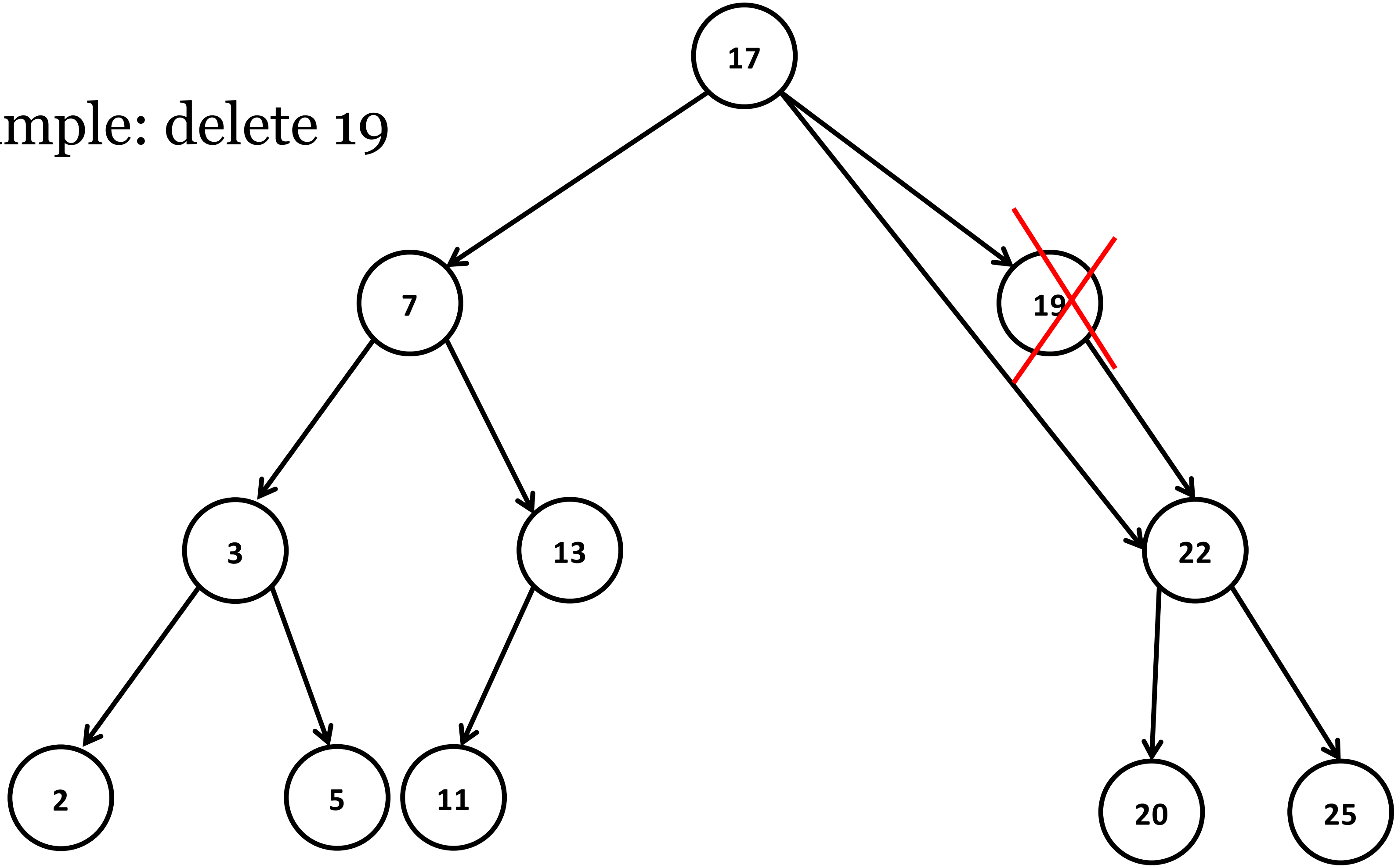
- Case 2: the node to be deleted has one child.

- Example: delete 13



BST - Deletion

- Case 2: the node to be deleted has one child.
- Example: delete 19



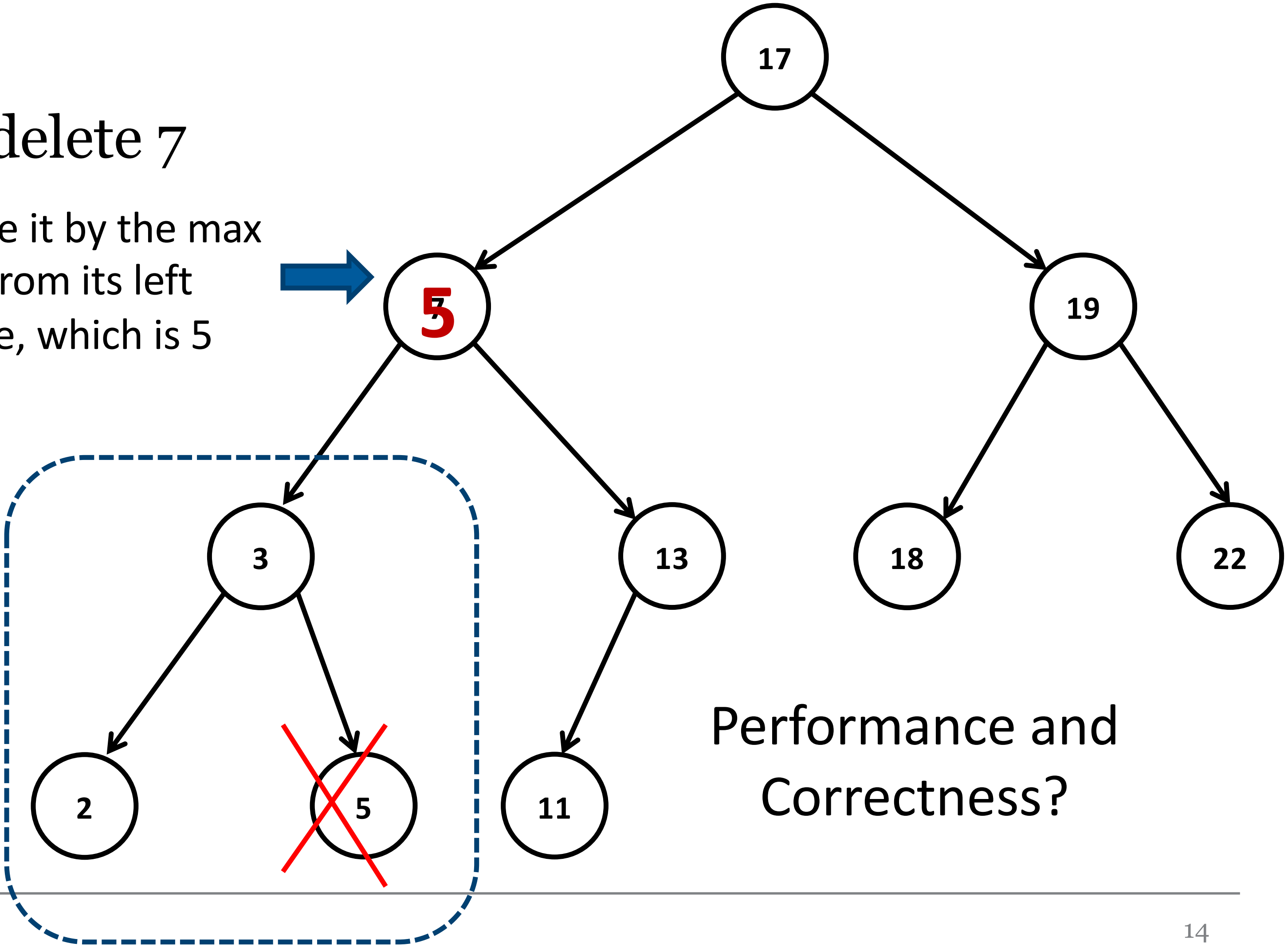
BST - Deletion

- Case 3: the node to be deleted has both children.

- Example: delete 7

Replace it by the max value from its left subtree, which is 5

Then delete 5 from its left subtree (case 1 or 2)



Performance and Correctness?

BST - Performance

- Searching, insertion, and deletion all take Θ(height) time in the worst case.
- Height is at most n-1.
- If height is k, then n is at most 1+2+...+2^k = 2^(k+1)-1.
  - n <= 2^(k+1)-1
  - k >= log(n+1)-1
  - Height is at least logarithmic in n.
- **[Fekete et al. 10]**: If the insertion order is random, then experimentally, BST's average height is less than 2.989 log(n).
- Therefore, in some sense, we can claim that for BST, searching, insertion, and deletion all take logarithmic time **on average**. (All three operations take linear time in the worst case).

Self-balancing BSTs

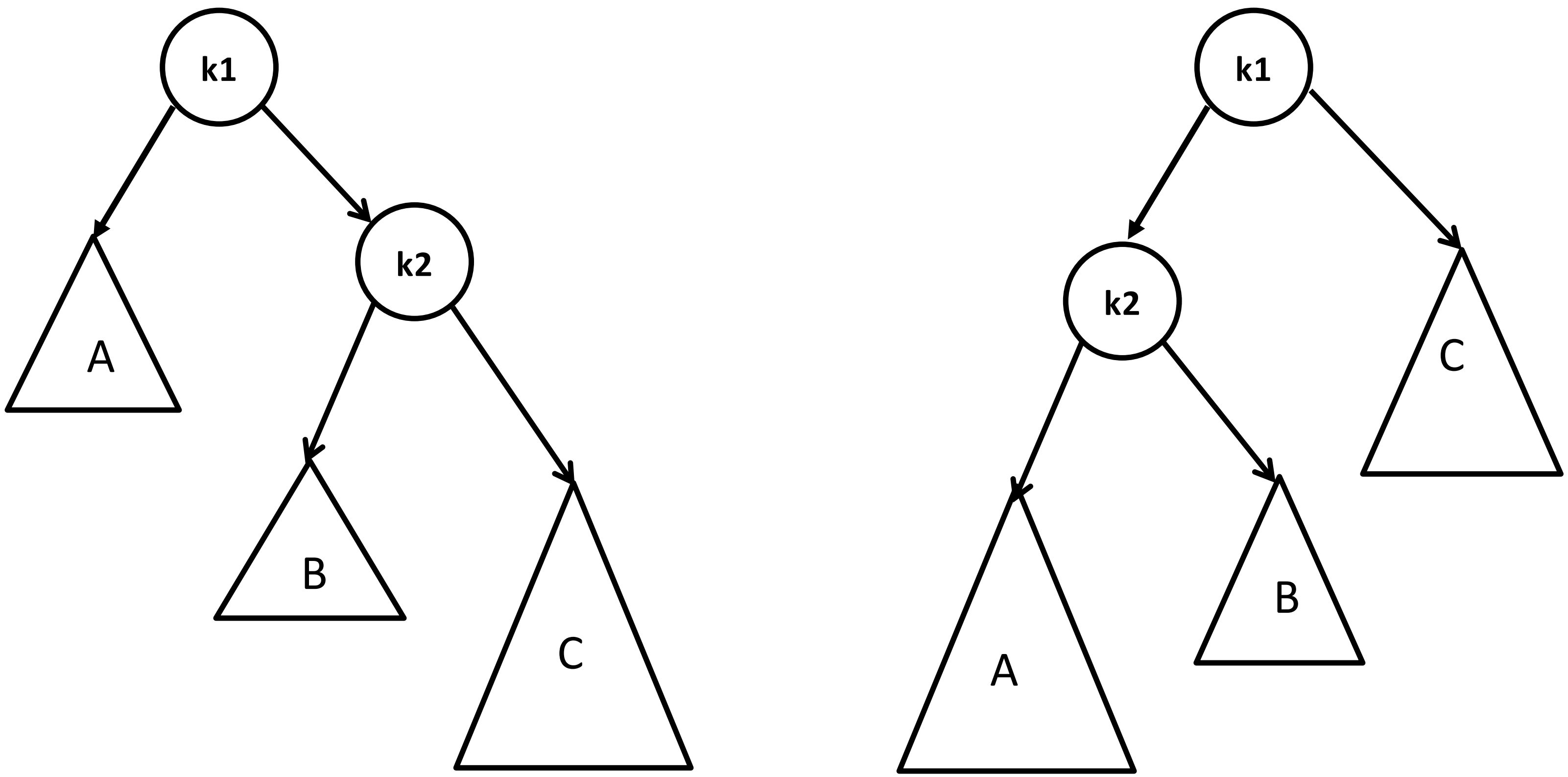
- A self-balancing BST automatically keeps its structure balanced.
- Example: **AVL tree**
- An AVL tree is a BST with a balance condition
  - For every node, the heights of two child subtrees can only differ by at most 1. See examples.
  - After insertion / deletion, if the above property is violated, then some housekeeping is needed to restore the property, which takes O(log n) extra time.
  - Since the tree is always fairly balanced, searching, insertion, and deletion all take logarithmic time in the worst case.
    - The height is not minimized, but still in O(log n)
- Examples of balanced trees with this definition

AVL trees

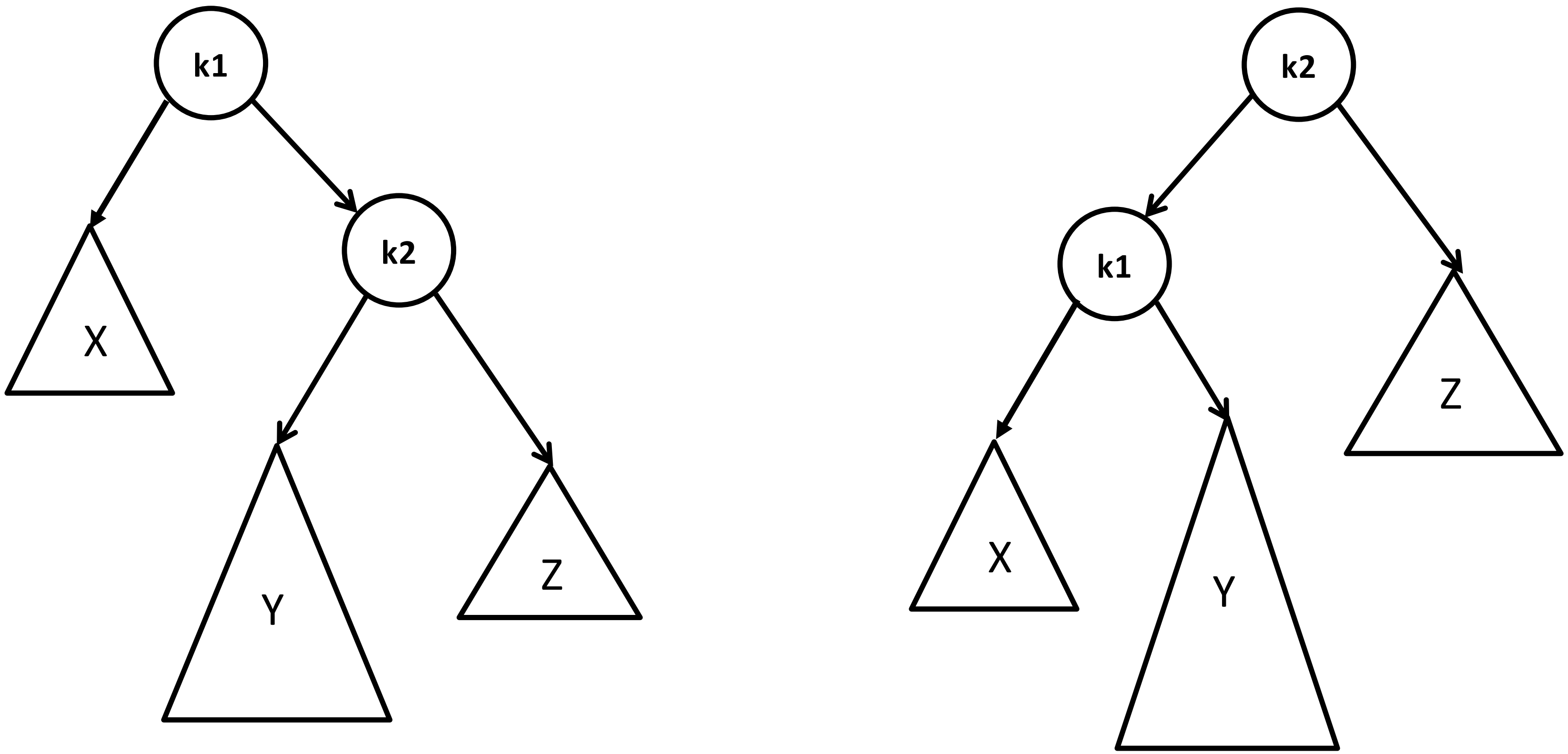
- Two definitions for Height of a tree in references: The number of nodes or the number of edges on the longest path from root to a leaf
  - Height of empty subtree 0 or -1
  - Either way, for AVL trees the height difference of two subtrees of each node is important

- What we need to see
  - With that condition (for every node, the heights of two child subtrees can only differ by at most 1), is the maximum height really O(log n)?
  - How to restore this property after insertion or deletion in O(log n)
    - How would it look like?

AVL Trees

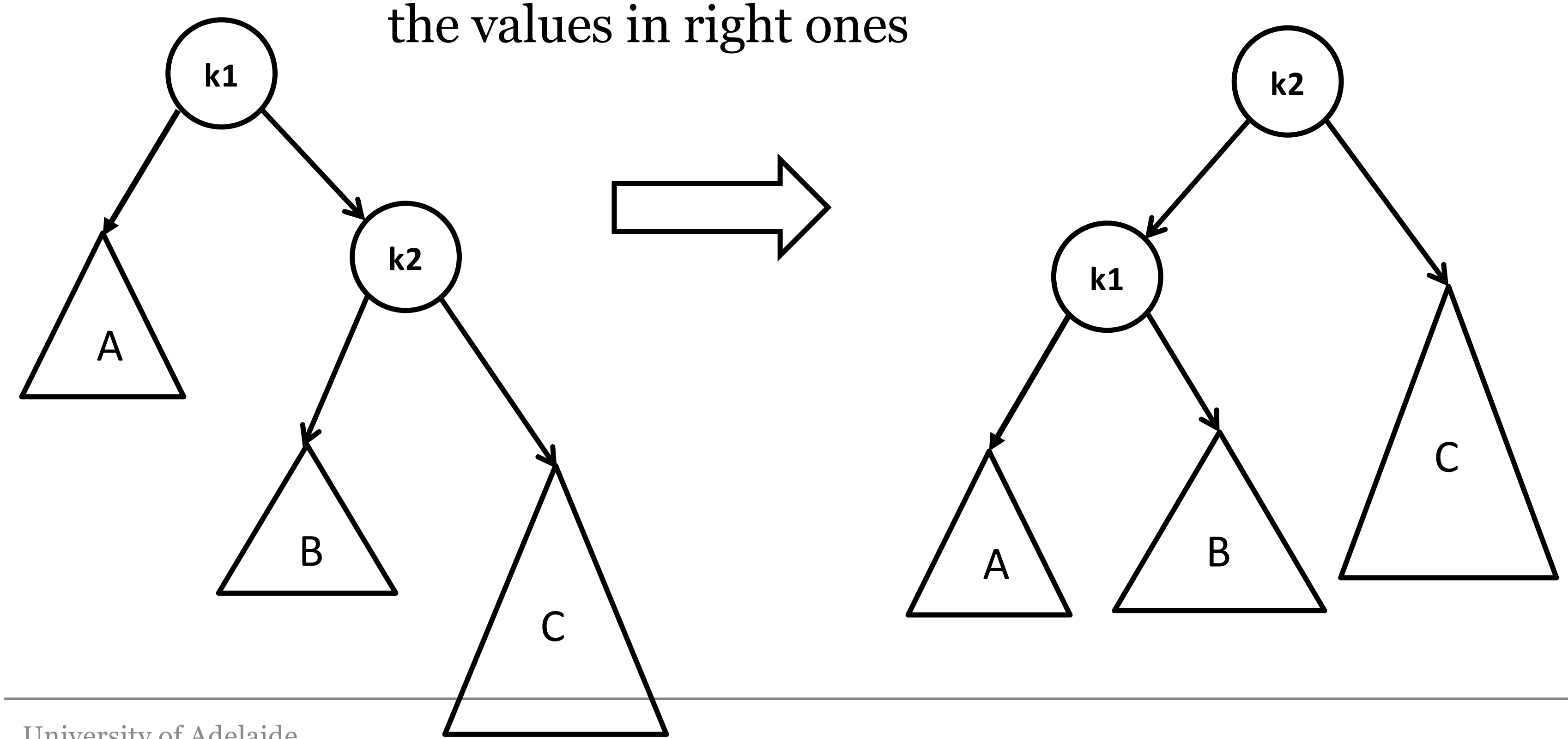


AVL Trees

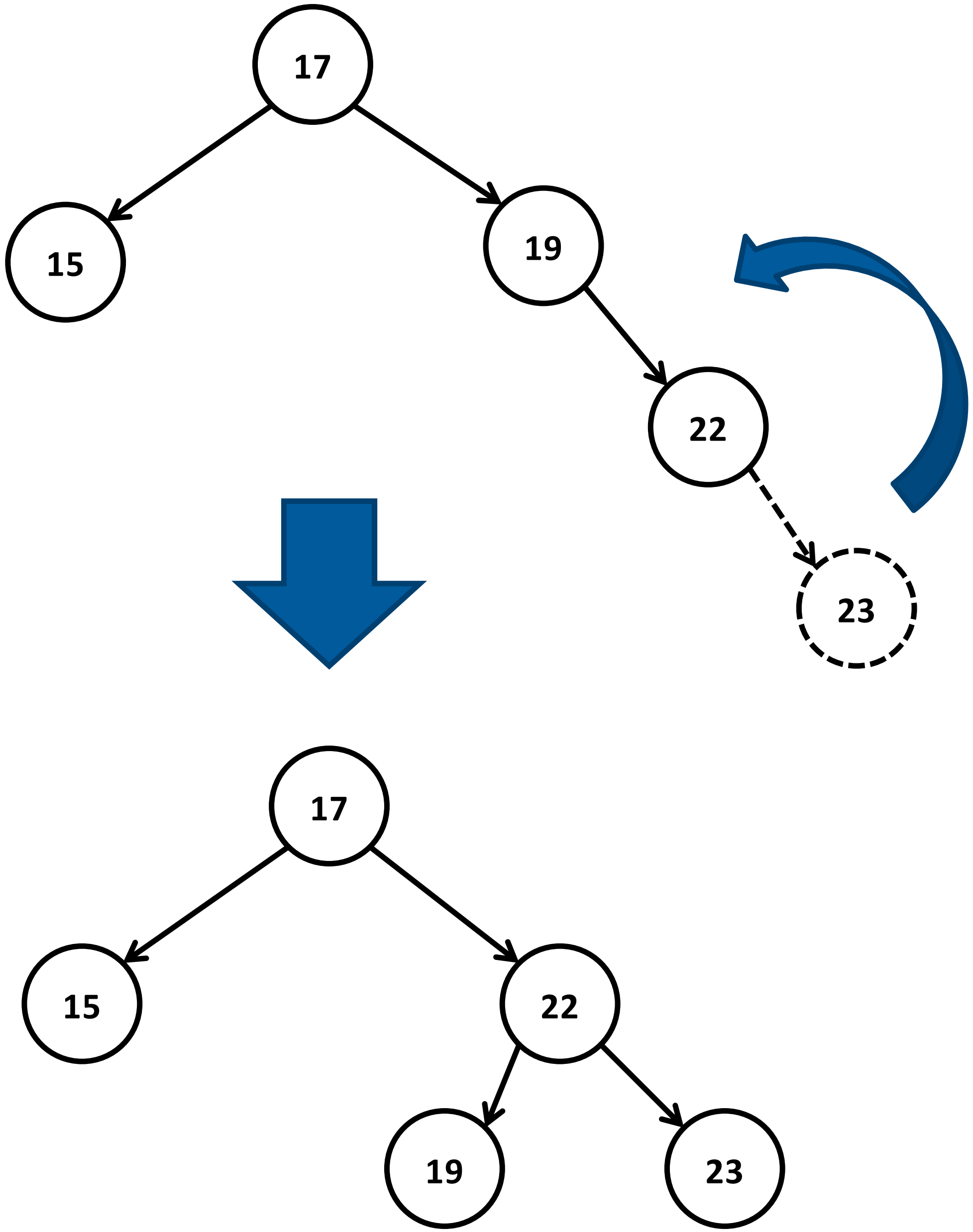


AVL Trees

- After an insertion, only nodes that are on the path from the insertion point to the root might have their balance altered. Number of these nodes is O(log n).
  - Only those nodes have their subtrees altered.
  - Observe that the values of element in left subtrees are less that the values in right ones

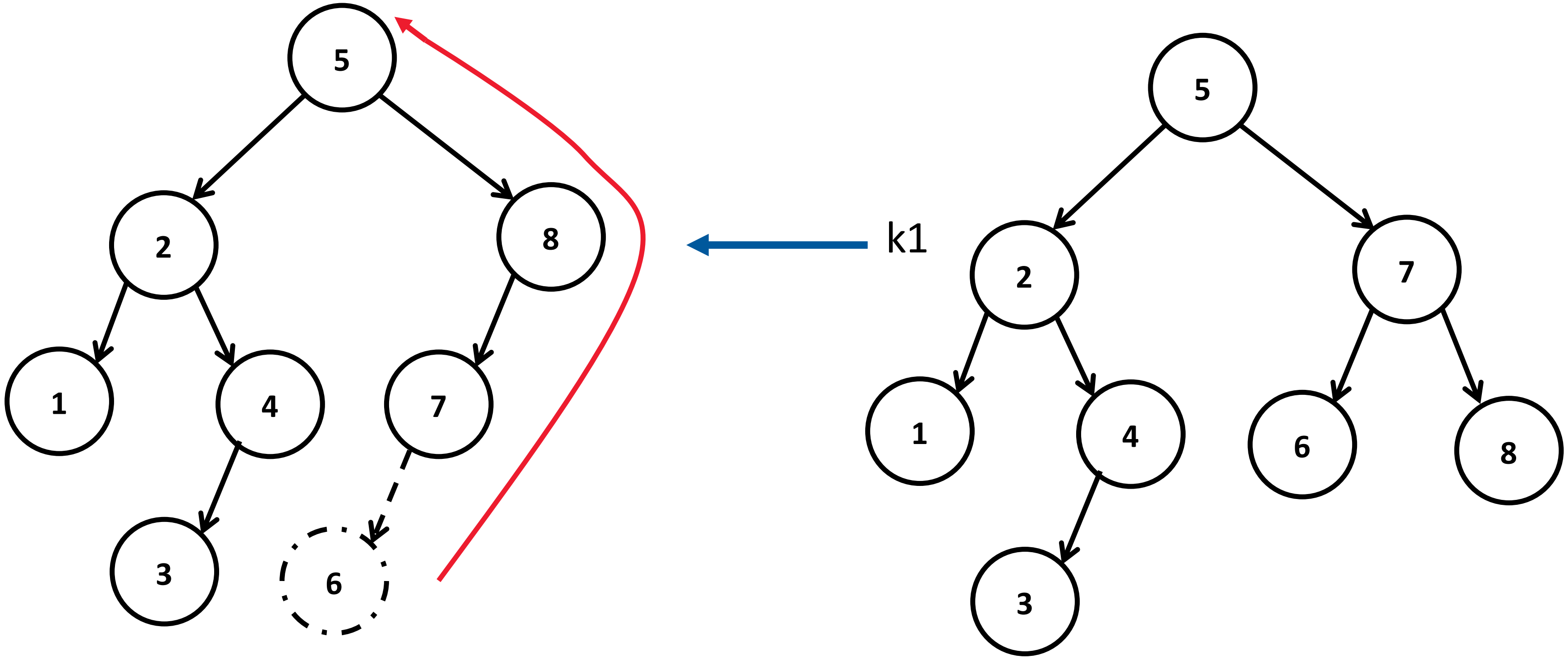


AVL Trees



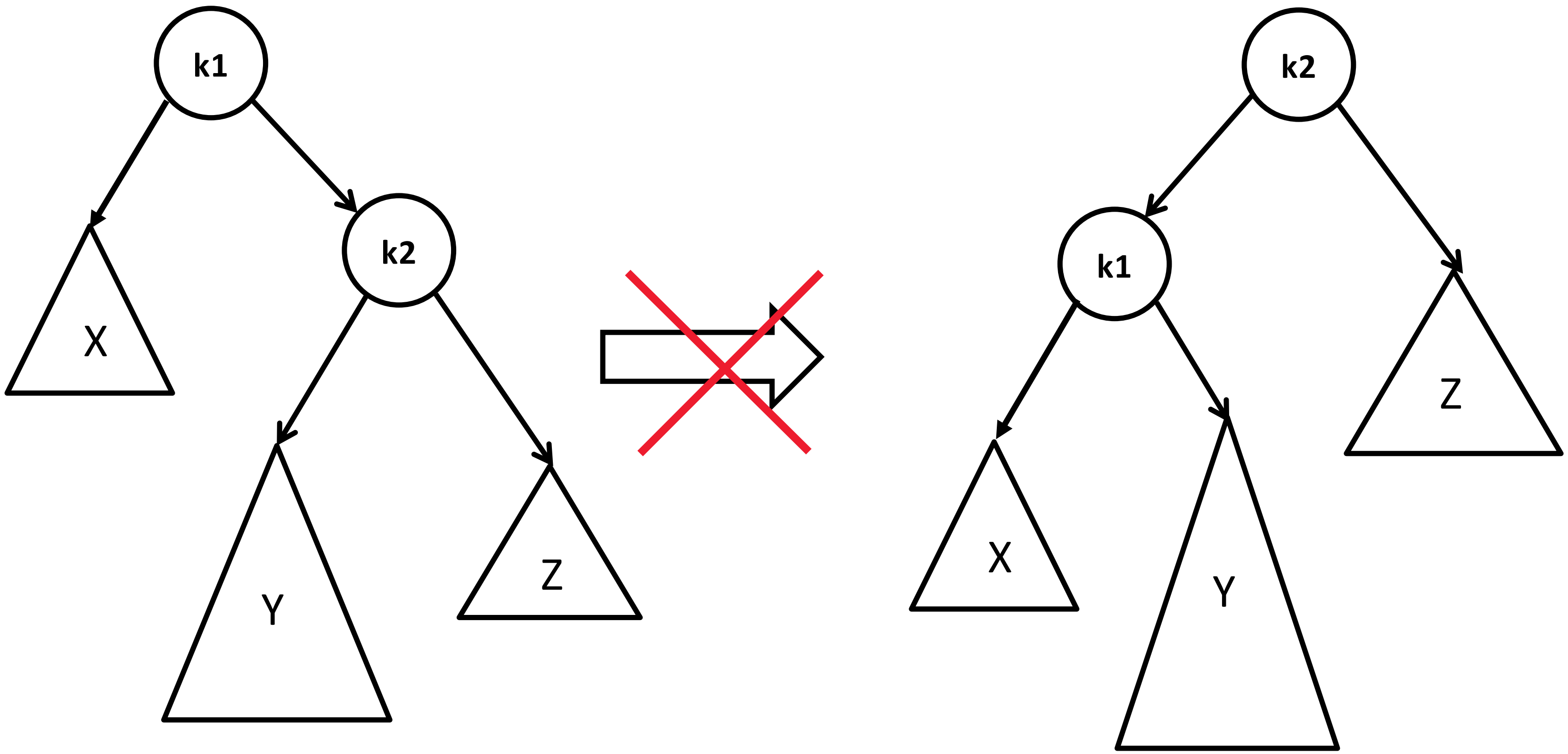
AVL Trees

Insert 6



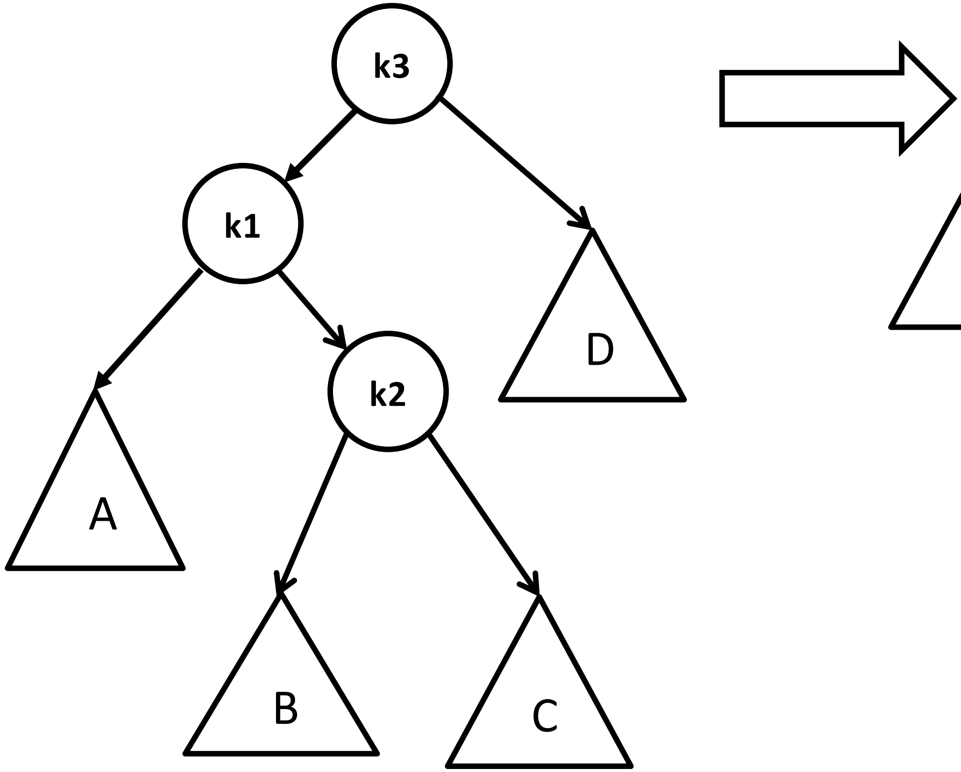
AVL Trees

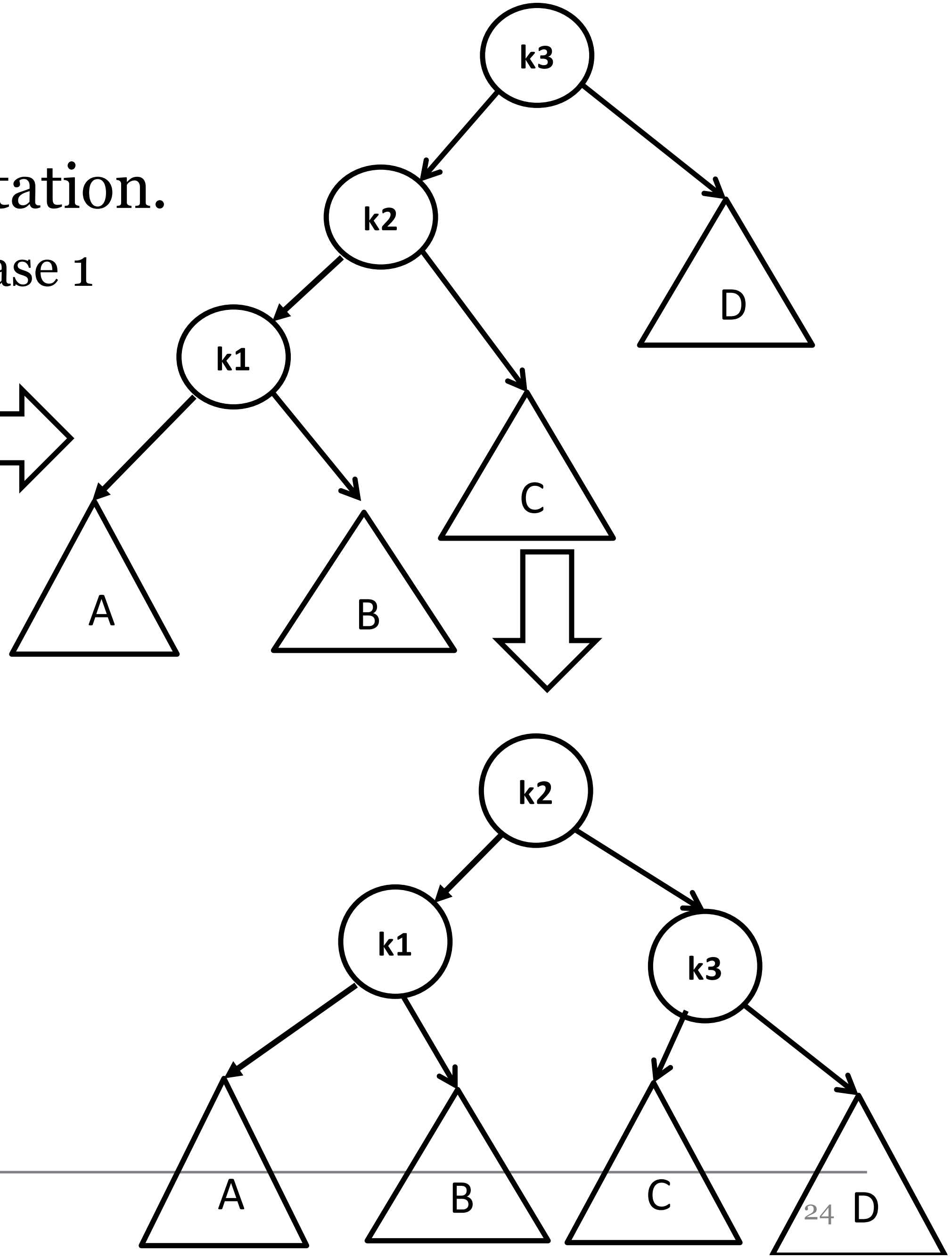
- Case 2: The Y subtree is too deep. A single rotation does not make it less deep.



AVL Trees

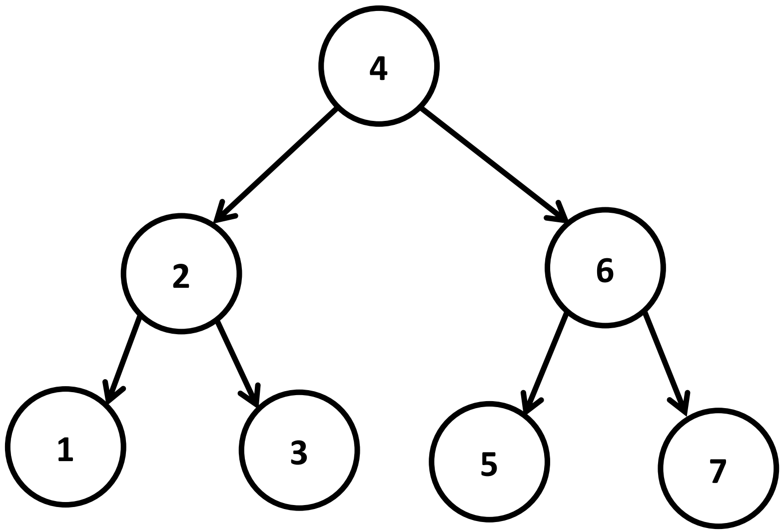
- Case 2: We need double rotation.
  - First rotation makes it like case 1





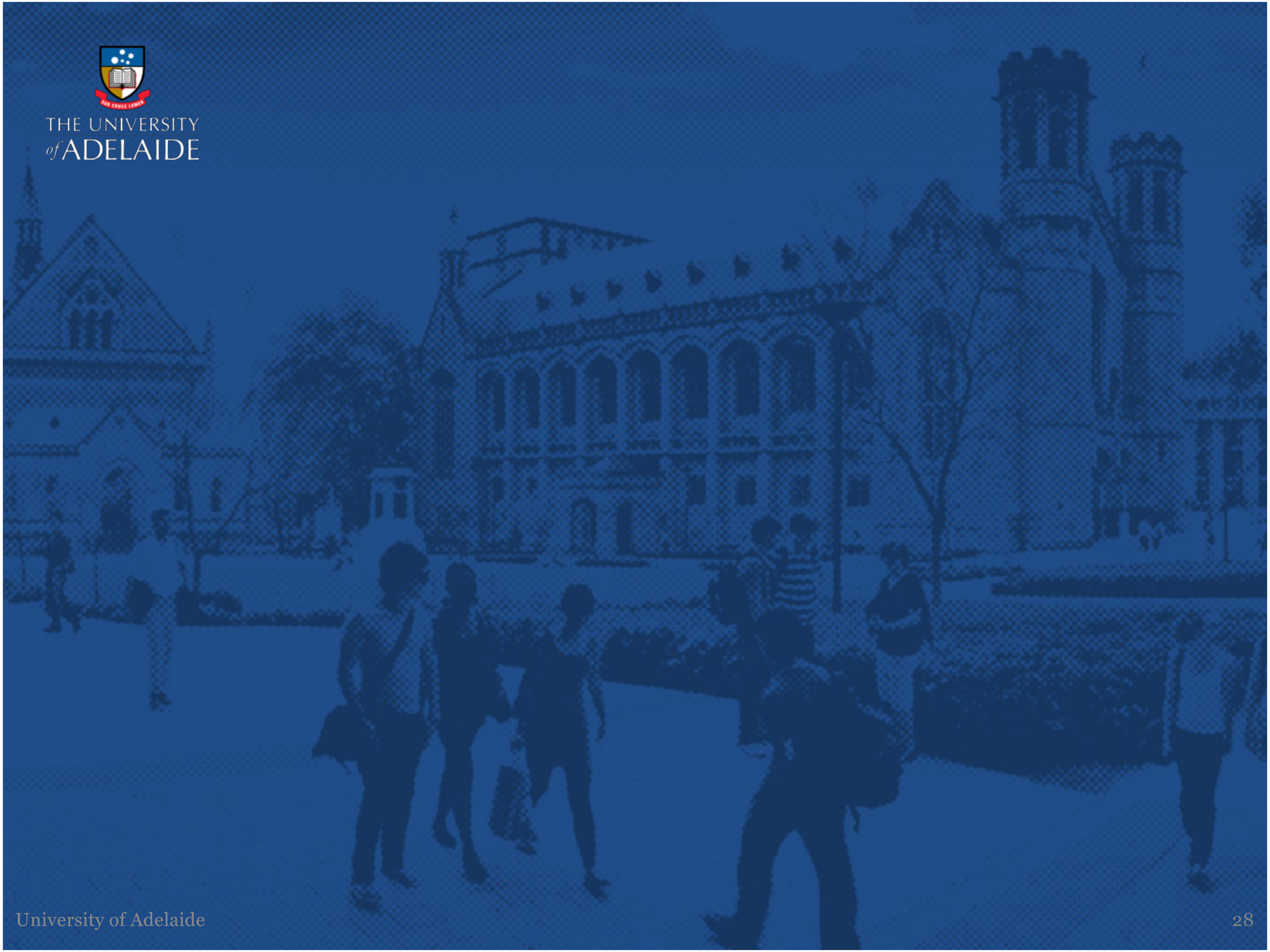
AVL Trees

- Example add 16, 15, 14, 13, 12



AVL Trees

- Assume the node that needs to be rebalanced is A. A violation might occur in four cases:
  - An insertion into the right subtree of the right child of A
  - An insertion into the right subtree of the left child of A
  - An insertion into the left subtree of the right child of A
  - An insertion into the right subtree of the right child of A
- Case 1&4 (2&3) are mirror image symmetries with respect to A.


THE UNIVERSITY
of ADELAIDE