CS 2100: Data Structures & Algorithms 1

Trees

~ Binary Search Trees (Part II) ~

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- Masks are **required** at all times during class (University Policy)
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

Topics

- Finish discussing BST Find and Insert

- BST FindMin/FindMax, Remove, Runtime Analysis

Binary Search Trees

Finalize discussion on BST **Find** and BST **Insert**

Reminder of CompareTo()

Binary Search Trees

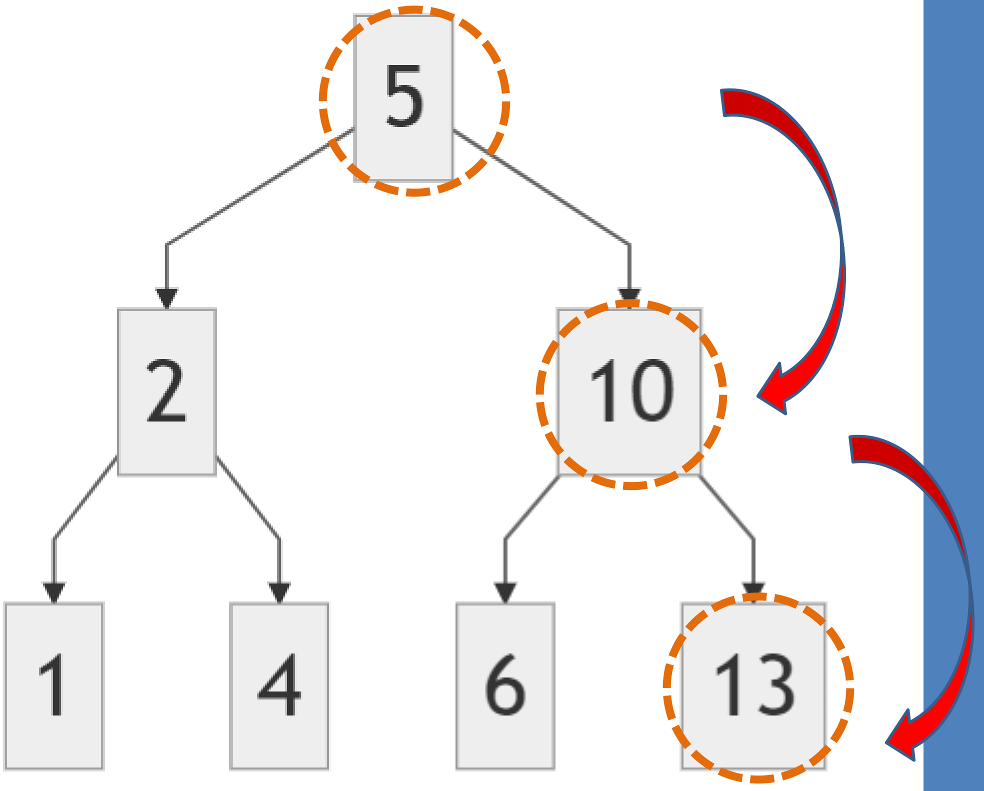
BST **FindMin**; BST **FindMax**; BST **Remove**

Runtime Analysis on BST operations

BST: FindMin() / FindMax()

- Given the way data is stored in a BST, there is a simple way to figure out the smallest **(minimum)** and **largest (maximum)** elements in the data set.

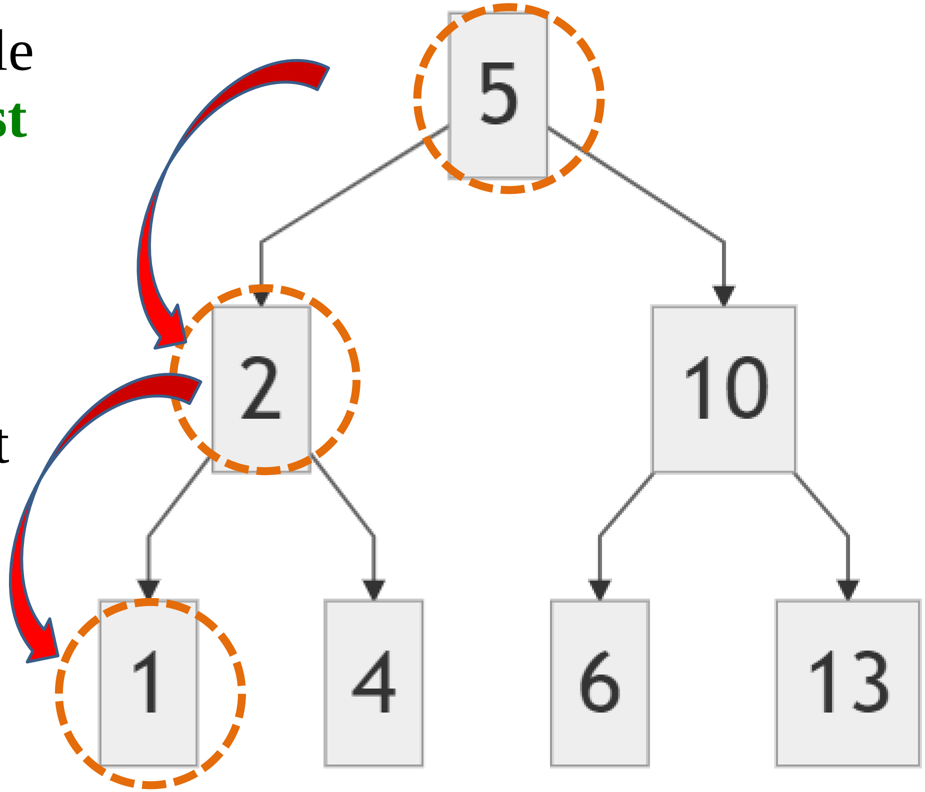
- To find the maximum element, traverse **RIGHT** *until you arrive at a node that has no right-child/subtree.*
  - The data value of **this** node is the maximum element in the BST
  - In this example, 13 is the largest (max) value



BST: FindMin() / FindMax()

- Given the way data is stored in a BST, there is a simple way to figure out the **smallest (minimum)** and **largest (maximum)** elements in the data set.

- To find the minimum element, traverse **LEFT** *until you arrive at a node that has no left-child/subtree.*
  - The data value of **this** node is the minimum element in the BST
  - In this example, 1 is the smallest (min) value



BST: Remove

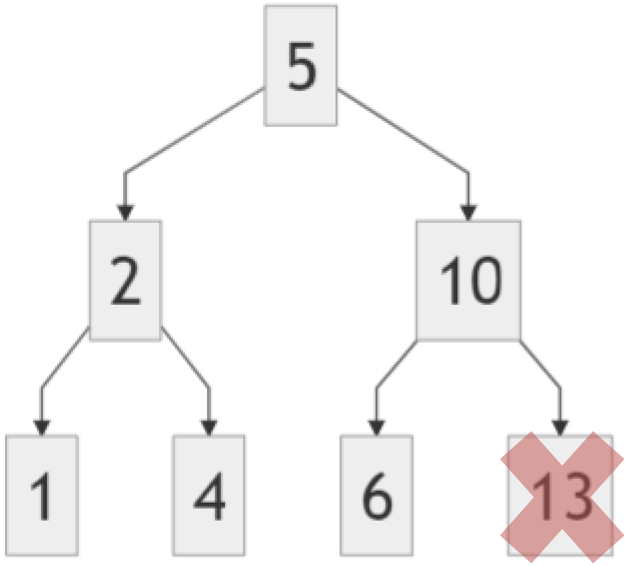
- **Removing** from a BST disrupts the tree structure
  - *Operation is slightly more complicated*

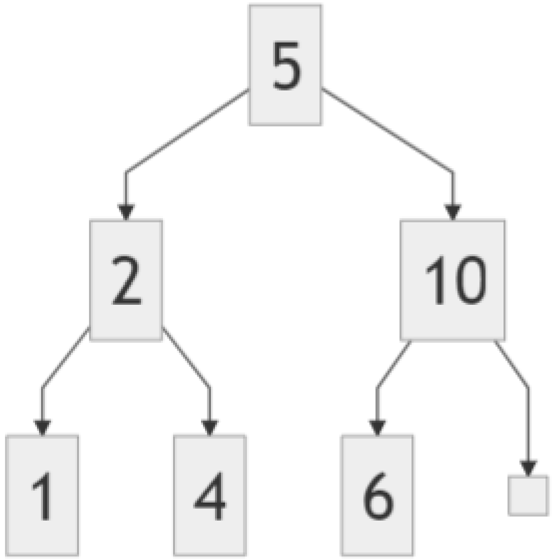
- Basic idea:
  - Find node to be removed
  - THREE CASES:

THREE CASES:	WHAT DO YOU DO?
1. Node has no children *(degree 0)*	delete node
2. Node has one child *(degree 1)*	replace node with its only child
3. Node has two children *(degree 2)*	find the next largest (or smallest) node to replace it – "**Successor Node**"

BST: Remove [Case 1] – Remove(13)

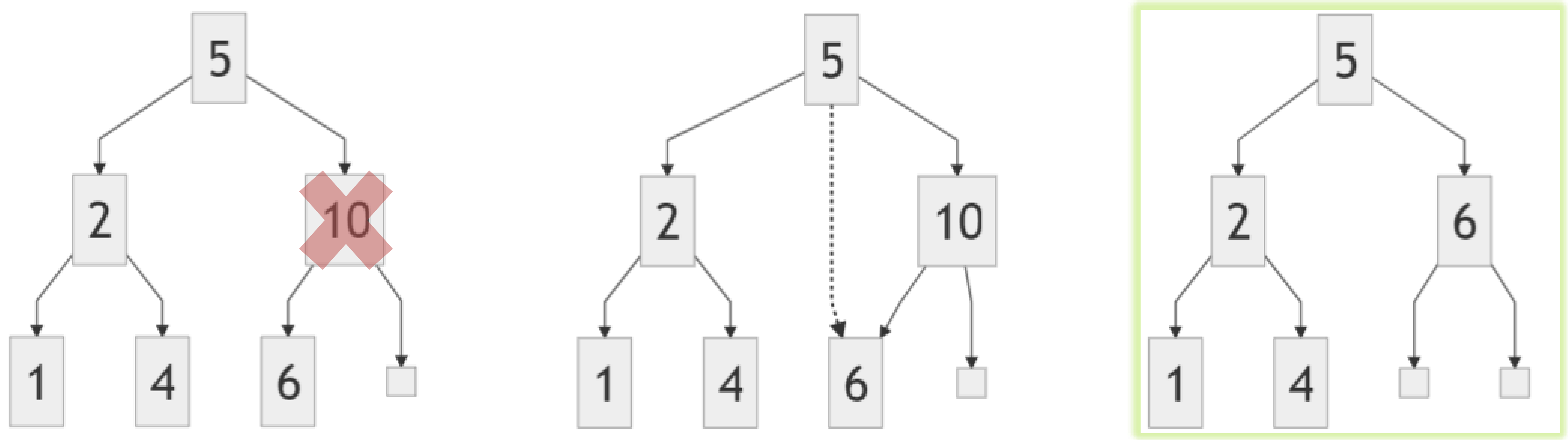
- No children – so just remove the node
  - Make sure parent pointer now points to NULL





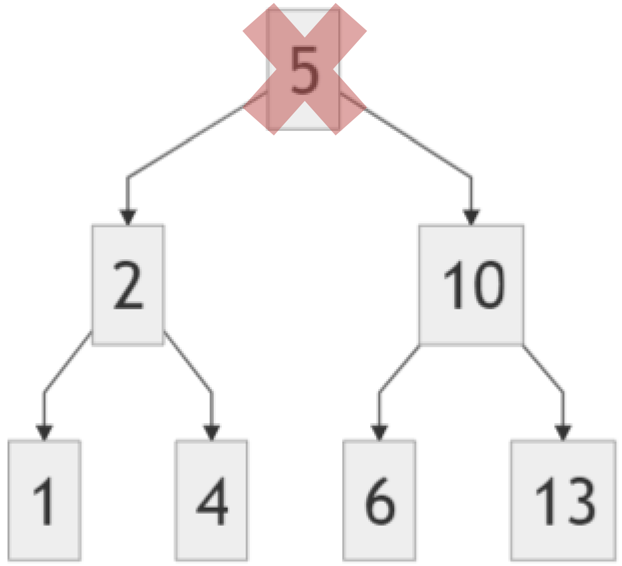
BST: Remove [Case 2] – Remove(10)

- One child – Make parent pointer point to child



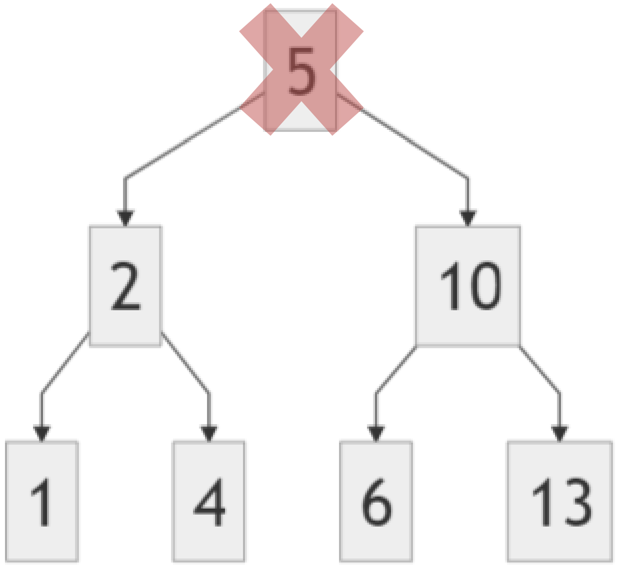
BST: Remove [Case 3] – Remove(5)

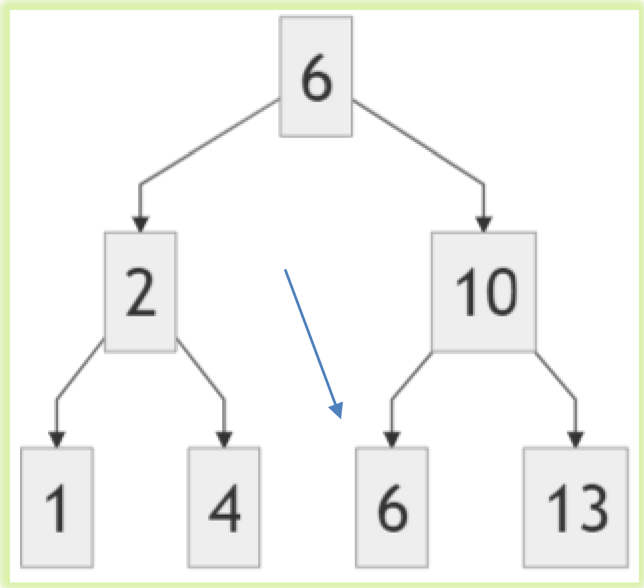
- Two children –
  - **Step 1**: Find successor
    - Next "largest" element
      - Minimum value in right sub-tree: **6**
    - Next "smallest" element
      - Maximum value in left sub-tree: **4**



BST: Remove [Case 3] – Remove(5)

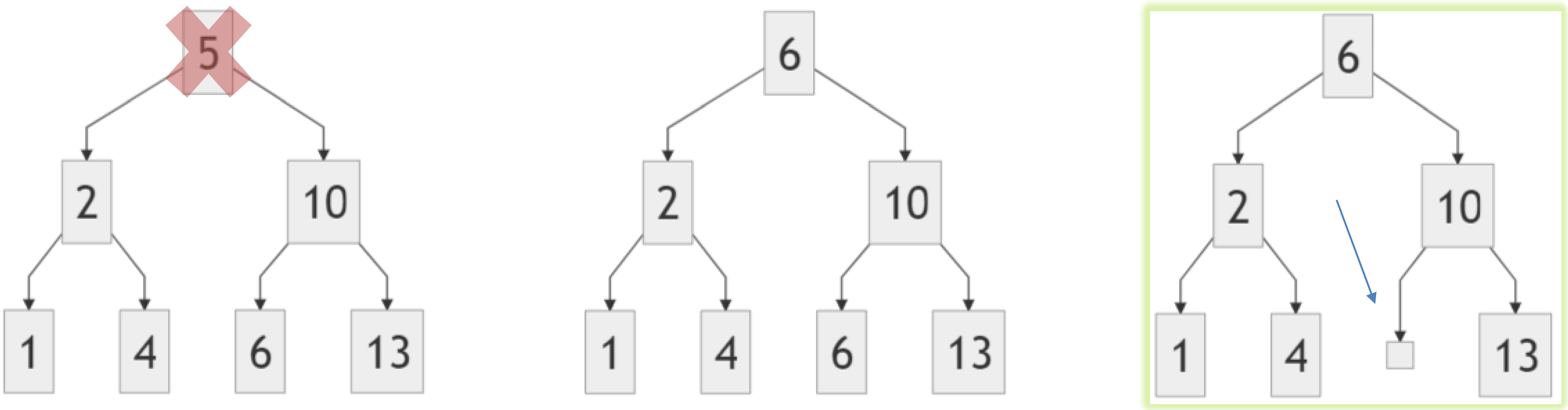
- **Step 2**: Replace deleting node with successor
  - Deleted node (5) overwritten with successor (6)





BST: Remove [Case 3] – Remove(5)

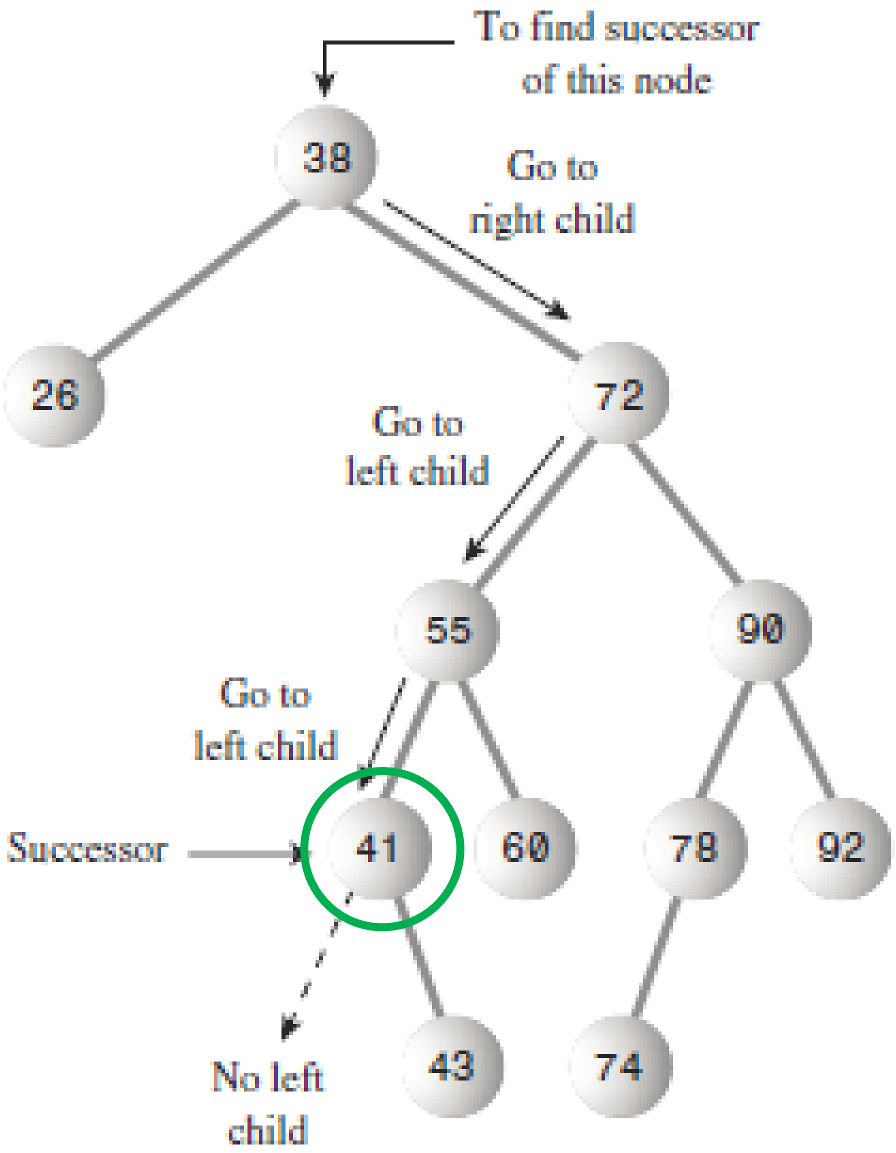
- **Step 3**: Delete successor
  - Recursively call remove(6) – successor will ALWAYS have 0 or 1 child. ***Why?***



Review Successor...

Find Successor of 38

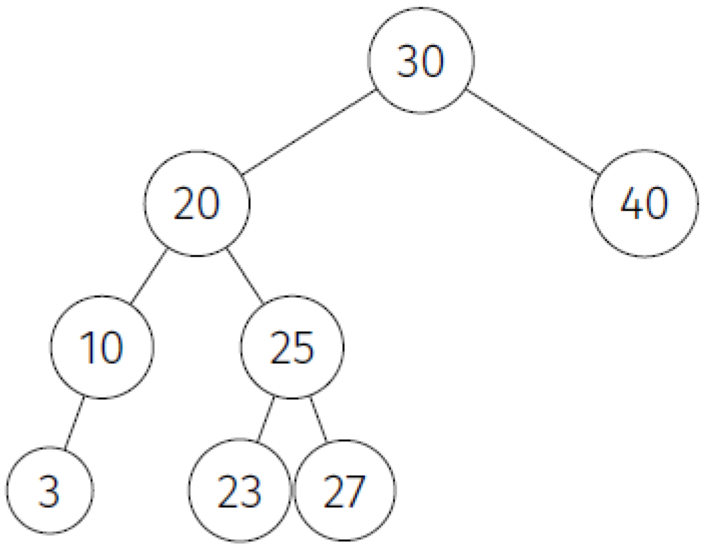
- *Minimum* of **right** subtree (*leftmost node*)
- Which is the next largest number



Finding the successor.

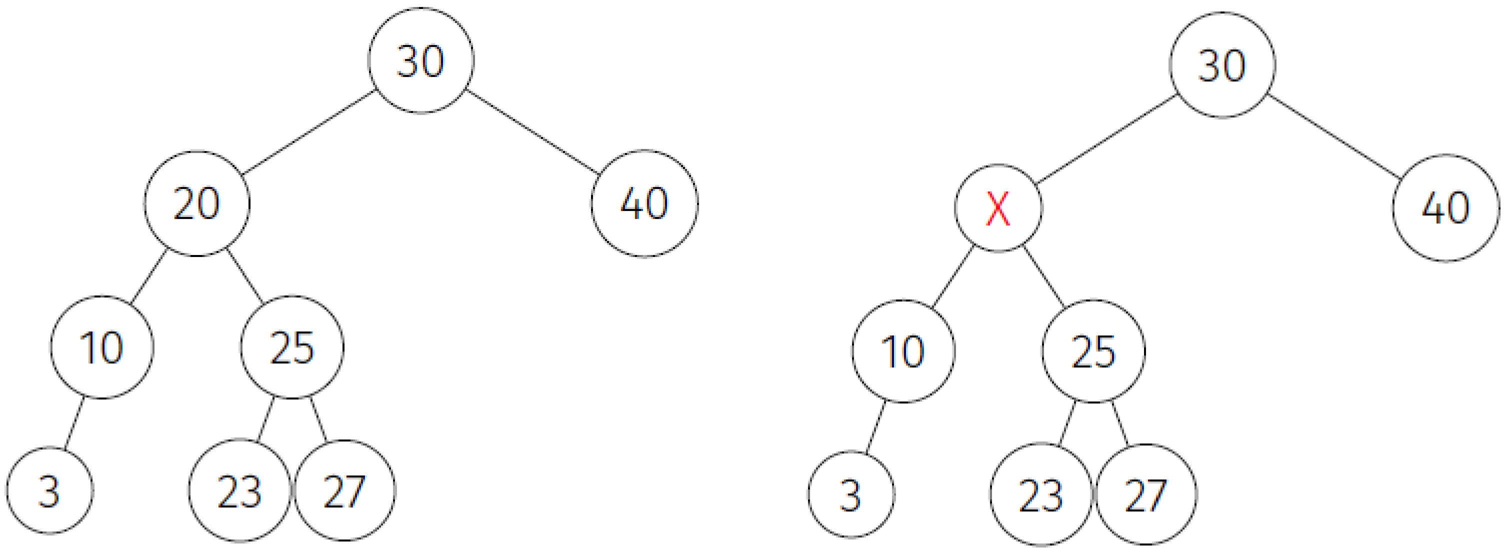
Remove: Another Example

- Delete 20 from the binary search tree



Remove: Another Example

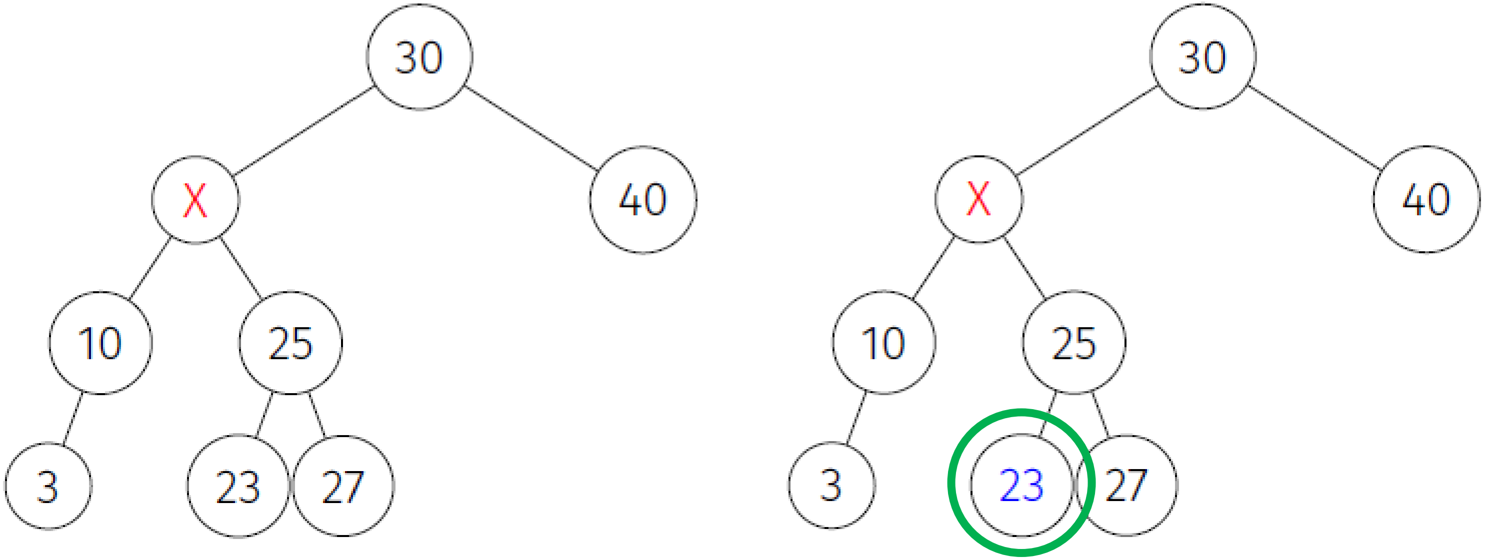
- Delete 20 from the binary search tree



- Need to find a successor for 20: next largest node!

Remove: Another Example

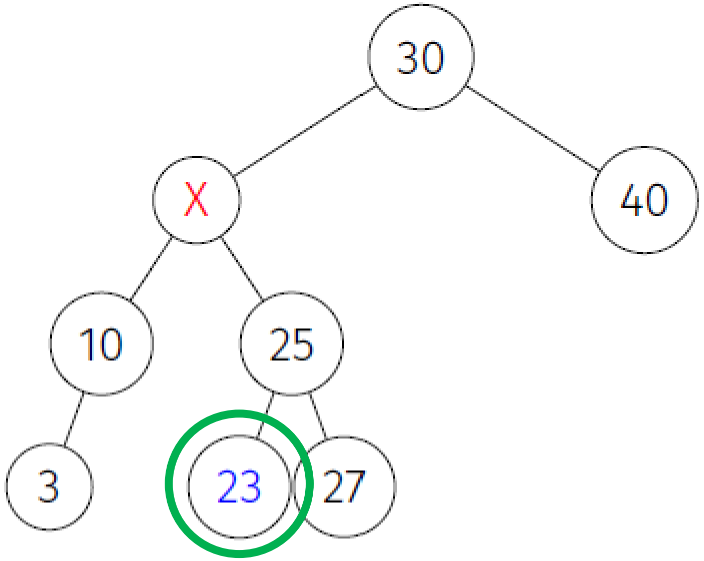
- Delete 20 from the binary search tree

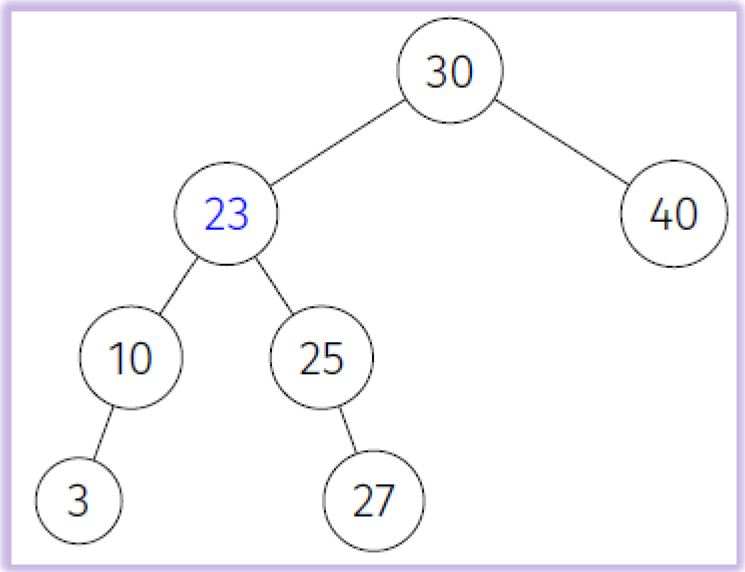


- **Left-most** node of the right subtree

Remove: Another Example

- Delete 20 from the binary search tree

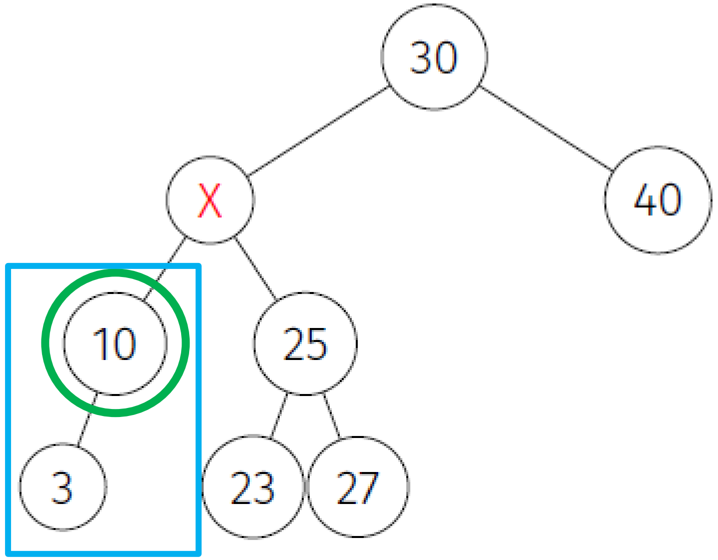


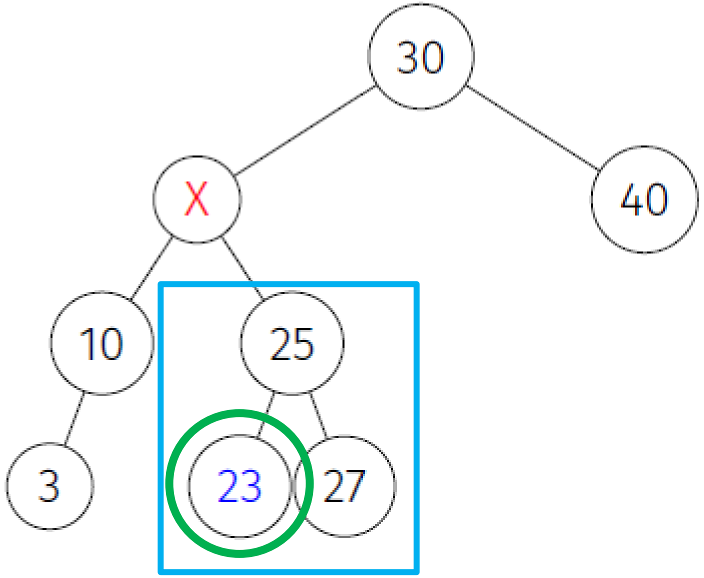


- **Easy-case:** move leaf 23 to replace 20

Successors of ‘X’

- What are the possible successors of ‘X’?

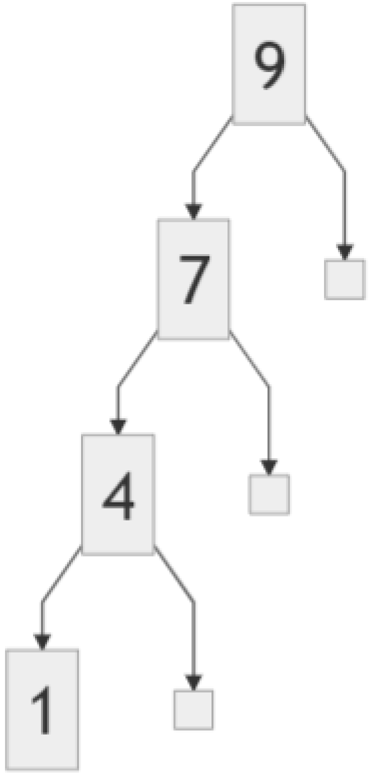




- **Right-most** node of the LEFT subtree → 10
- **Left-most** node of the RIGHT subtree → 23

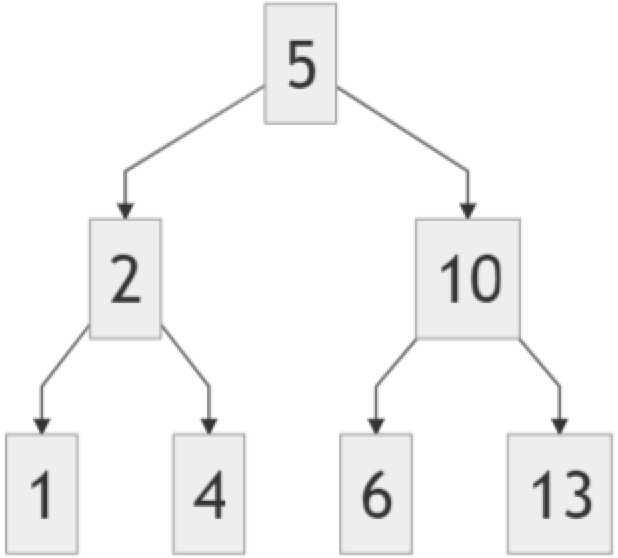
BST: Height

- Worst Case Height: **Linear**. Just a straight line



BST: Height

- Best Case Height: **log(n)** where *n* is num nodes ***Why?***



Perfect Binary Tree

- A “perfect” binary tree has all leaves at same depth
- Every node has 0 or 2 children