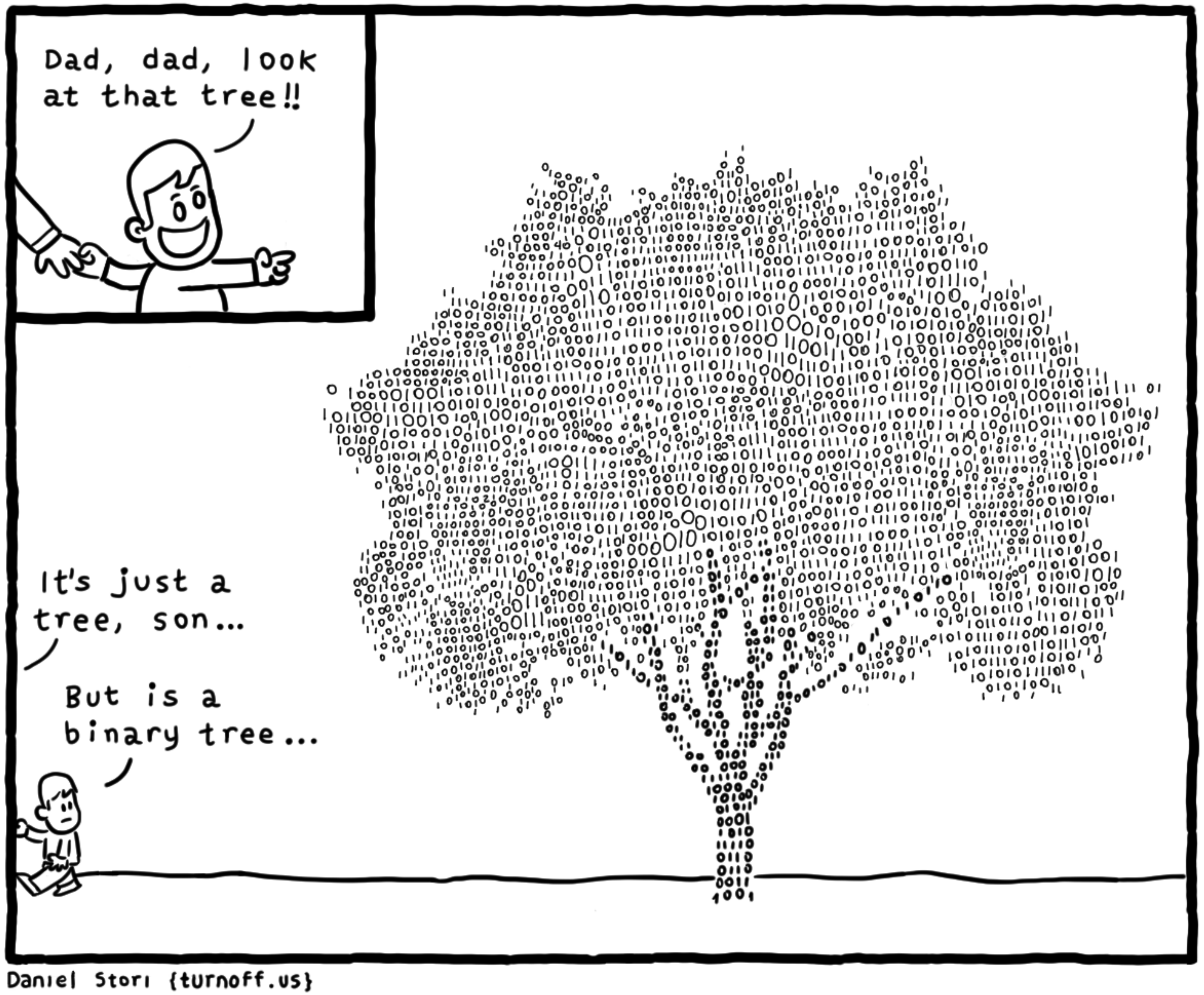
[CSED233-01] Data Structure

Binary Search Tree

Jaesik Park

POSTECH

Daniel Stori {turnoff.us}

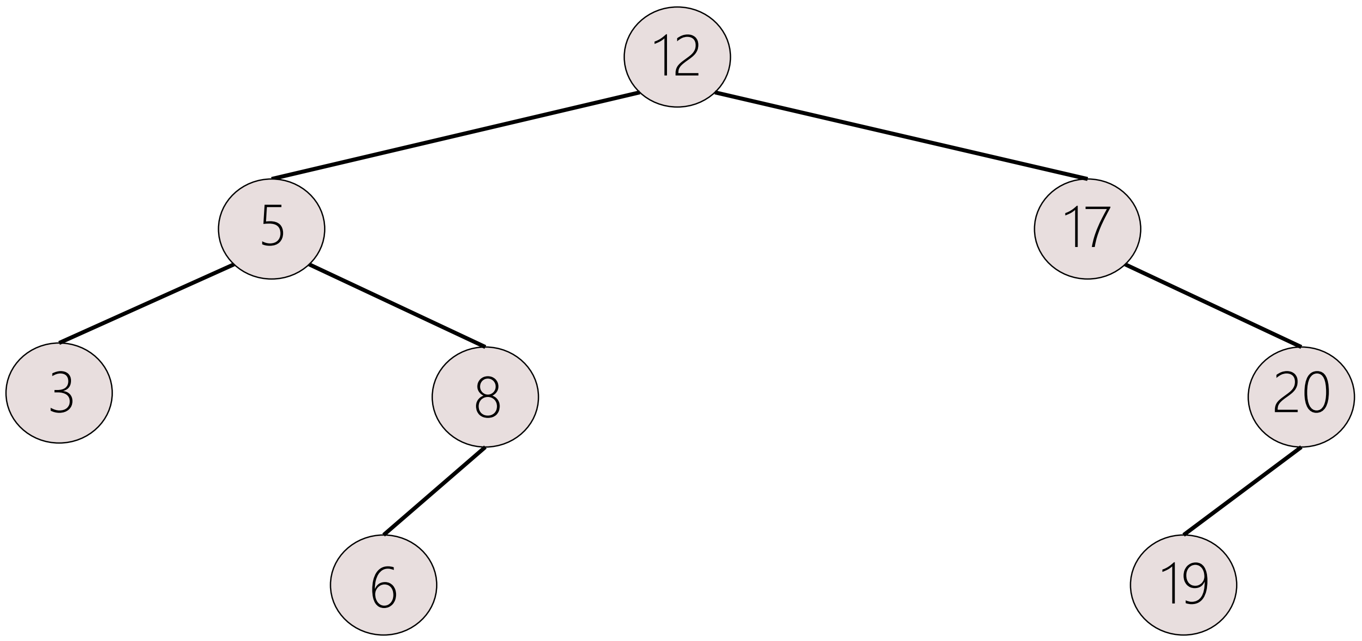

Dad, dad, look at that tree!!
It's just a tree, son...
But is a binary tree...
Daniel Stori {turnoff.us}

Binary Search Tree (BST)

- A binary tree
  - Each node has a (*key*, *value*) pair, where *key* is unique
  - BST property
    - Every node is ordered by key which belongs to a total order
      - For any two non-equal keys X & Y, either $X < Y$ or $X > Y$
    - The key of any node is greater than all keys stored in its left subtree & less than all keys in its right subtree

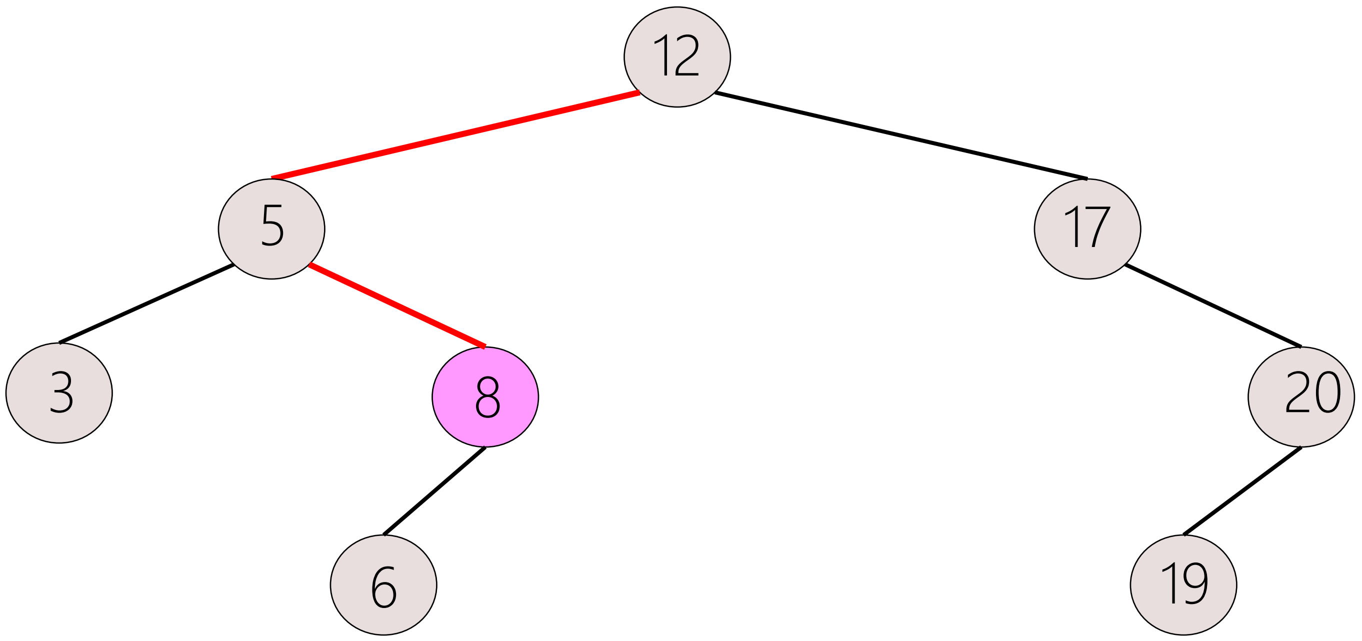
- Also known as ordered/sorted binary tree

Binary Search Tree (BST): Example



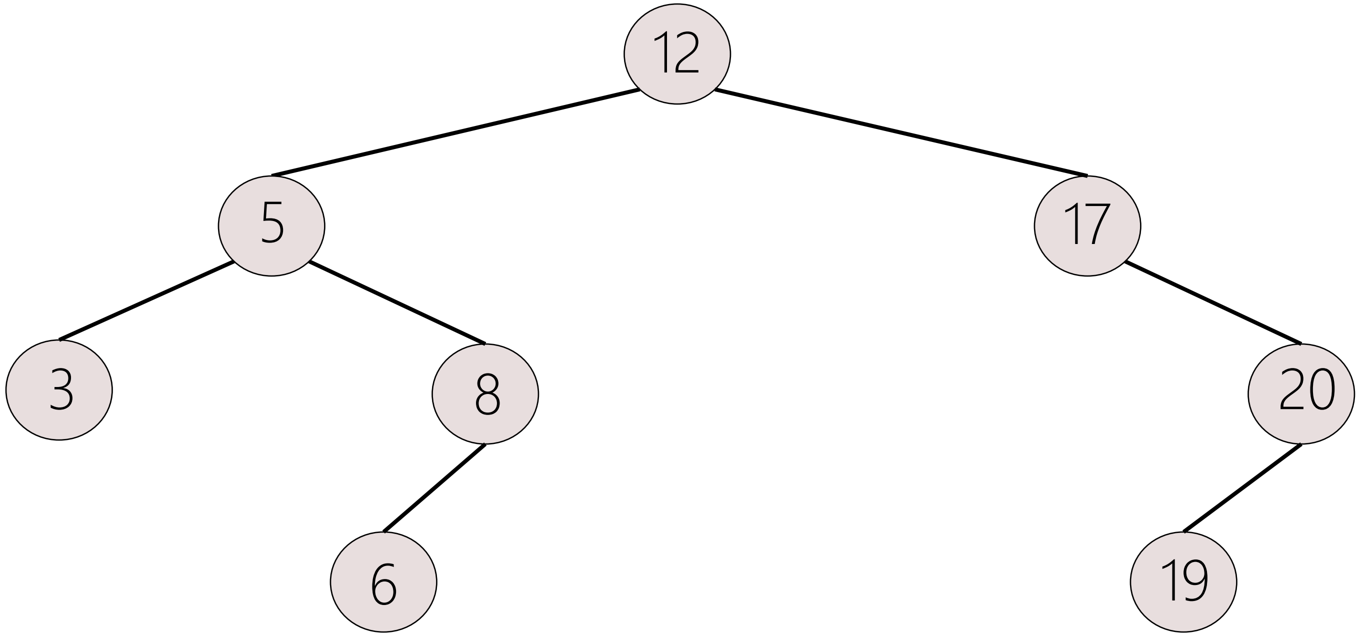
- Only keys (not priority values) are shown
- *Search*(8, *D*)

BST: Search(x, D)



- After *Search*(8, *D*)
- Time complexity = O(*height*)
    - O(log n) if BST is balanced
    - O(n) if BST is a linear list (worst case)

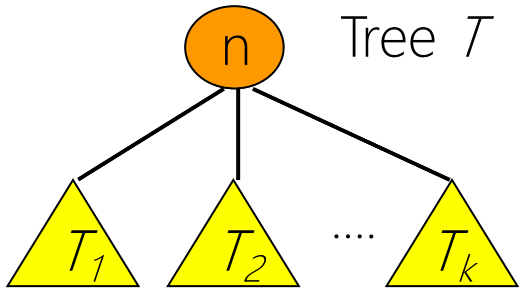
BST: Sort(*D*) in Ascending Order



- How to sort the keys in an ascending order?
  - Do an *in-order* traversal
- Time complexity = O(n)

Review: Tree Traversal

- Types of traversals



- $Preorder(T) = \langle {\color{red}n},\ Preorder(T_1), \ldots, Preorder(T_k)\rangle$
- $Postorder(T) = \langle Postorder(T_1), \ldots, Postorder(T_k),\ {\color{red}n}\rangle$
- $Inorder(T) = \langle Inorder(T_1),\ {\color{red}n},\ Inorder(T_2), \ldots, Inorder(T_k)\rangle$
  - No natural definition of *Inorder* (except for binary tree)

Tree Traversals: Examples

- *Preorder*(*T*)

  = ?

  1, 2, 4, 5, 3, 6, 8, 7

- *Postorder*(*T*)

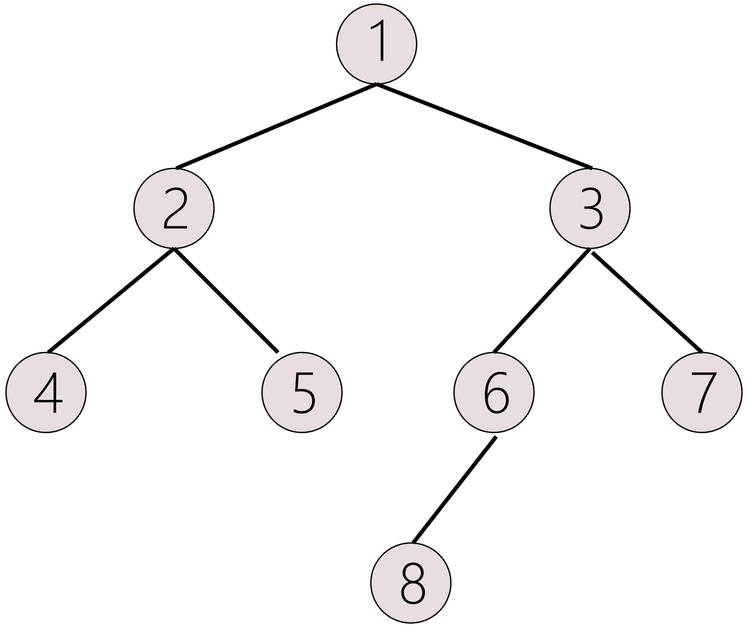
  = ?

  4, 5, 2, 8, 6, 7, 3, 1

- *Inorder*(*T*)

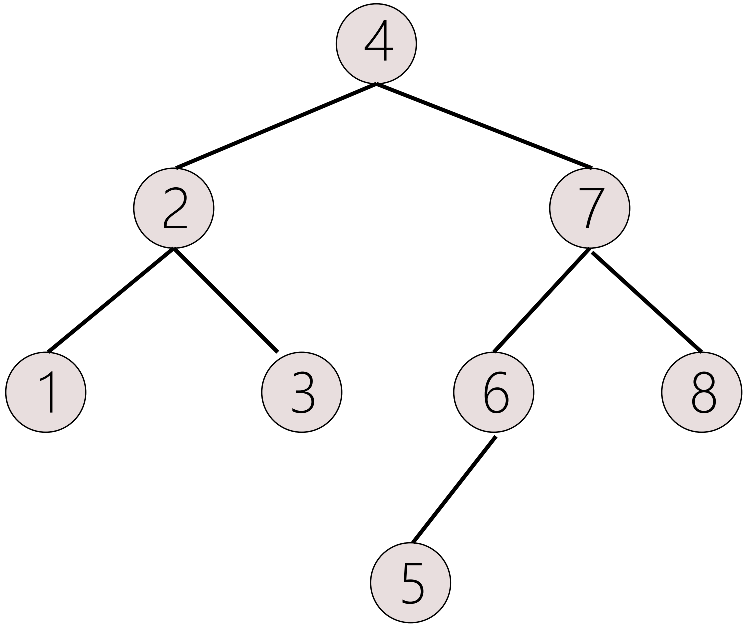
  4, 2, 5, 1, 8, 6, 3, 7

  = ?

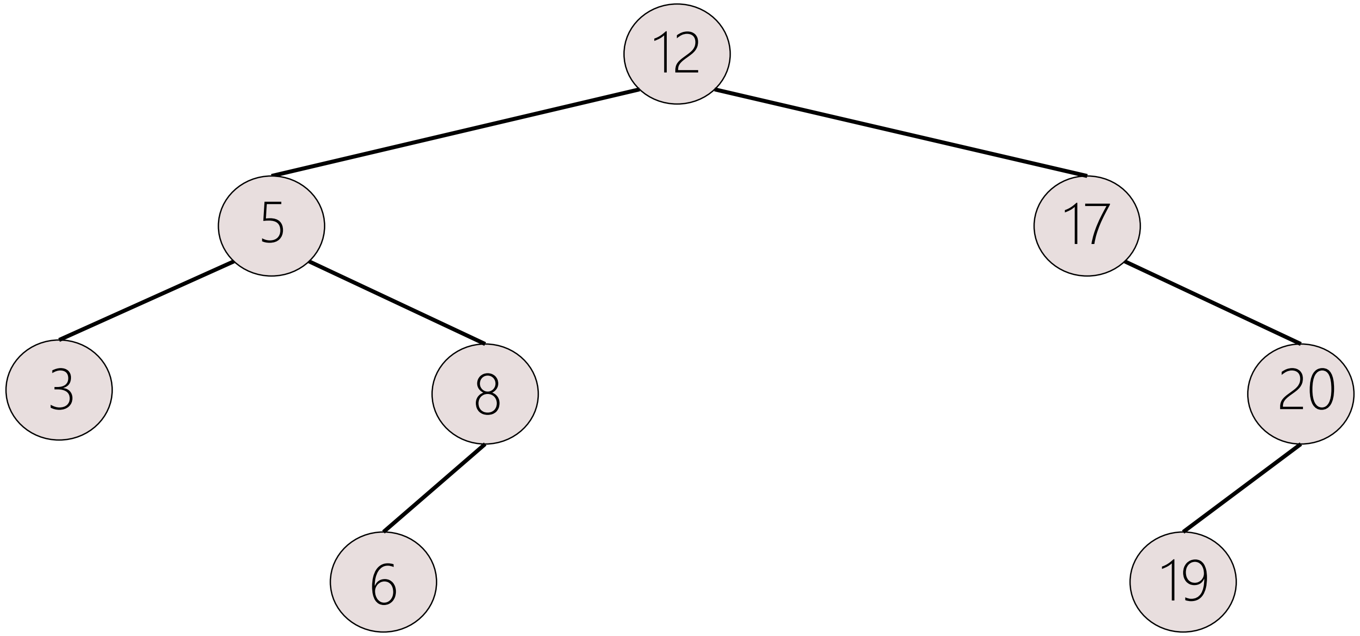


Inorder Tree Traversal: Binary Search Tree

- *Inorder*(*T*)
  = 1, 2, 3, 4, 5, 6, 7, 8

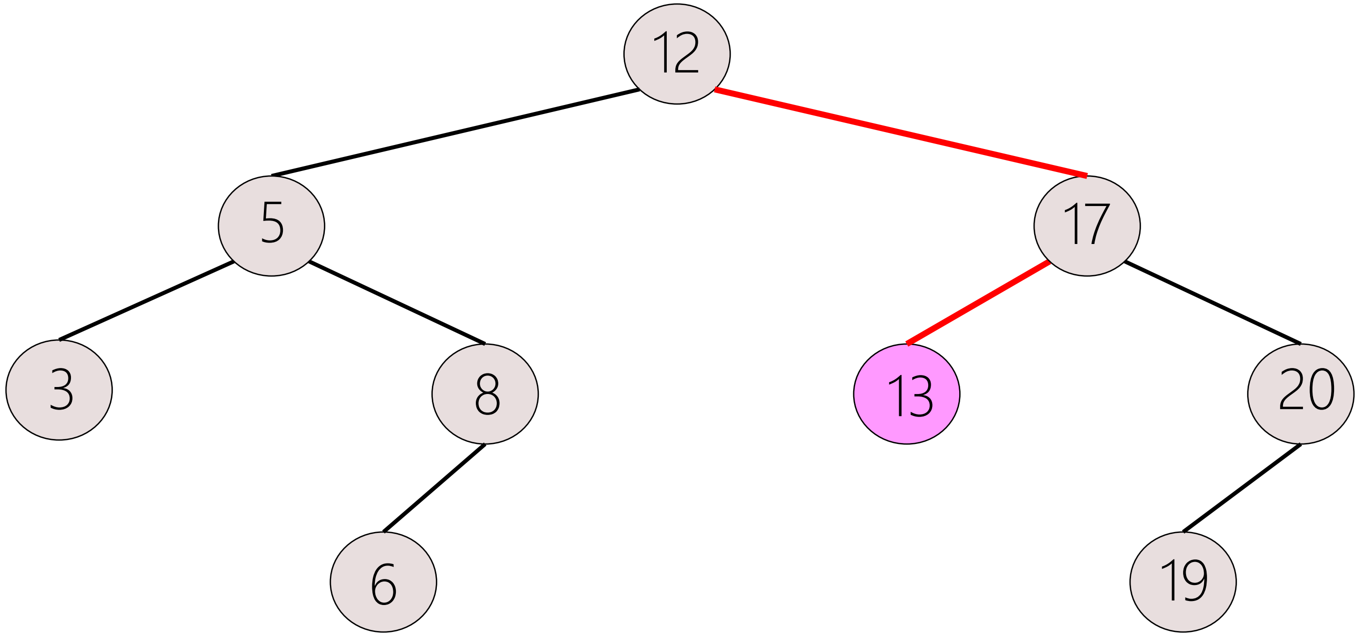


BST: Insert(x, D)



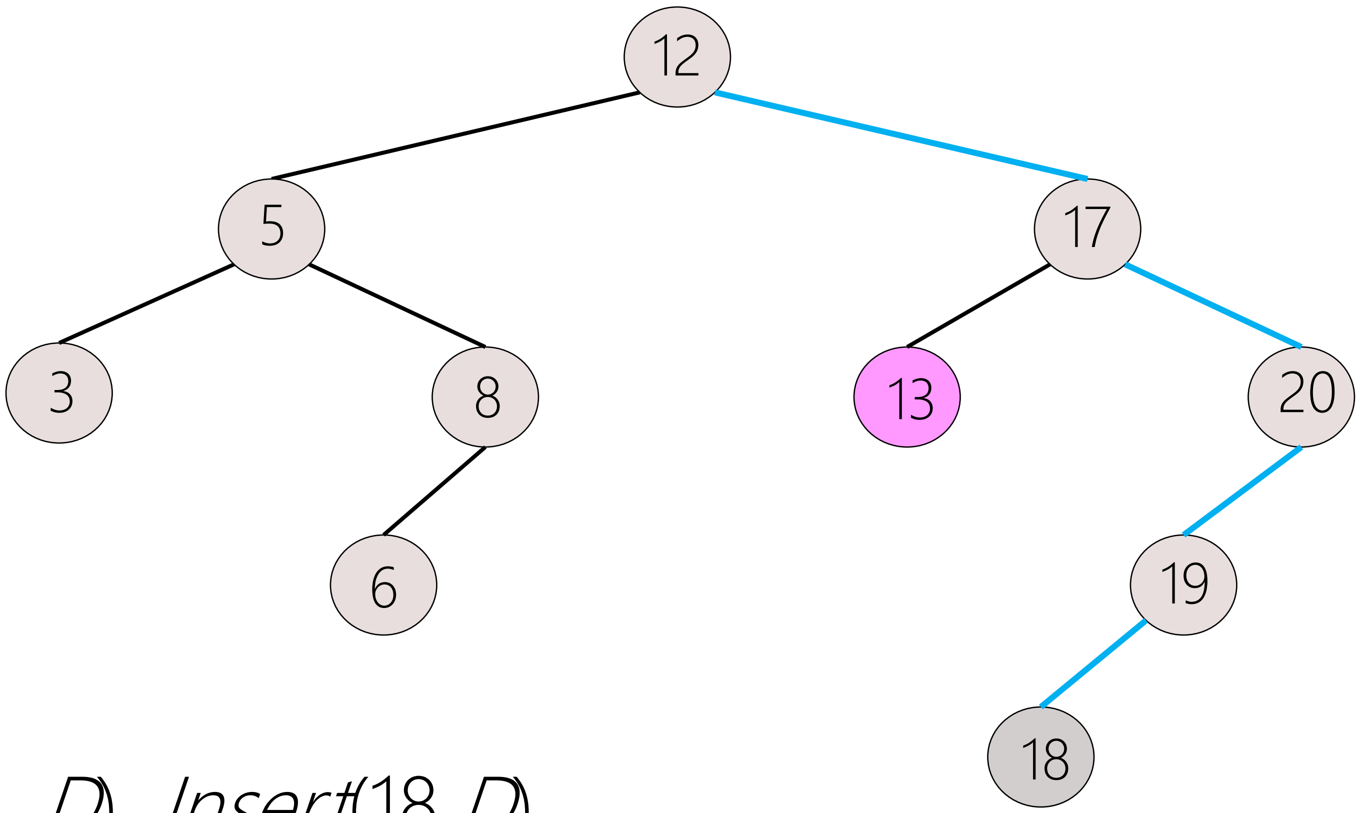
- *Insert*(13, *D*)

BST: Insert(x, D)



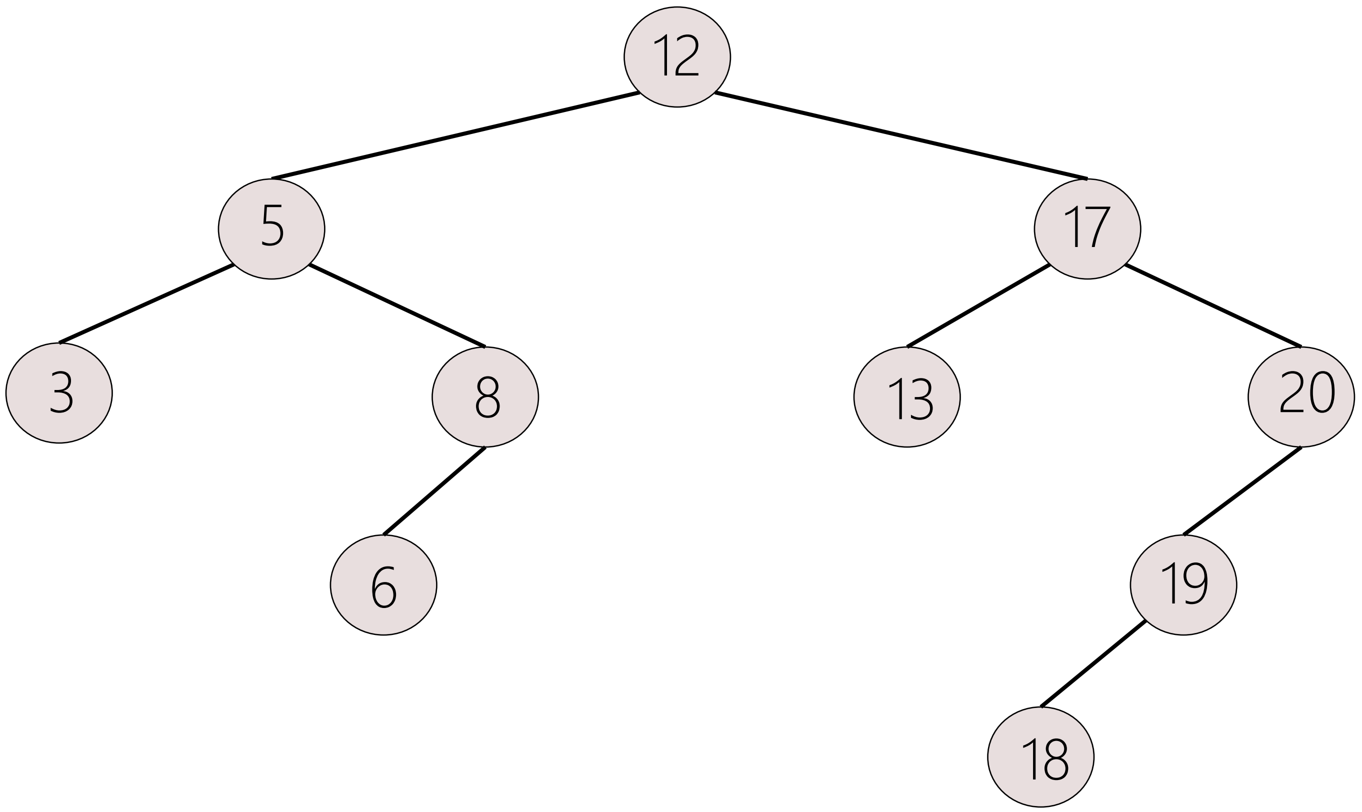
- After *Insert*(13, D)
- *Insert*(18, D)

BST: Insert(x, D)



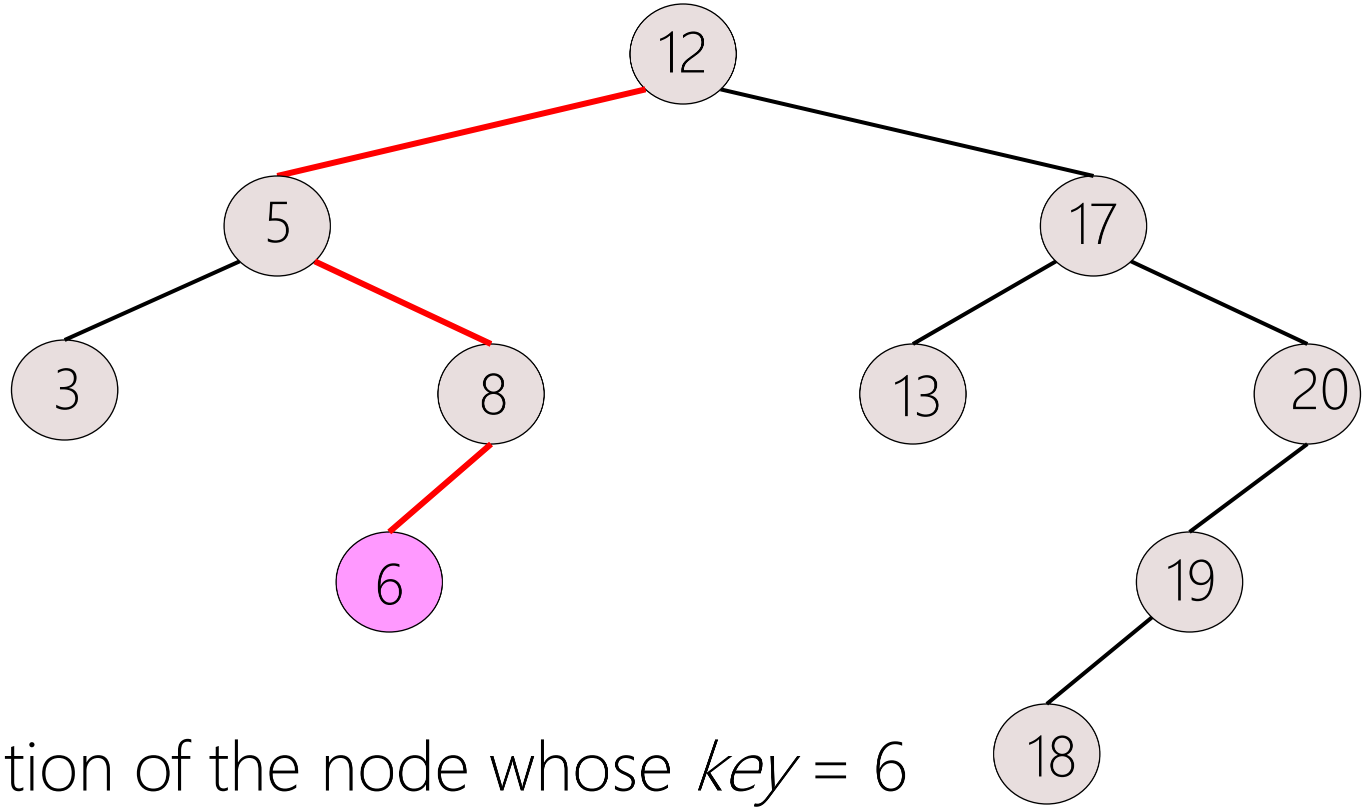
- After *Insert*(13, *D*), *Insert*(18, *D*)
- Time complexity = ?

BST: Delete(x, D) – From a Leaf Node



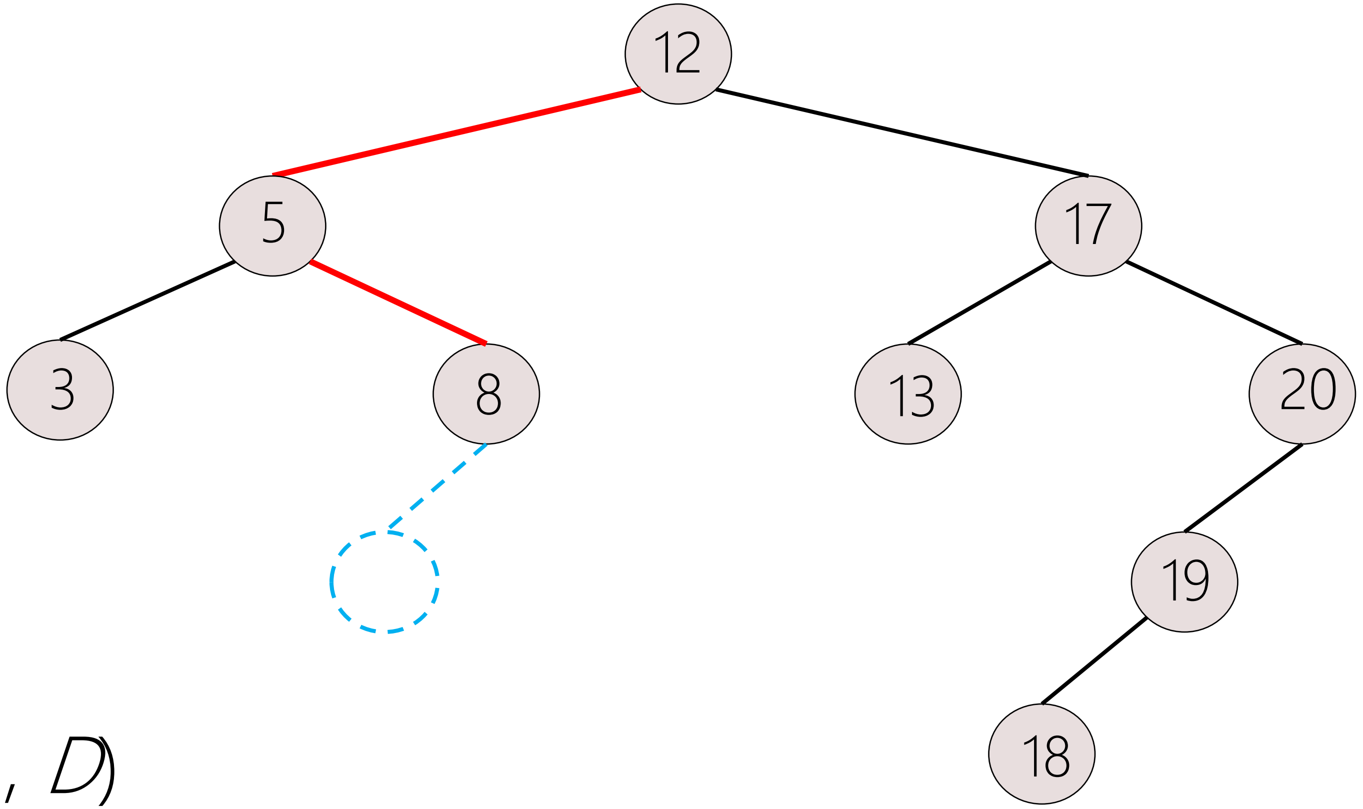
- *Delete*(6, *D*)

BST: Delete(x, D) – From a Leaf Node



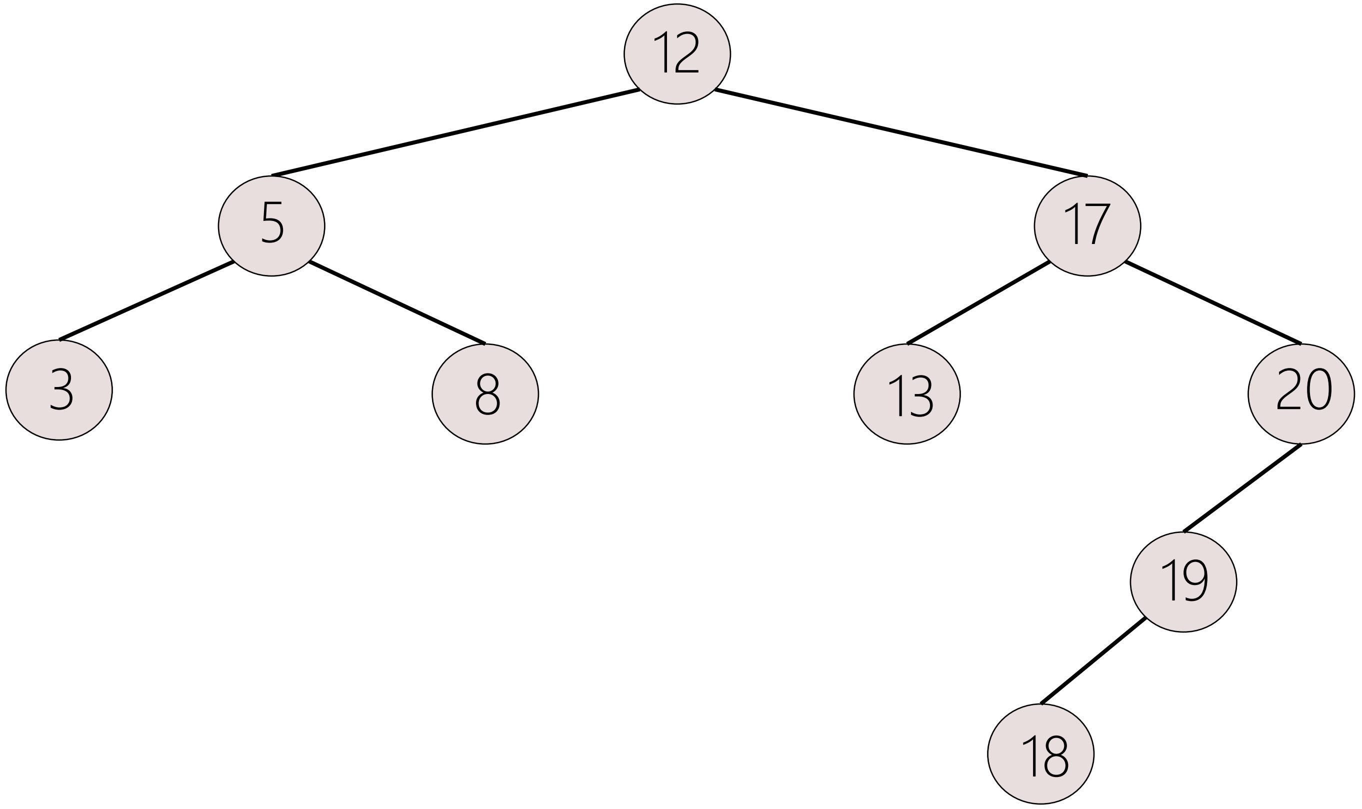
- *Delete*(6, *D*)
  - Find the location of the node whose *key* = 6
- Easy case: no children

BST: Delete(x, D) – From a Leaf Node



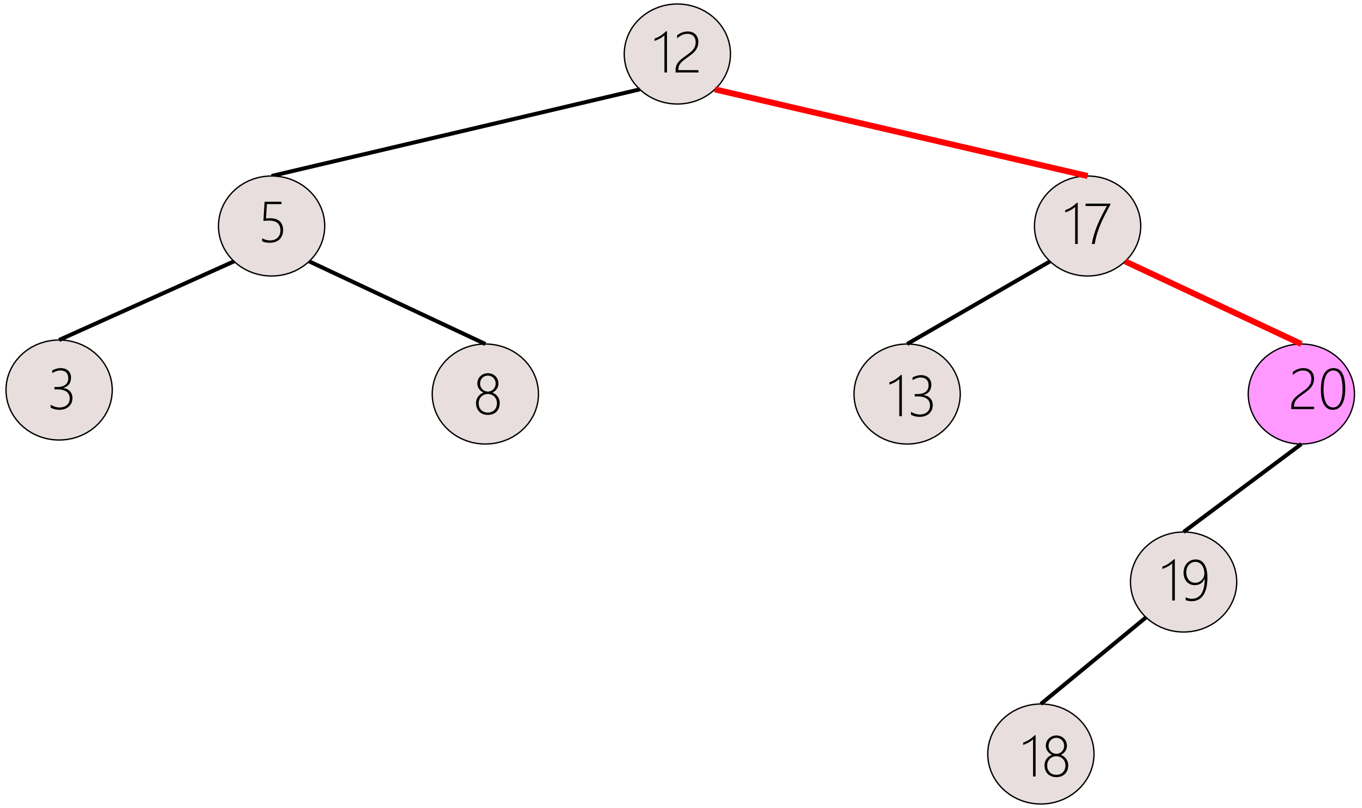
- After *Delete*(6, *D*)

BST: Delete(x, D) – From a Degree 1 Node



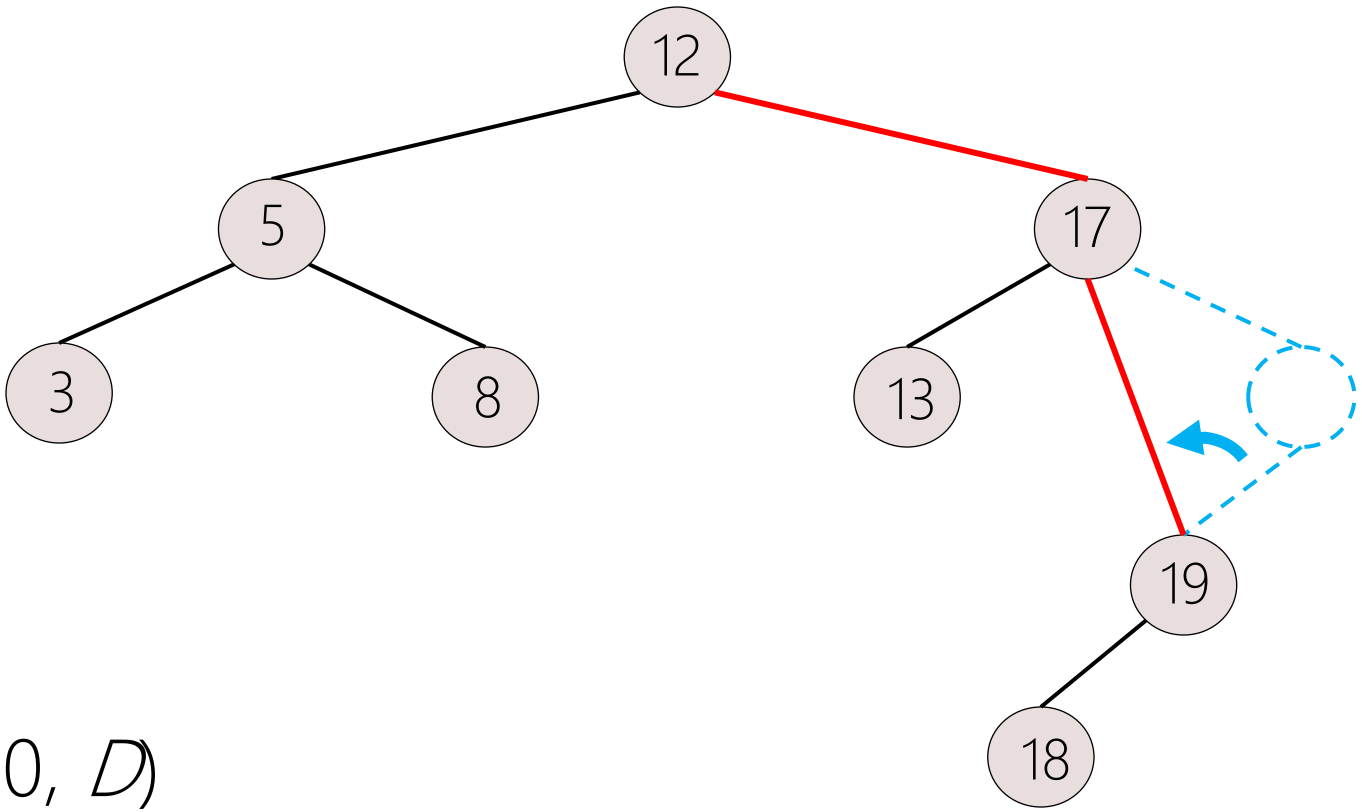
- *Delete*(20, D)

BST: Delete(x, D) – From a Degree 1 Node



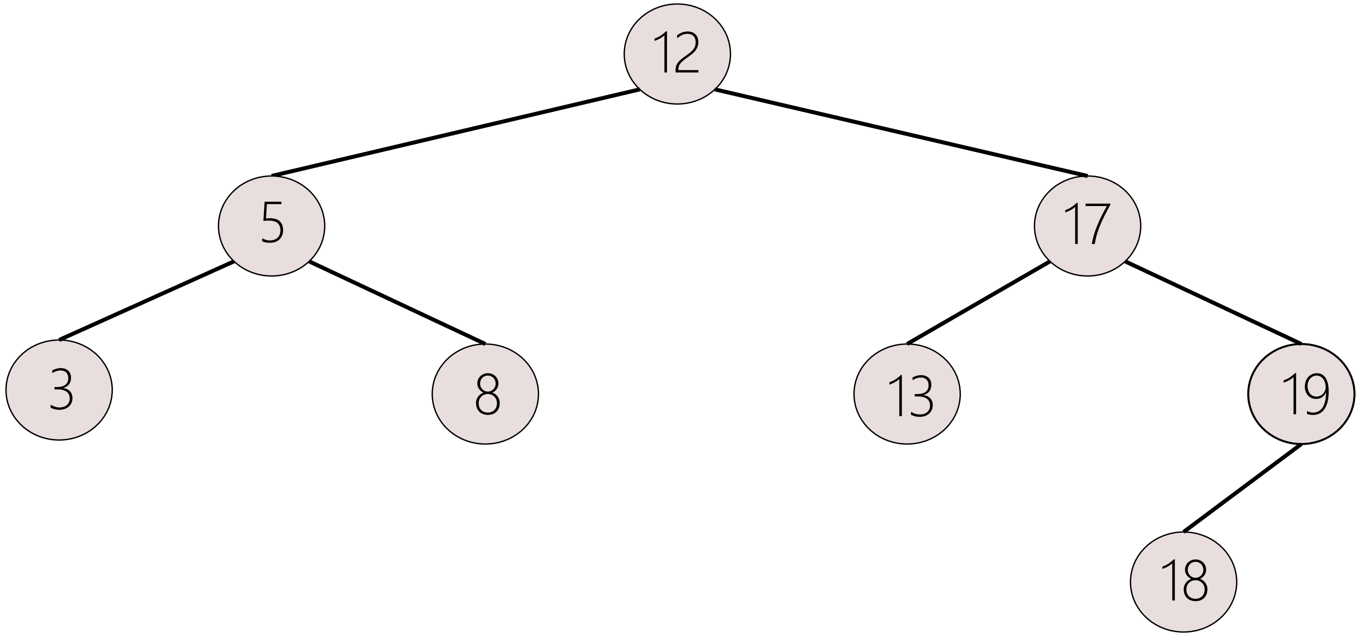
- *Delete*(20, *D*)
  - Find the location of the node whose *key* = 20
- Almost as easy as the leaf case: only one child

BST: Delete(x, D) – From a Degree 1 Node



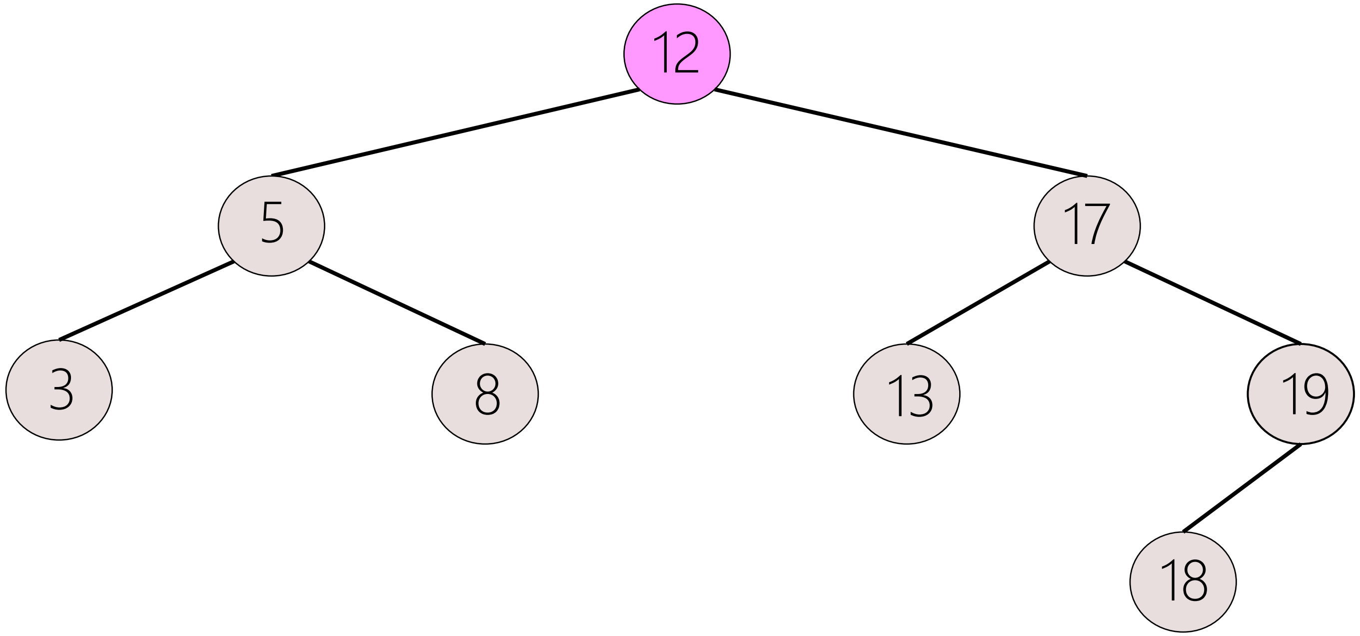
- After *Delete*(20, *D*)

BST: Delete(x, D) – From a Deg. 2 Node (1)



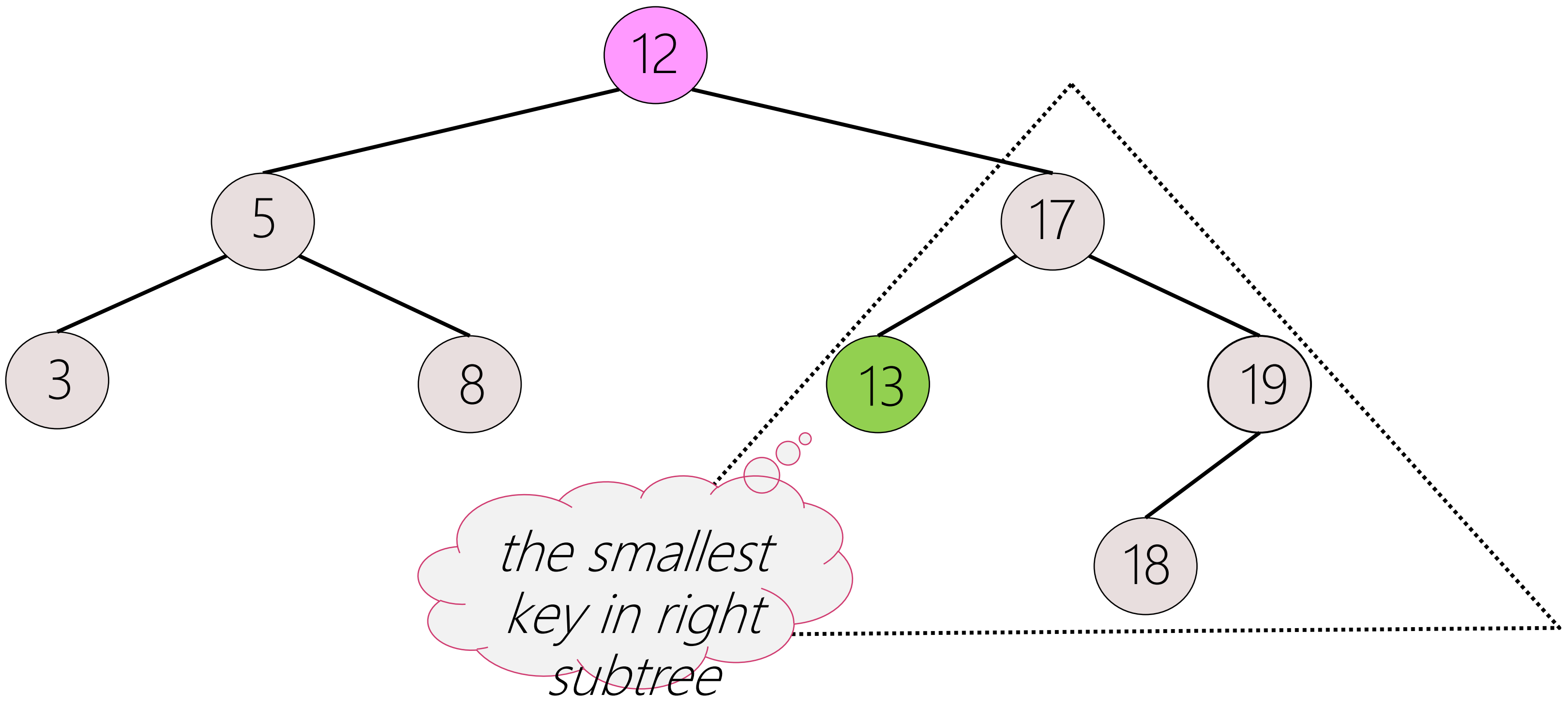
- *Delete*(12, *D*)

BST: Delete(x, D) – From a Deg. 2 Node (1)



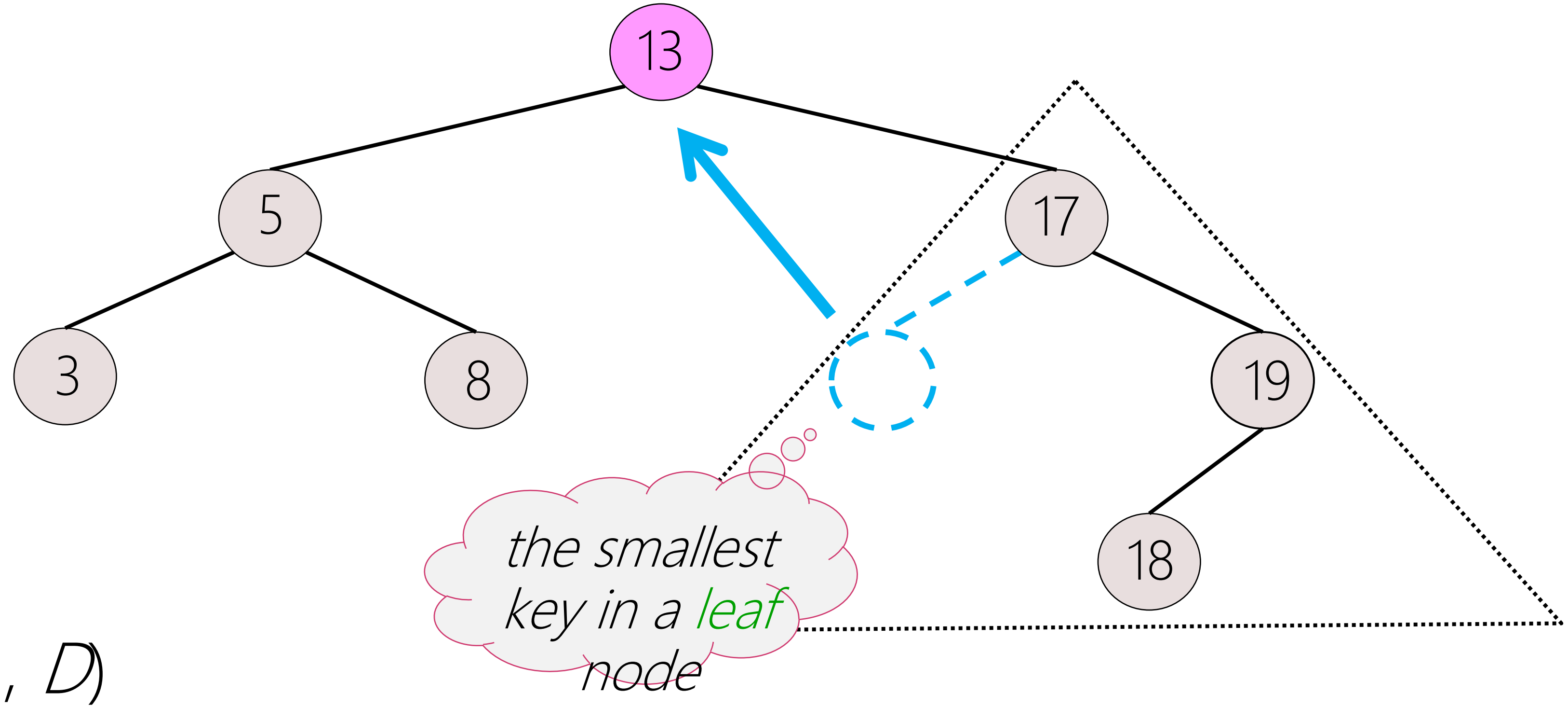
- *Delete*(12, *D*)
  - Identify the location of the node whose *key* = 12
- Hard case: two children

BST: Delete(*x*, *D*) – From a Deg. 2 Node (1)



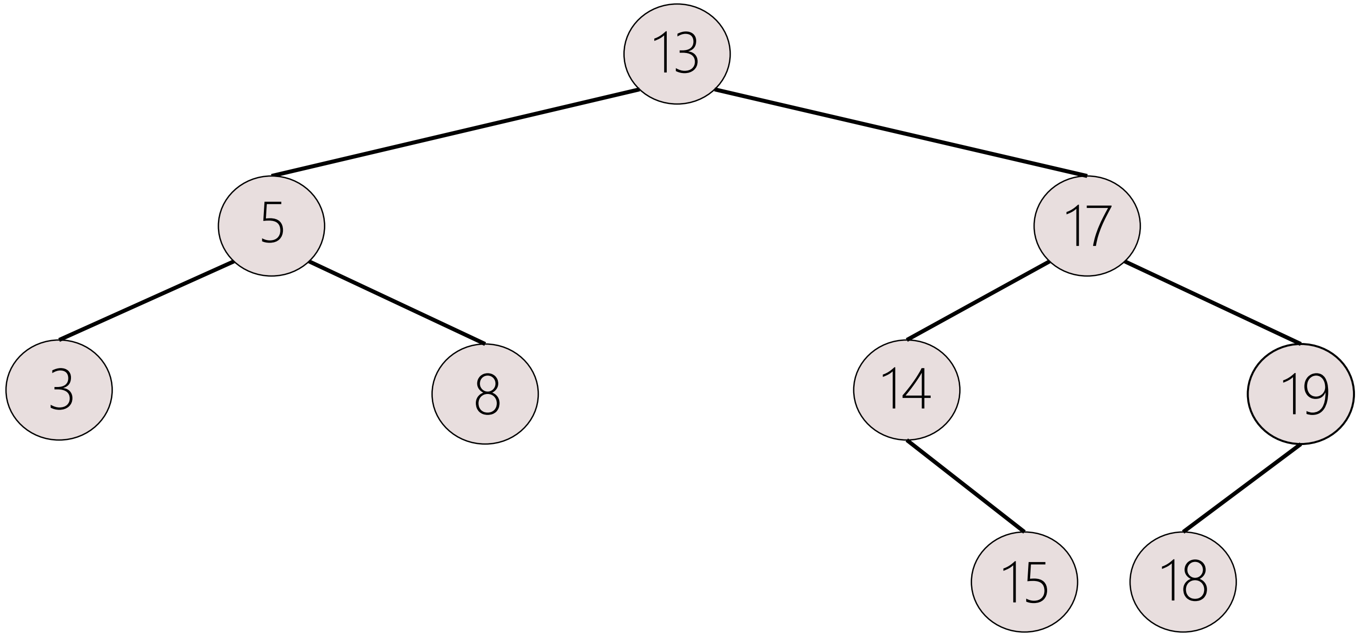
- *Delete*(12, *D*)
  - Identify the location of the node whose *key* = 12
  - Replace it with the smallest key in its right subtree (or the largest key in its left subtree)

BST: Delete(x, D) – From a Deg. 2 Node (1)



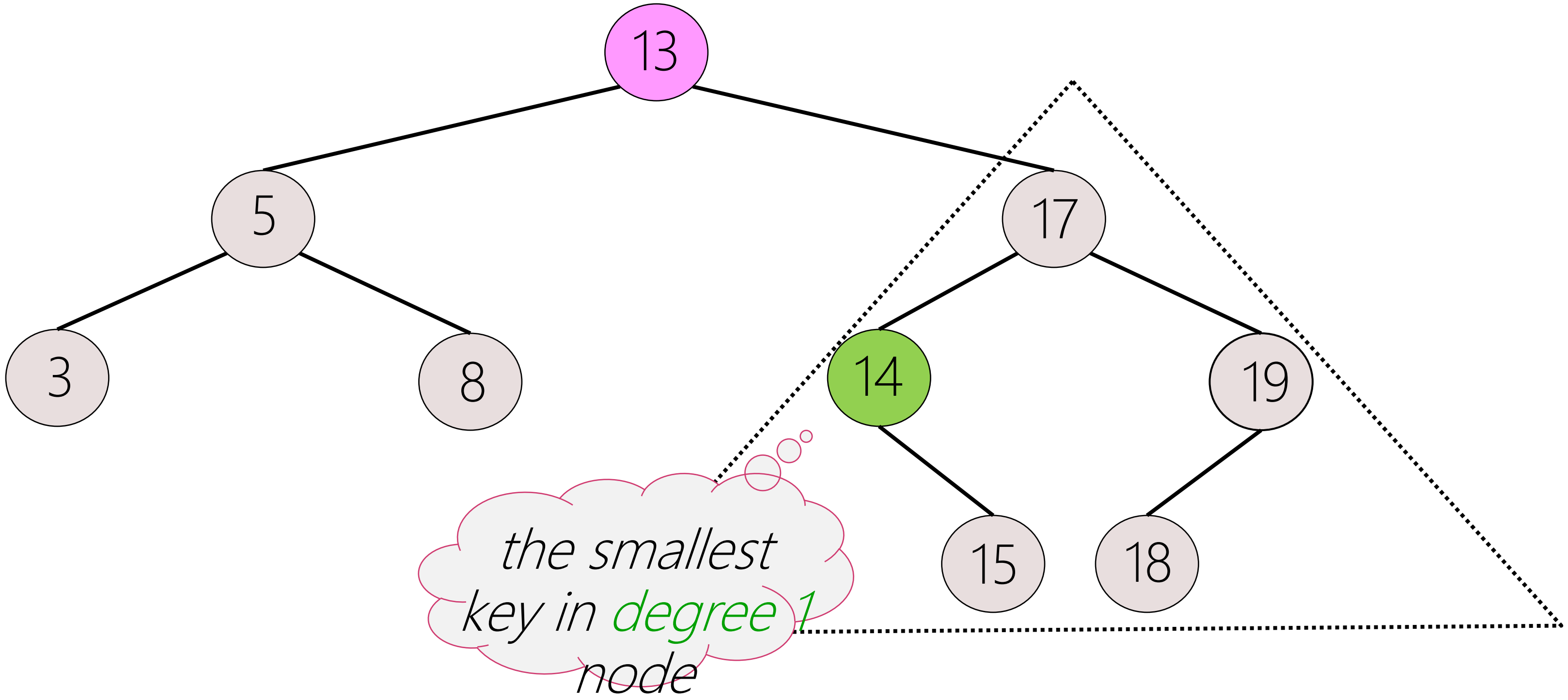
- After *Delete*(12, *D*)
- Note:
  - The smallest key in the right subtree must be in a leaf (like in this example) or degree 1 node (like in the next example)

BST: Delete(x, D) – From a Deg. 2 Node (2)



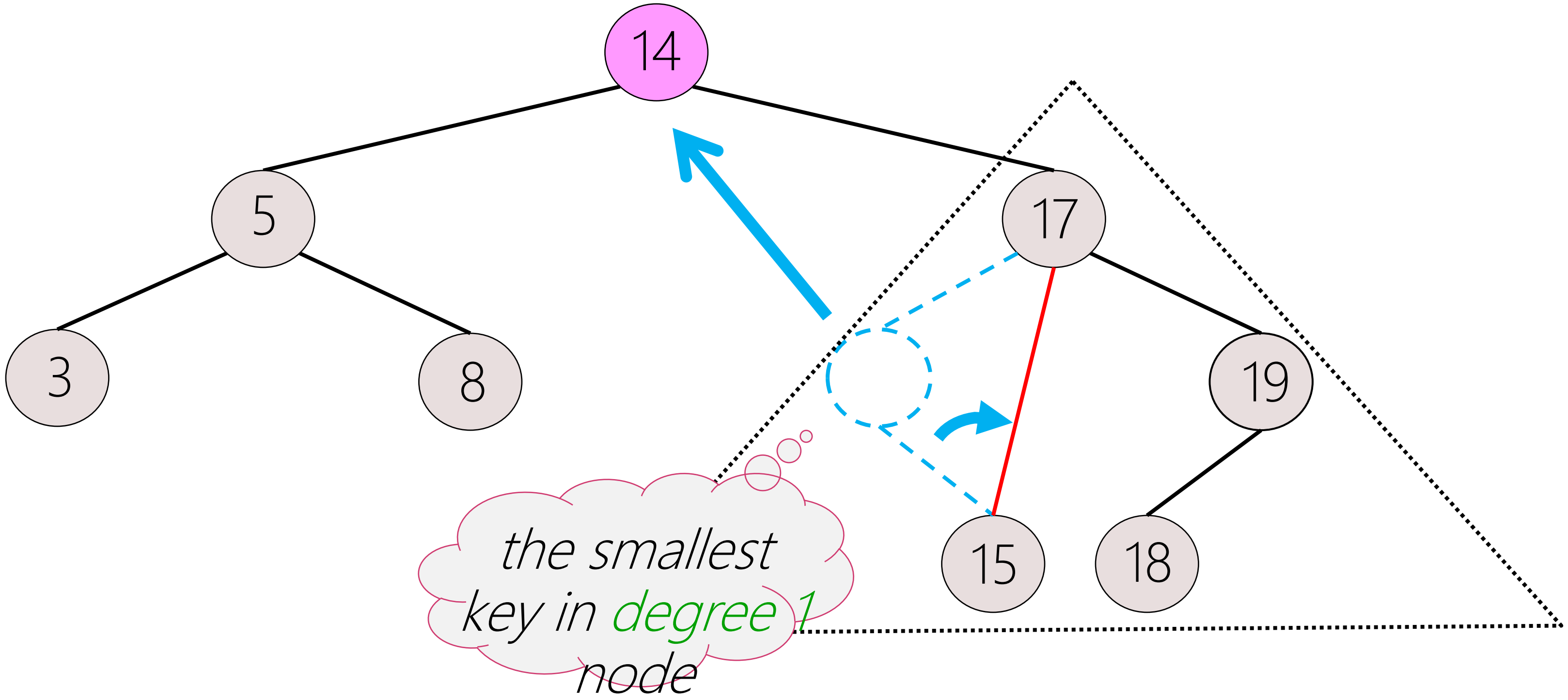
- After *Insert*(14, *D*) & *Insert*(15, *D*)
- Then *Delete*(13, *D*)

BST: *Delete*(*x*, *D*) – From a Deg. 2 Node (2)



- *Delete*(13, *D*)
  - Here, the smallest key is in a degree 1 node

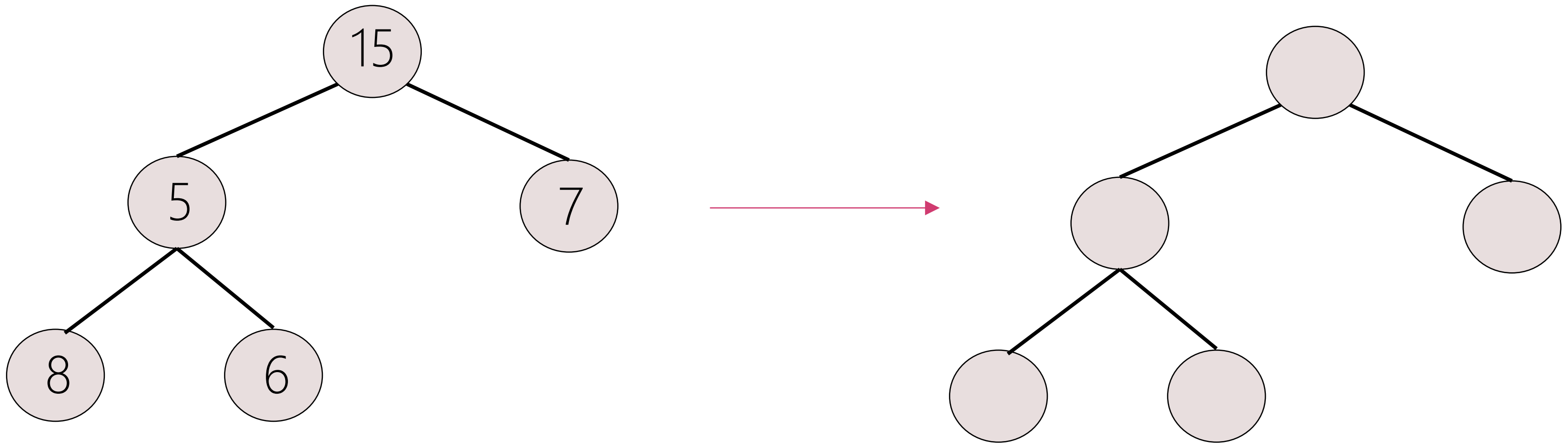
BST: Delete(x, D) – From a Deg. 2 Node (2)



- After *Delete*(13, D)
- Time complexity = ?   O(*height*)

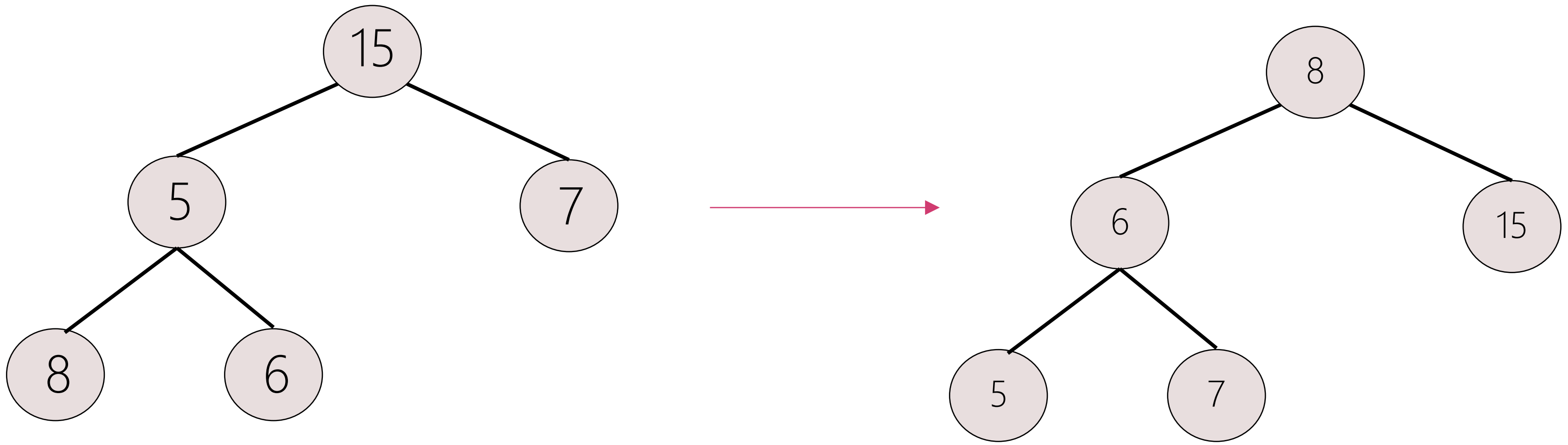
Convert a Binary Tree into a Binary Search Tree?

- Keep the structure same, but change the values only



Convert a Binary Tree into a Binary Search Tree?

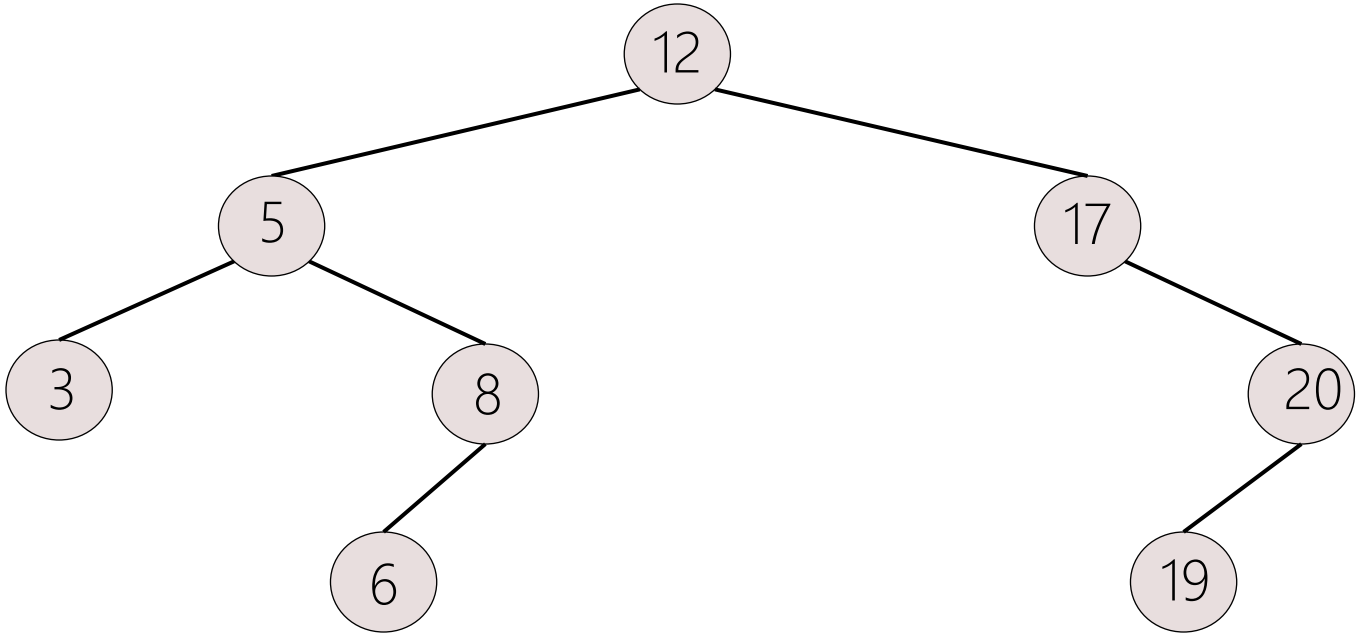
- Keep the structure same, but change the values only



1. Inorder traversal
2. Sort the traversal list
3. Copy the sorted list during another traversal

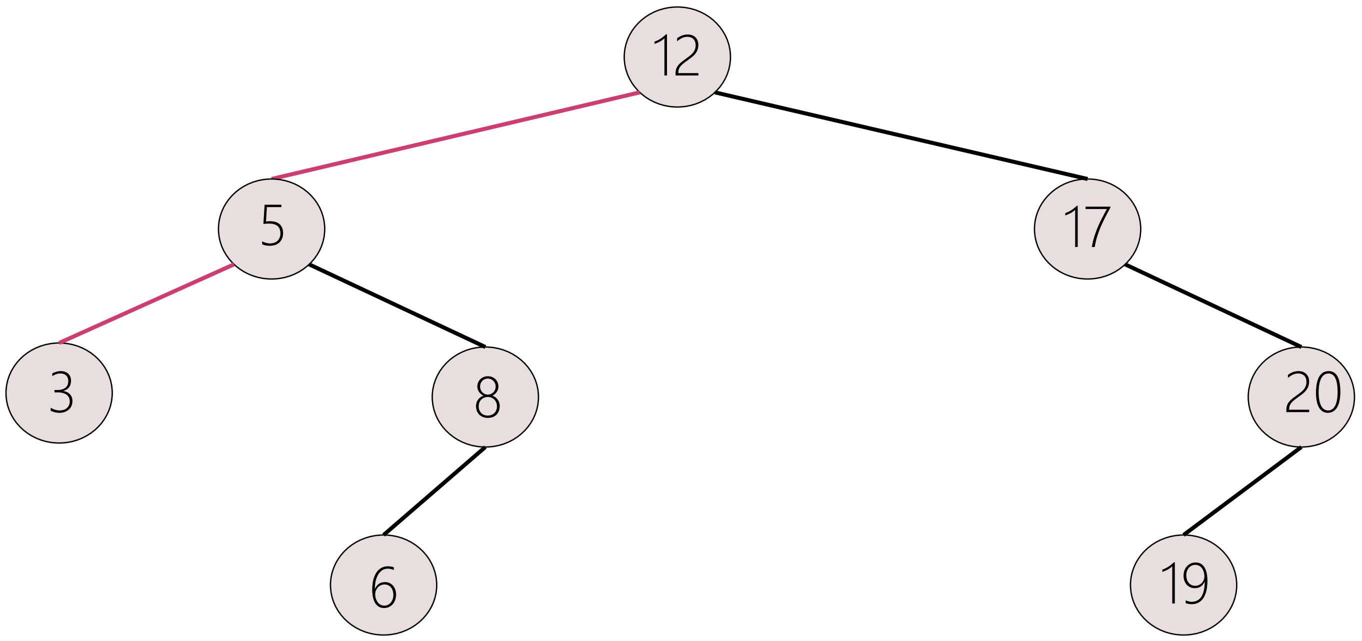
Time complexity?

BST: SearchMinimum(D)



- Find the node with minimum value

BST: SearchMinimum(D)



- Find the node with minimum value
  - Traverse the node from root to left

References

- Further reading list and references
  - https://www.geeksforgeeks.org/binary-search-tree-data-structure/

- Slide credit
  - Jaesik Park
  - Seung-Hwan Baek
  - Jong-Hyeok Lee