

Dictionary

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted
on the internet without the written permission of the copyright owners.



Abstract Data Types Seen So Far

	Insert	Min	Extract_Min	Union
Priority Queues	✓	✓	✓	✗
Mergeable Priority Queues	✓	✓	✓	✓



Abstract Data Type: Dictionary

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted
on the internet without the written permission of the copyright owners.



Abstract Data Type: Dictionary

- Object : Set S of (elements with) keys



Abstract Data Type: Dictionary

- Object : Set S of (elements with) keys
- Operations:



Abstract Data Type: Dictionary

- Object : Set S of (elements with) keys
- Operations:
 - **Search(S, x)**: Returns element with key x, if x is in S; else return “Not Found”



Abstract Data Type: Dictionary

- Object : Set S of (elements with) keys
- Operations:
 - **Search(S, x)**: Returns element with key x, if x is in S; else return “Not Found”
 - **Insert(S, x)**: Inserts x in S



Abstract Data Type: Dictionary

- Object : Set S of (elements with) keys
- Operations:
 - **Search(S, x):** Returns element with key x , if x is in S ; else return “Not Found”
 - **Insert(S, x):** Inserts x in S
 - **Delete(S, x):** Deletes x from S



Data Structures for Dictionaries

Linked List



Data Structures for Dictionaries

Linked List

- **Insert**



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$
- **Search**



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$
- **Search** : $\Theta(n)$



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$
- **Search** : $\Theta(n)$
- **Delete** :



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$
- **Search** : $\Theta(n)$
- **Delete** : $\Theta(n)$



Data Structures for Dictionaries

Linked List

- **Insert** : $\Theta(1)$
- **Search** : $\Theta(n)$
- **Delete** : $\Theta(n)$

Goal: Data Structure that does each operation in $\Theta(\log n)$ time

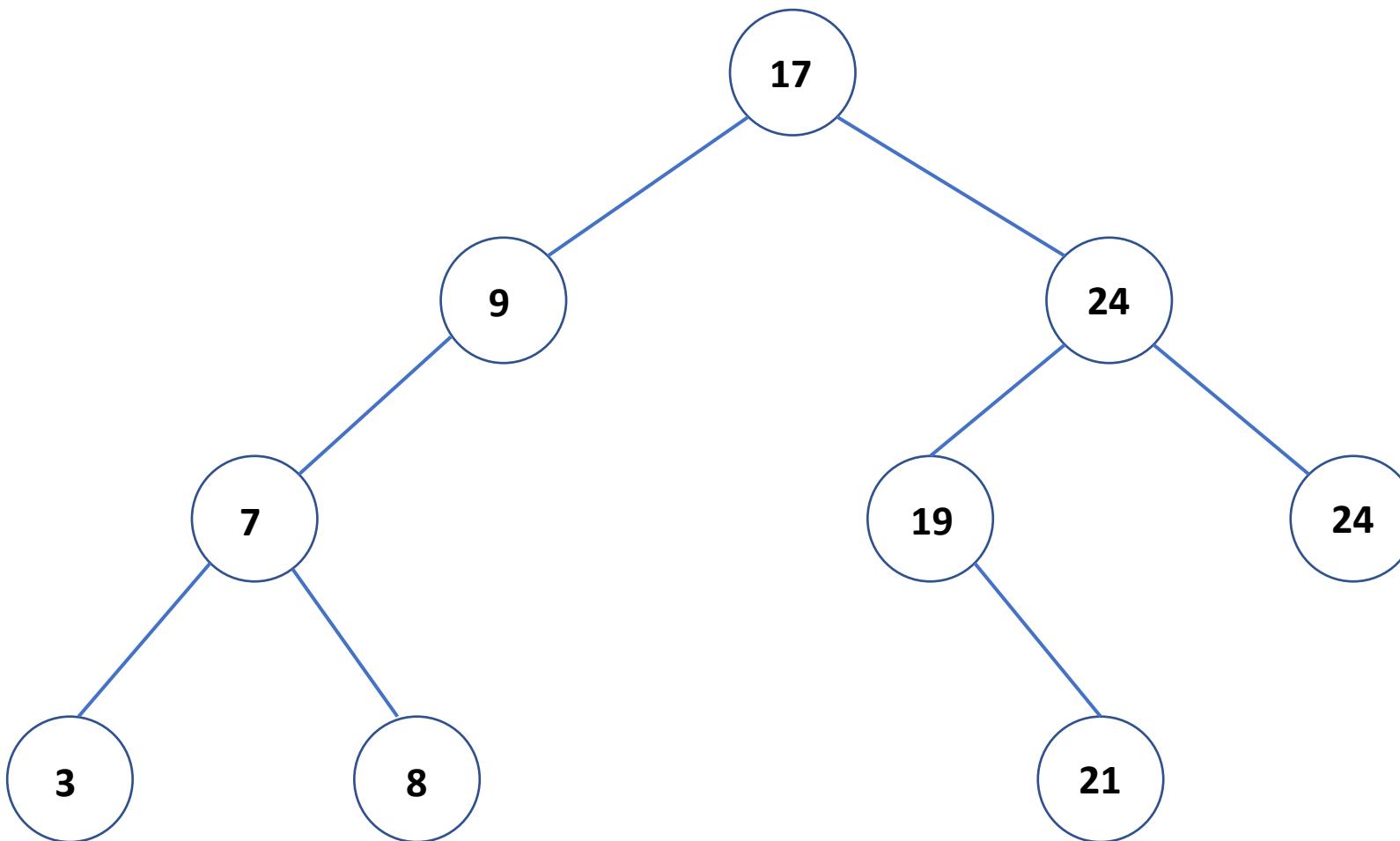


Binary Search Trees

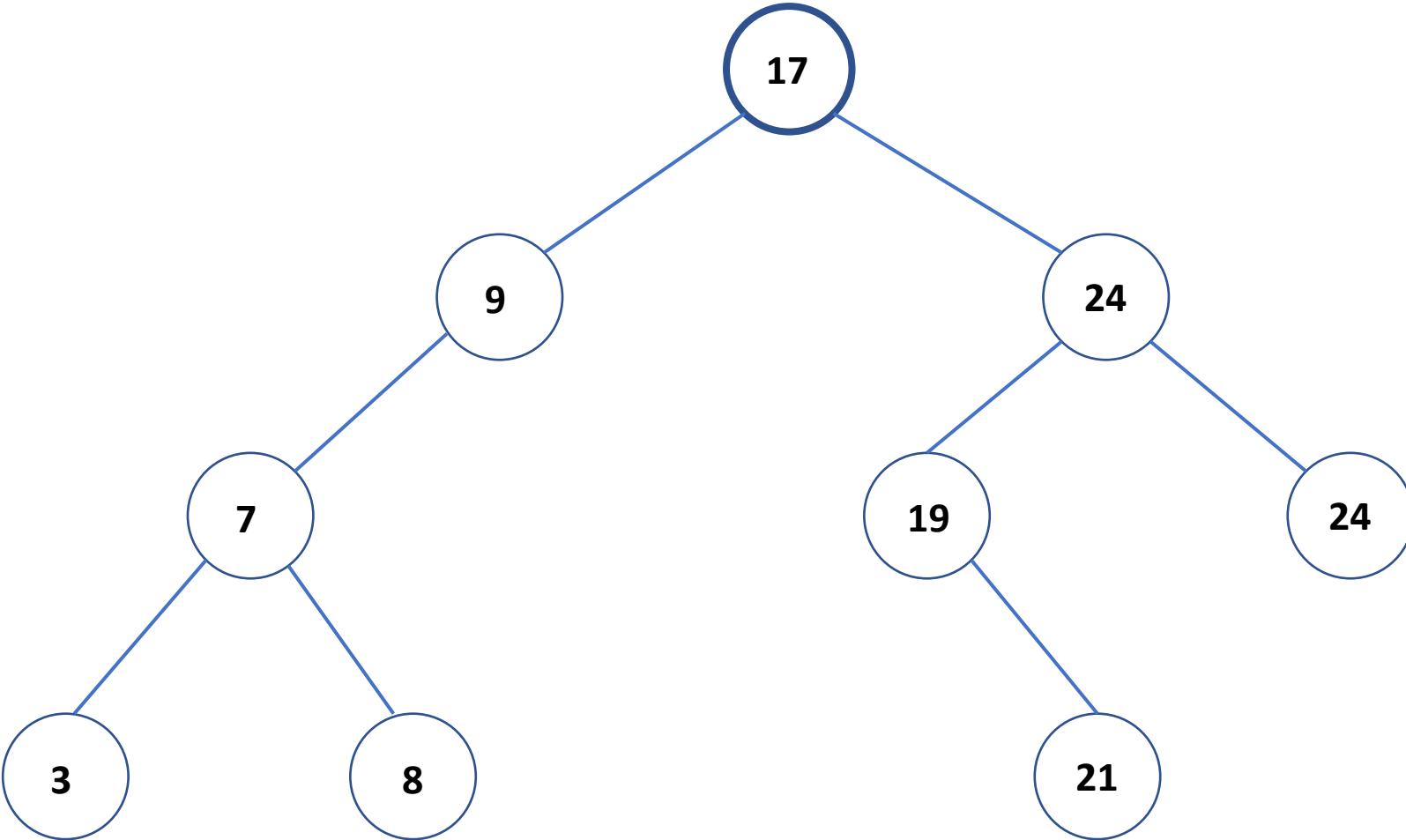
© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted
on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)



Binary Search Tree (BST)

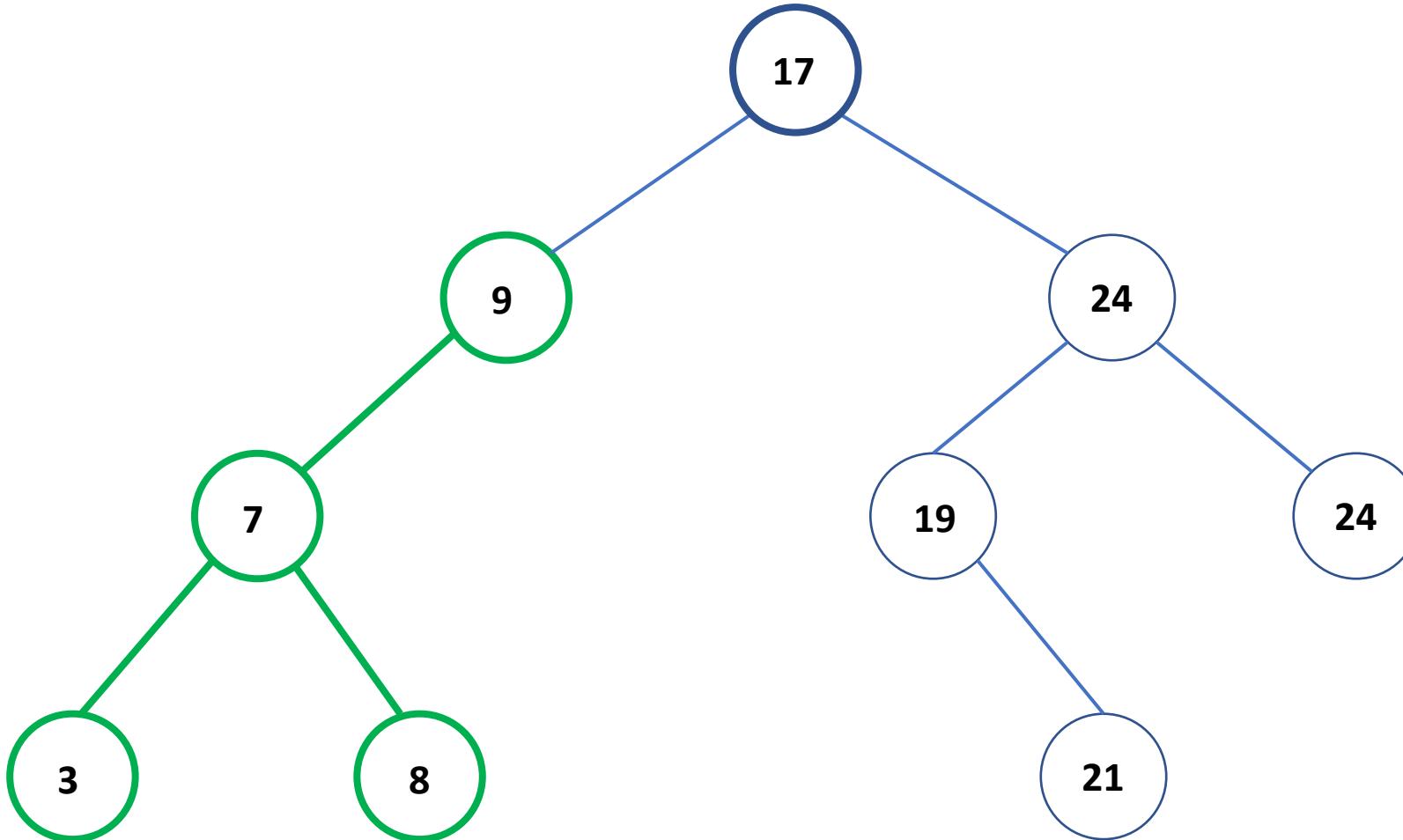


For each node

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

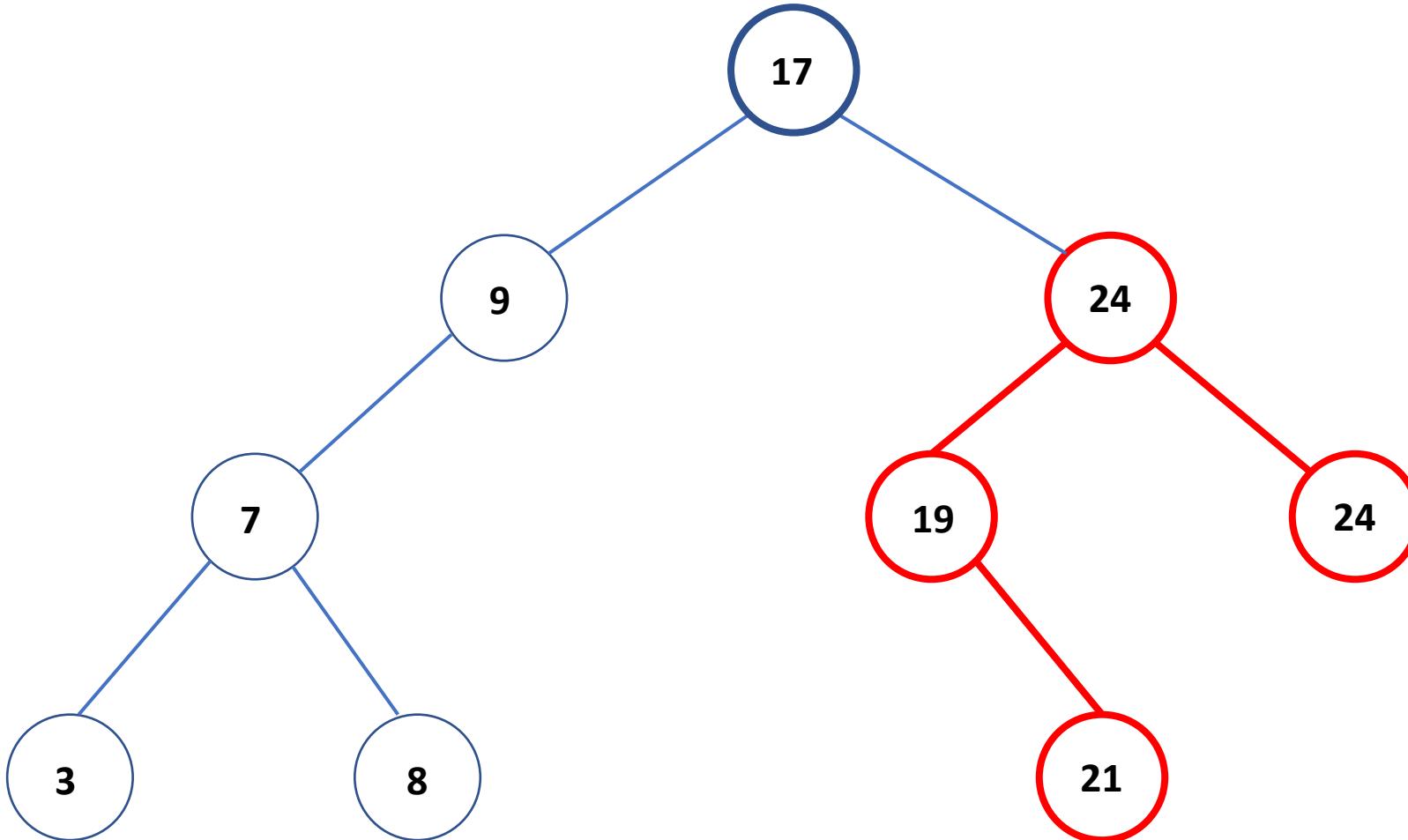


For each node: keys in **left** subtree \leq the node's key

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

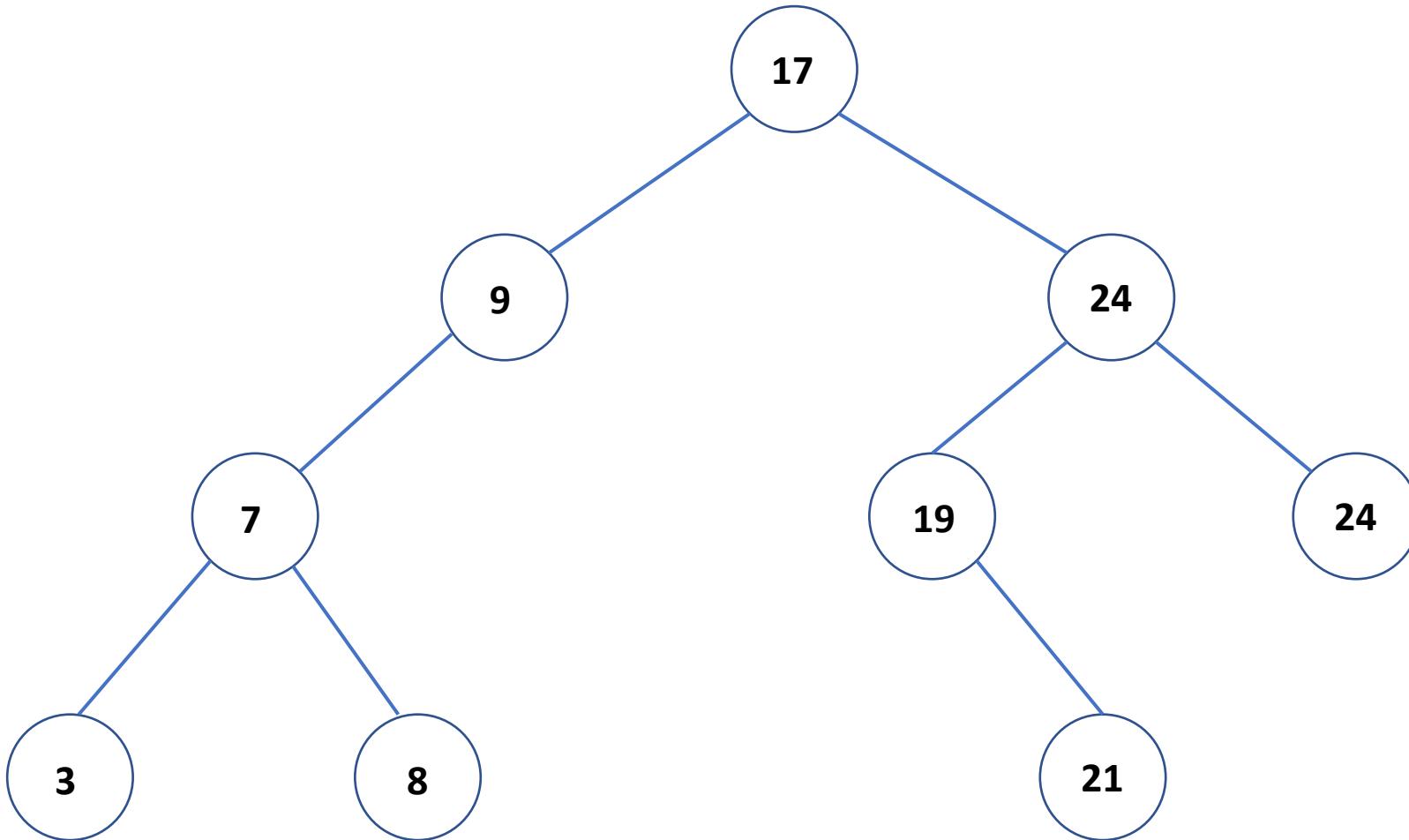


For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

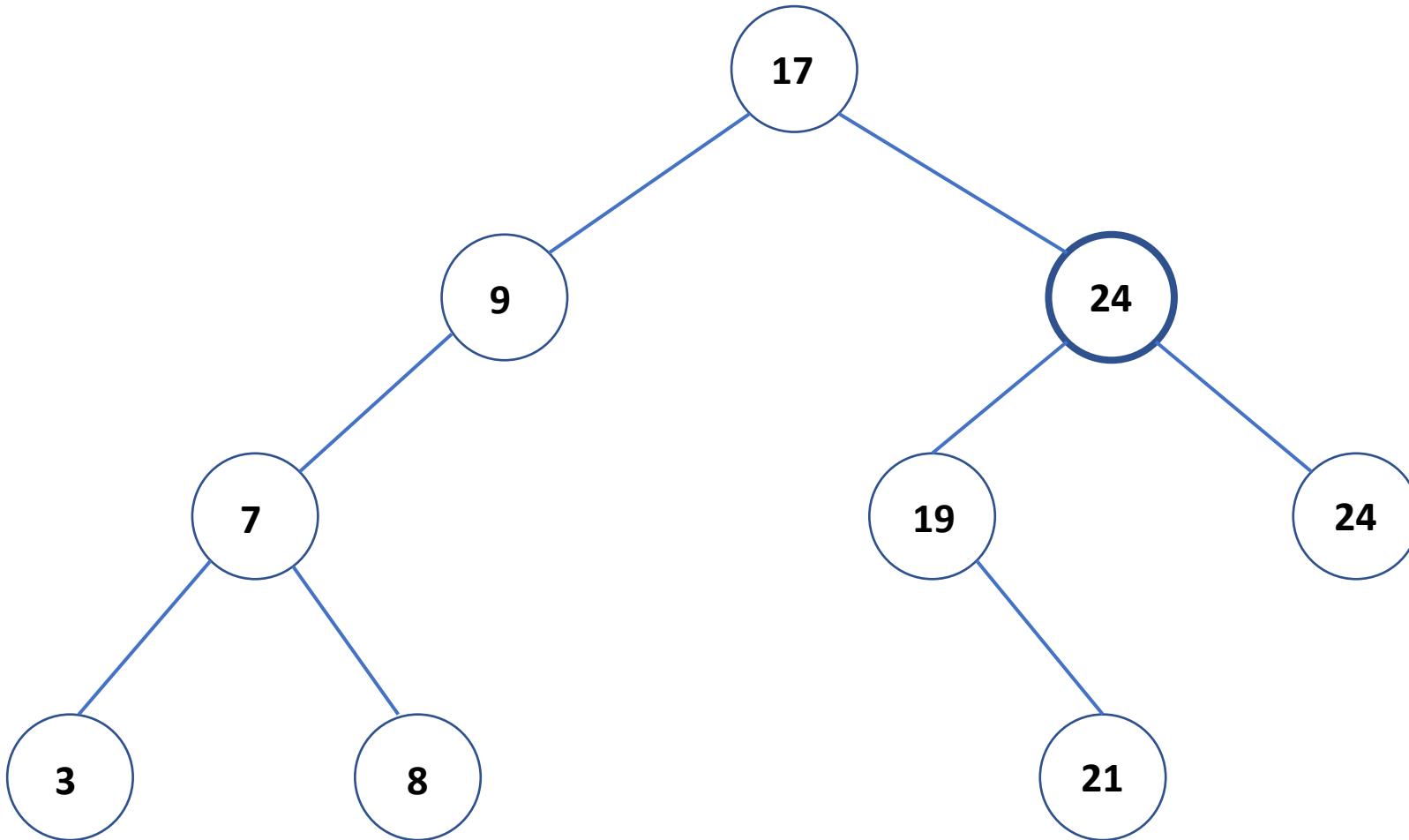


For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

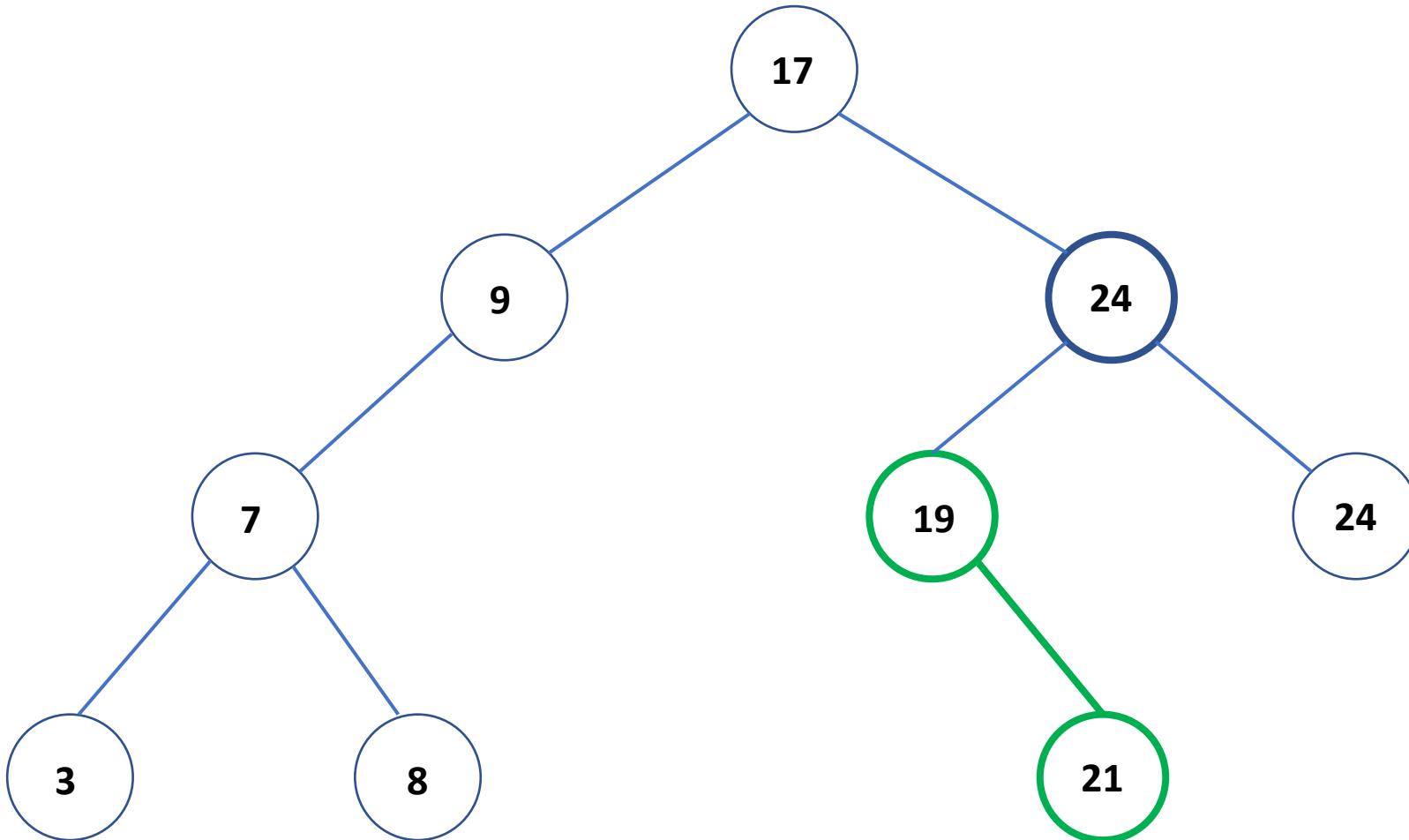


For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

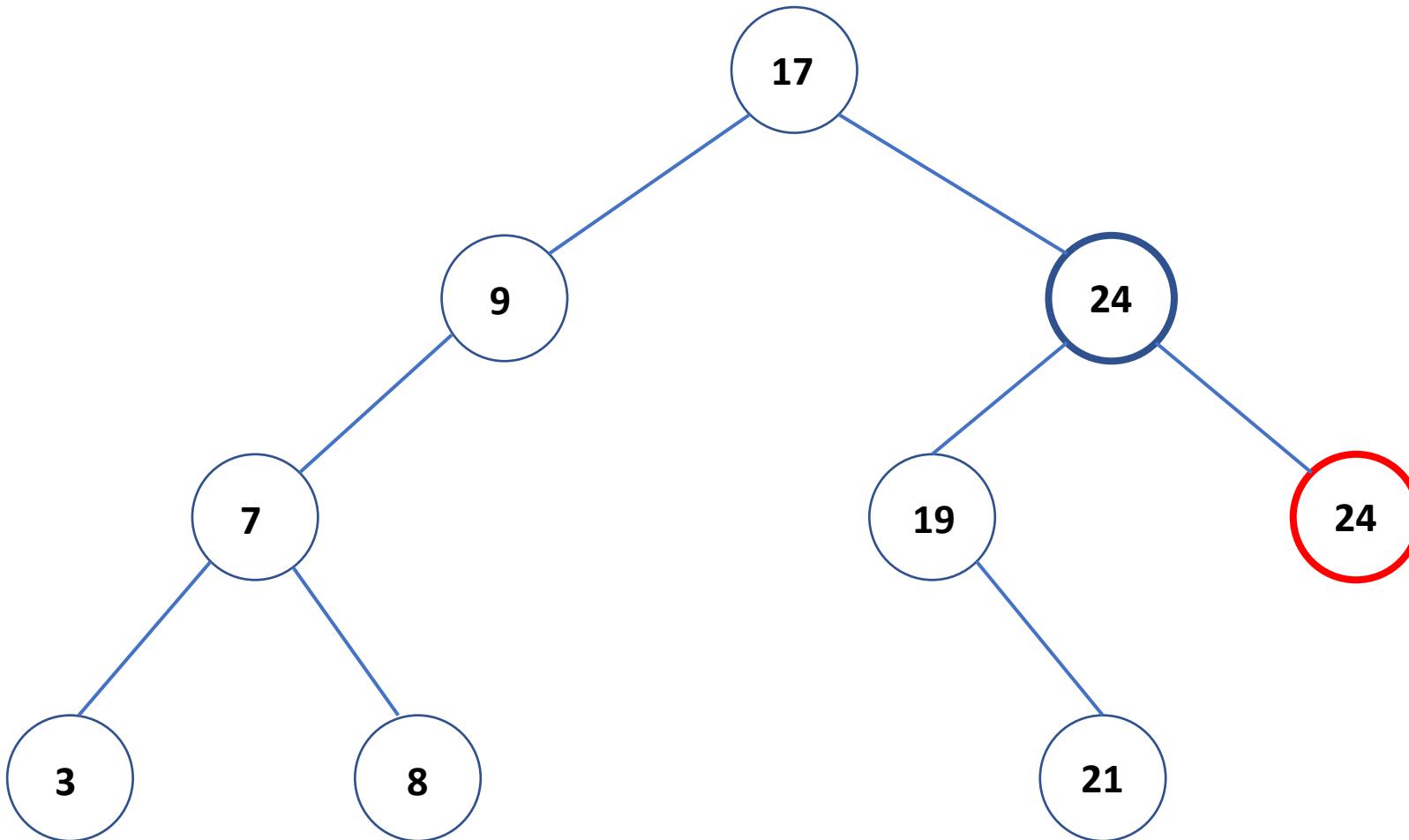


For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Binary Search Tree (BST)

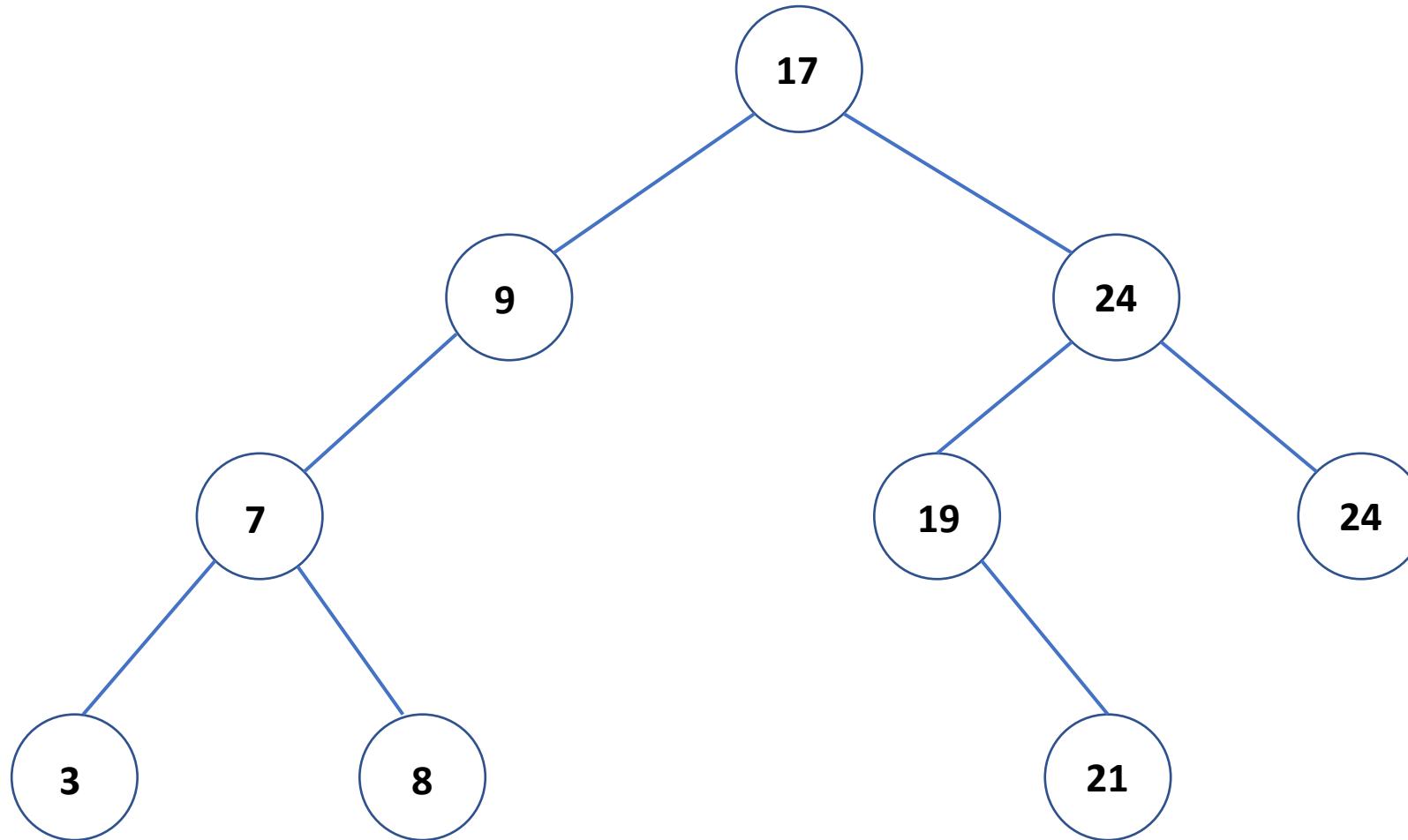


For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



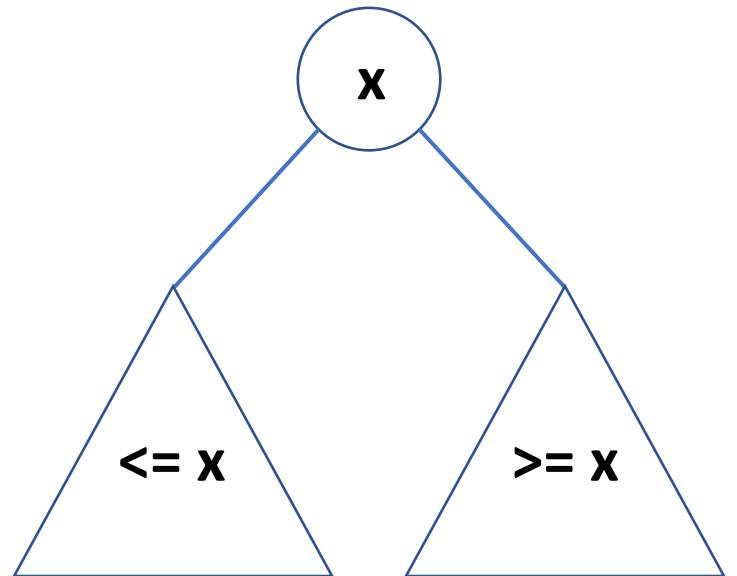
Binary Search Tree (BST)



For each node: keys in **left** subtree \leq the node's key \leq keys in **right** subtree

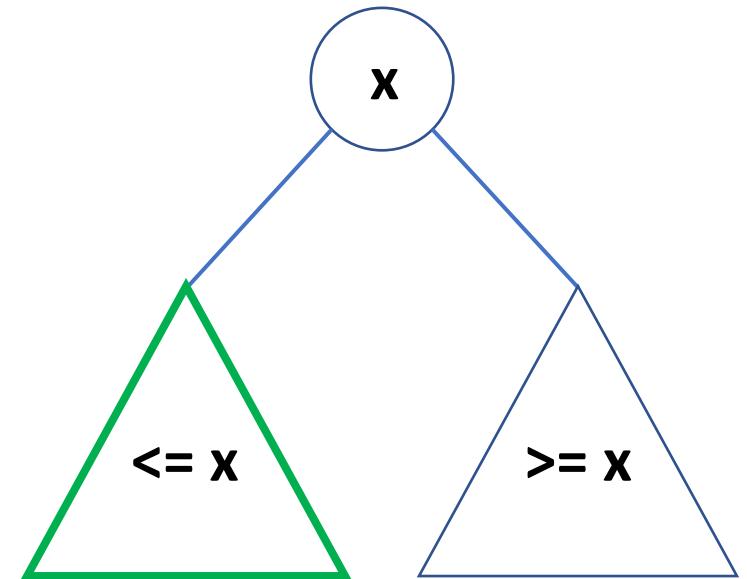


In-order Traversal



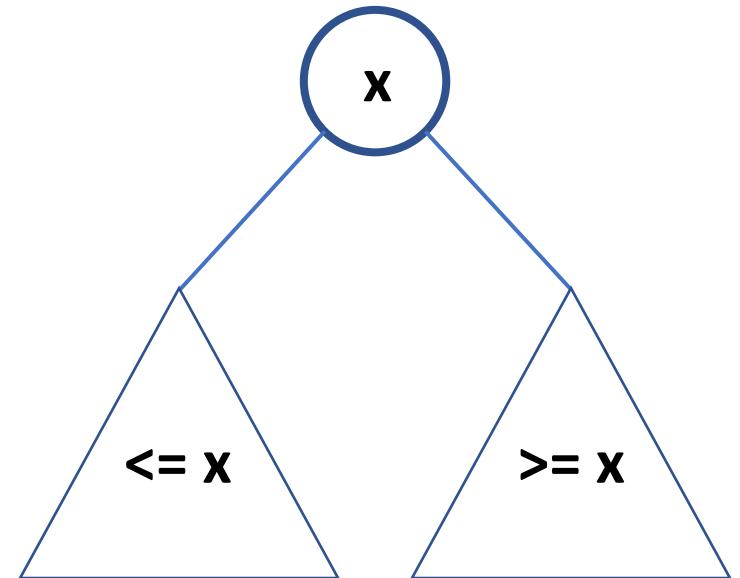
In-order Traversal

- Traverse node's **left subtree recursively**



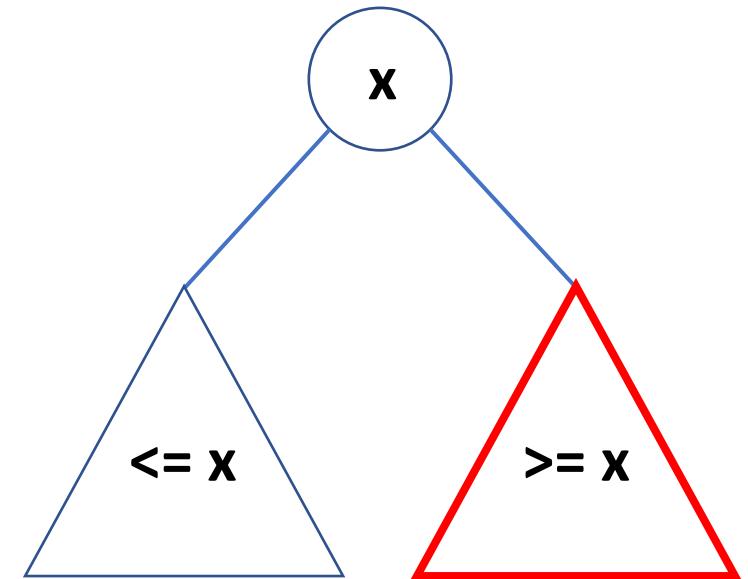
In-order Traversal

- Traverse node's **left subtree recursively**
- Visit node



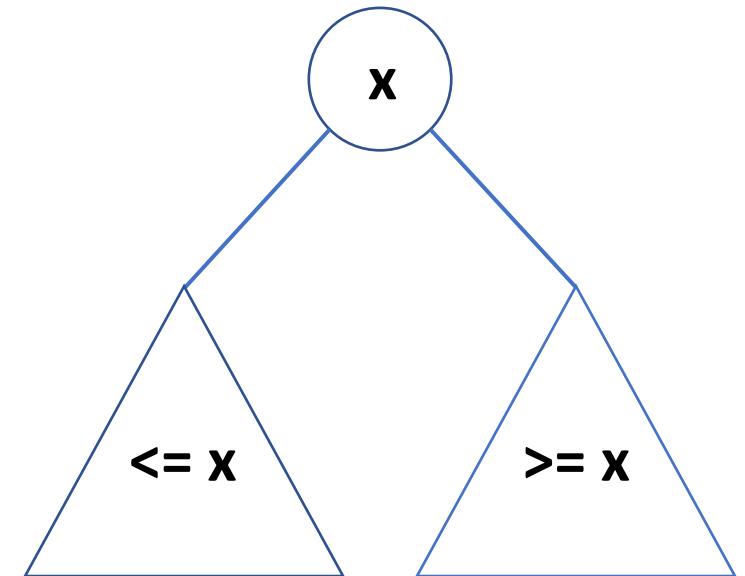
In-order Traversal

- Traverse node's **left subtree** recursively
- Visit node
- Traverse node's **right subtree** recursively



In-order Traversal

- Traverse node's **left subtree** recursively
- Visit node
- Traverse node's **right subtree** recursively



Fact: In-order traversal of a BST visits keys in ascending sorted order



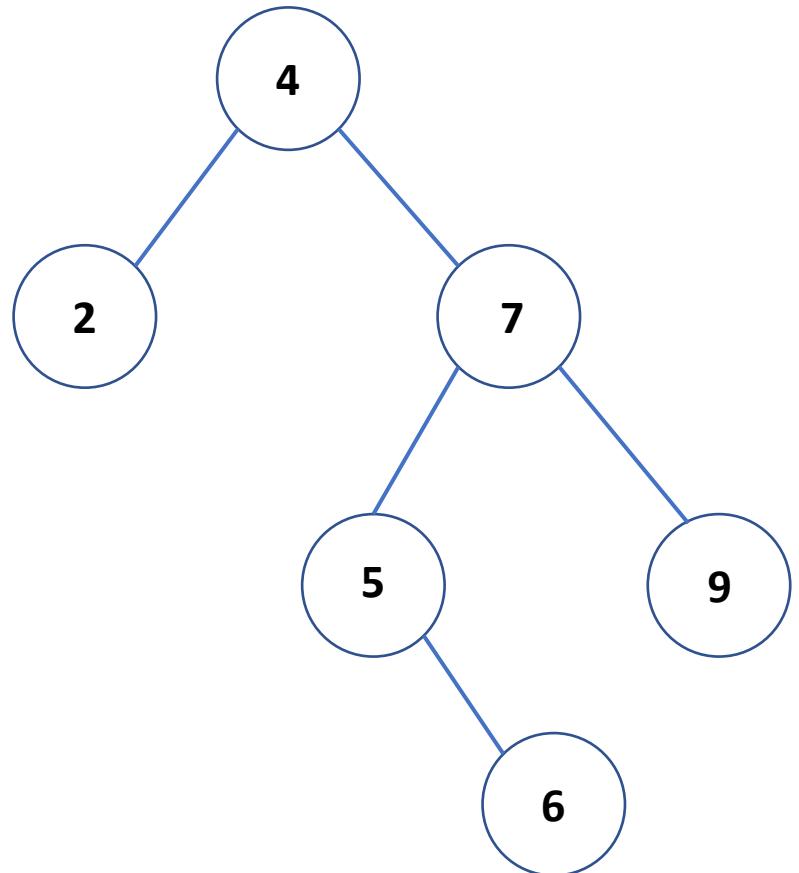
Is the BST for a set S of keys unique?



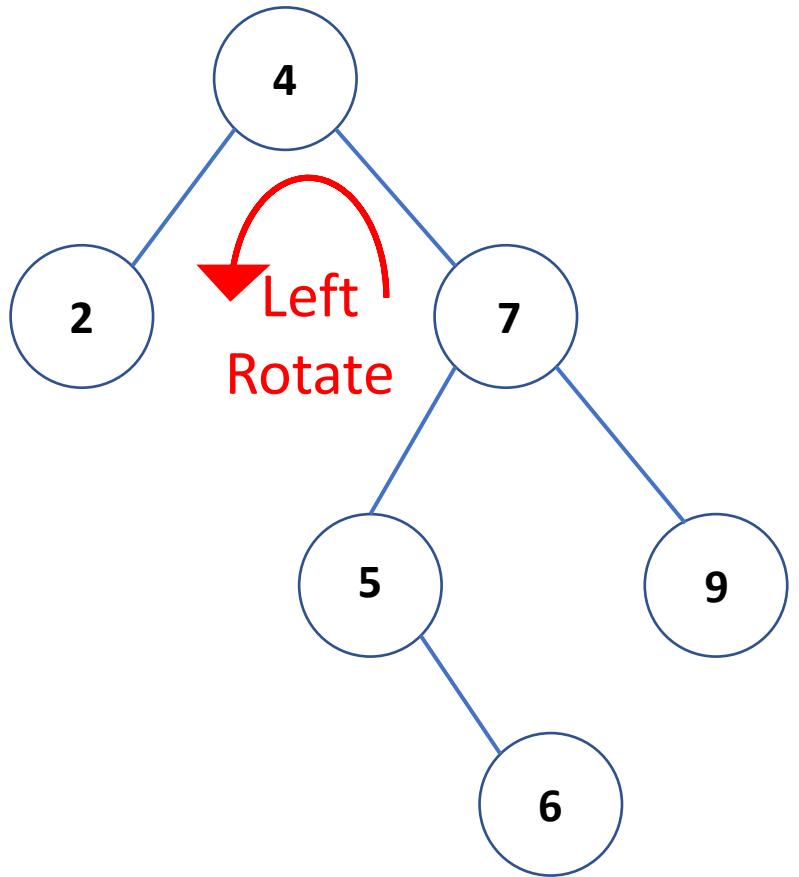
Is the BST for a set S of keys unique? **No!**



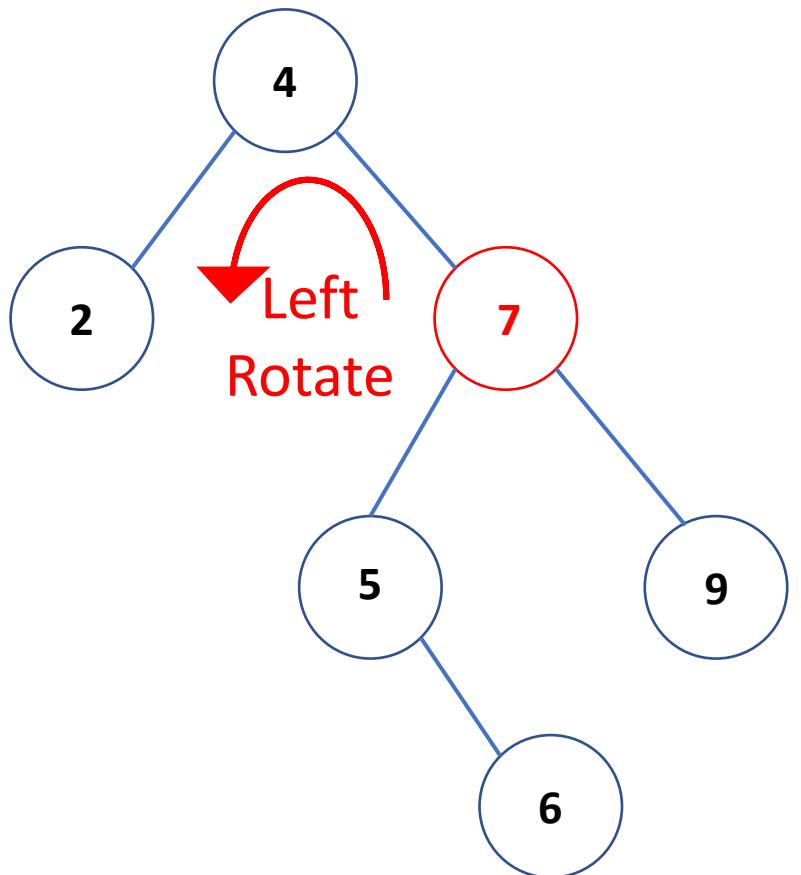
Is the BST for a set S of keys unique? No!



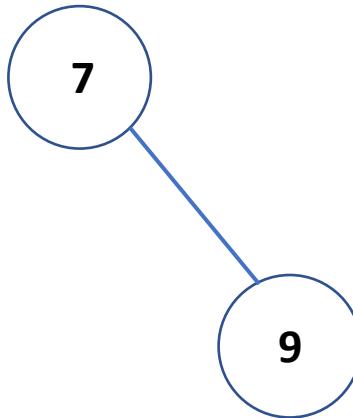
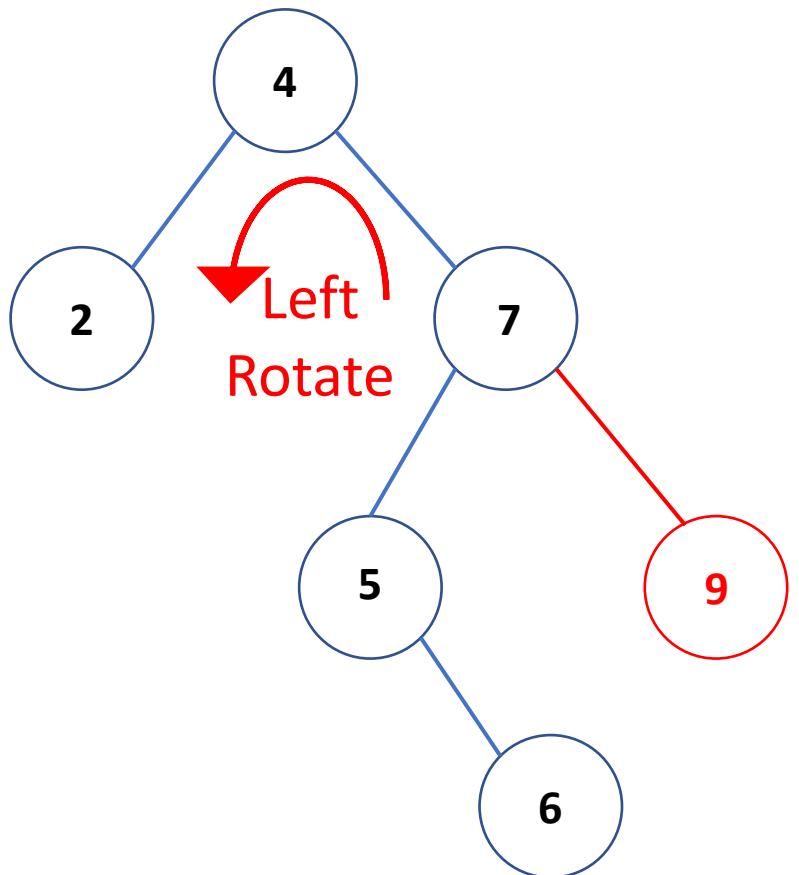
Is the BST for a set S of keys unique? No!



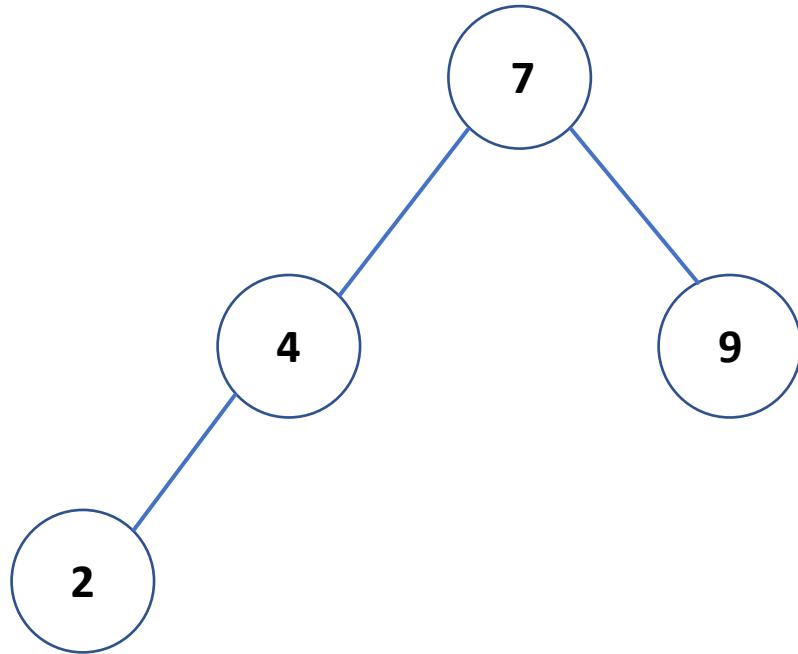
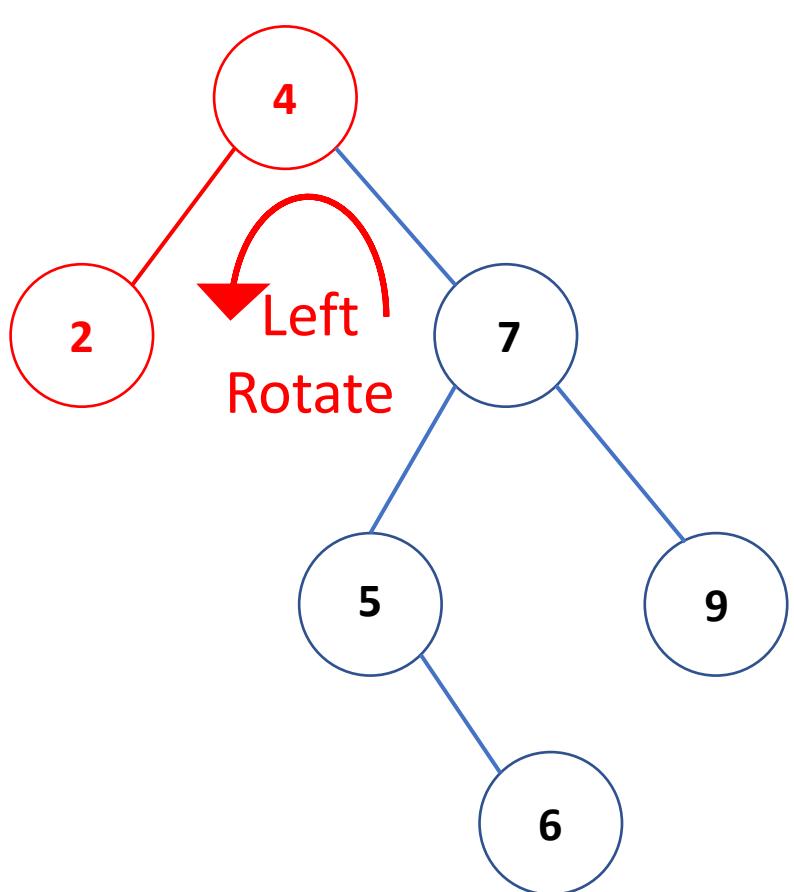
Is the BST for a set S of keys unique? No!



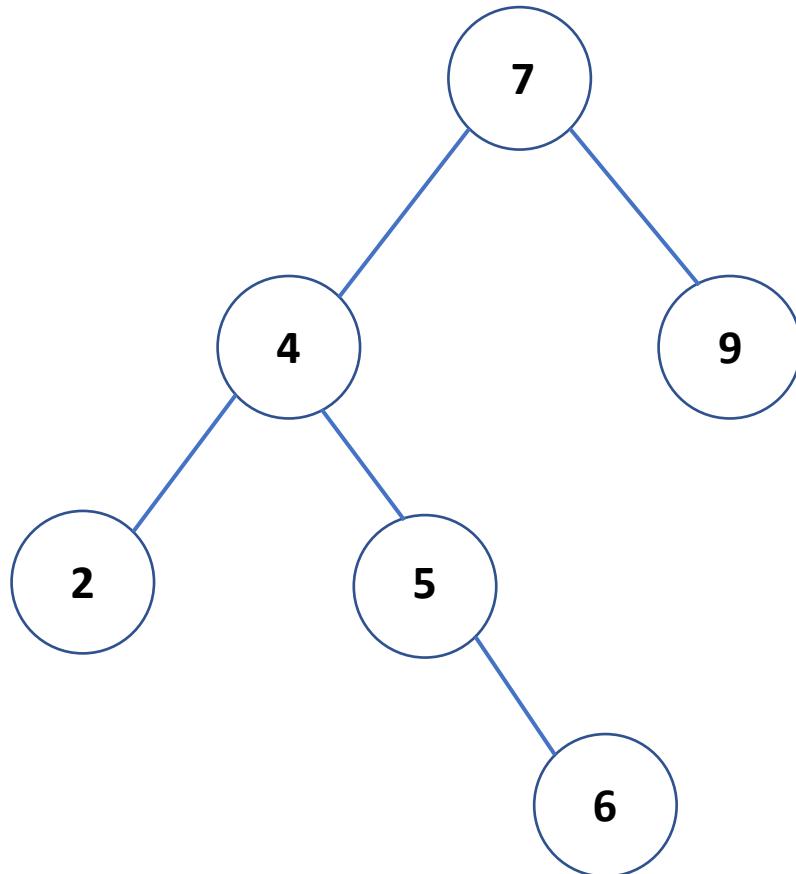
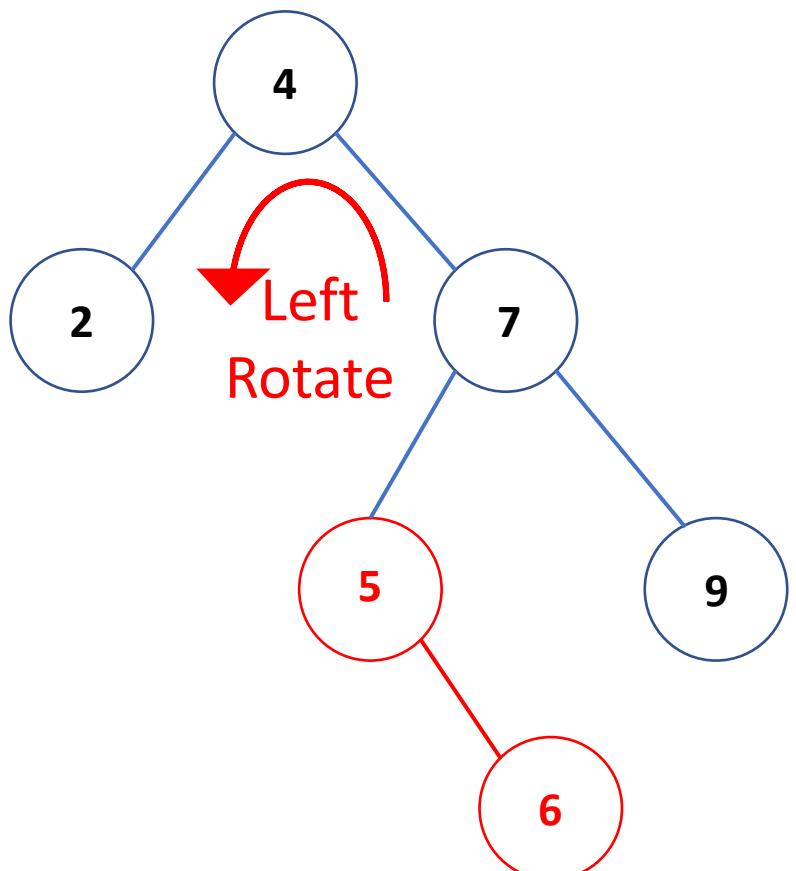
Is the BST for a set S of keys unique? No!



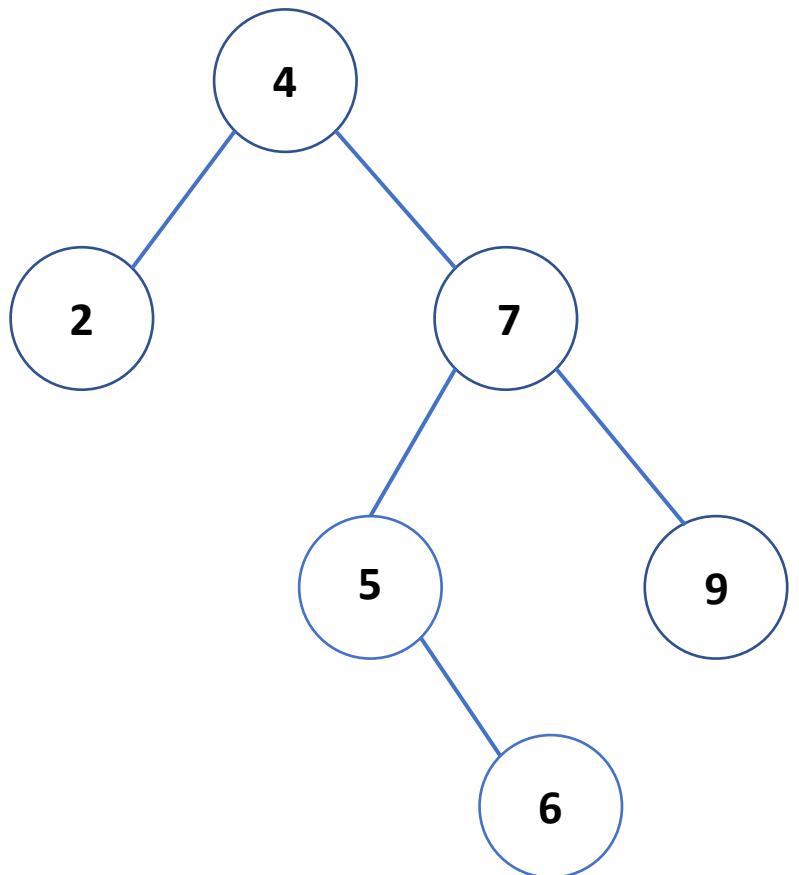
Is the BST for a set S of keys unique? No!



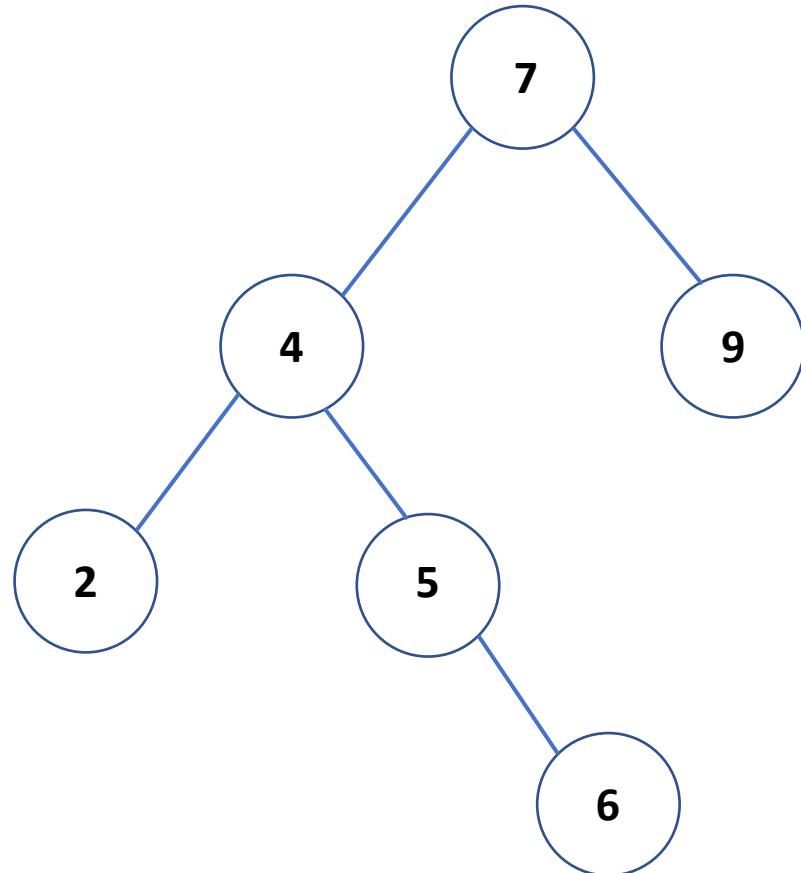
Is the BST for a set S of keys unique? No!



Is the BST for a set S of keys unique? No!



Left
Rotate



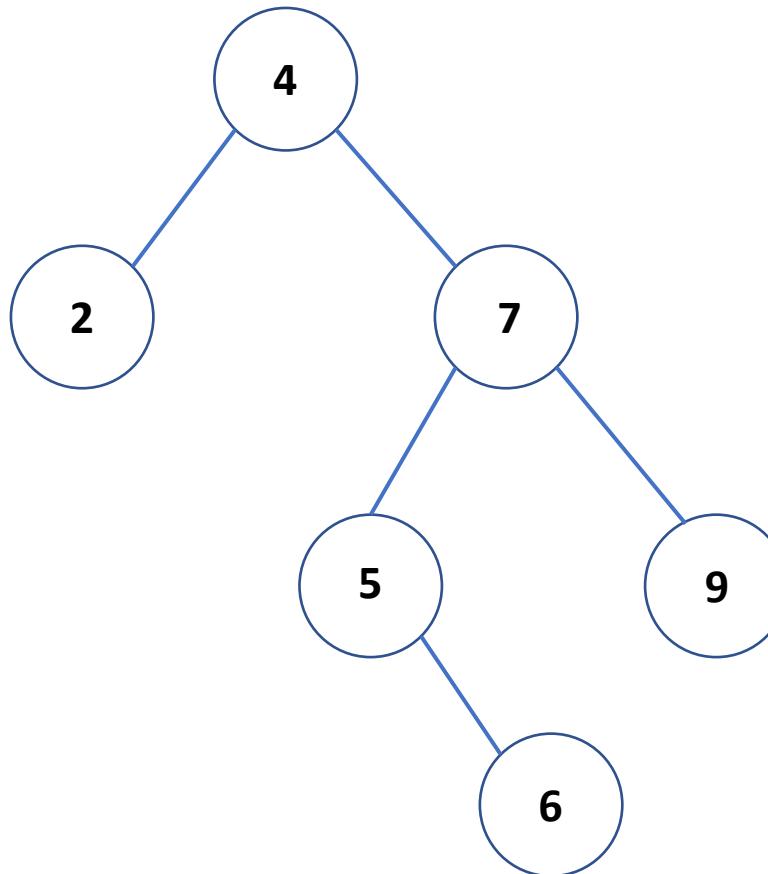
Operations on Binary Search Tree T

- **Search(T, x)**
- **Insert(T, x)**
- **Delete(T, x)**

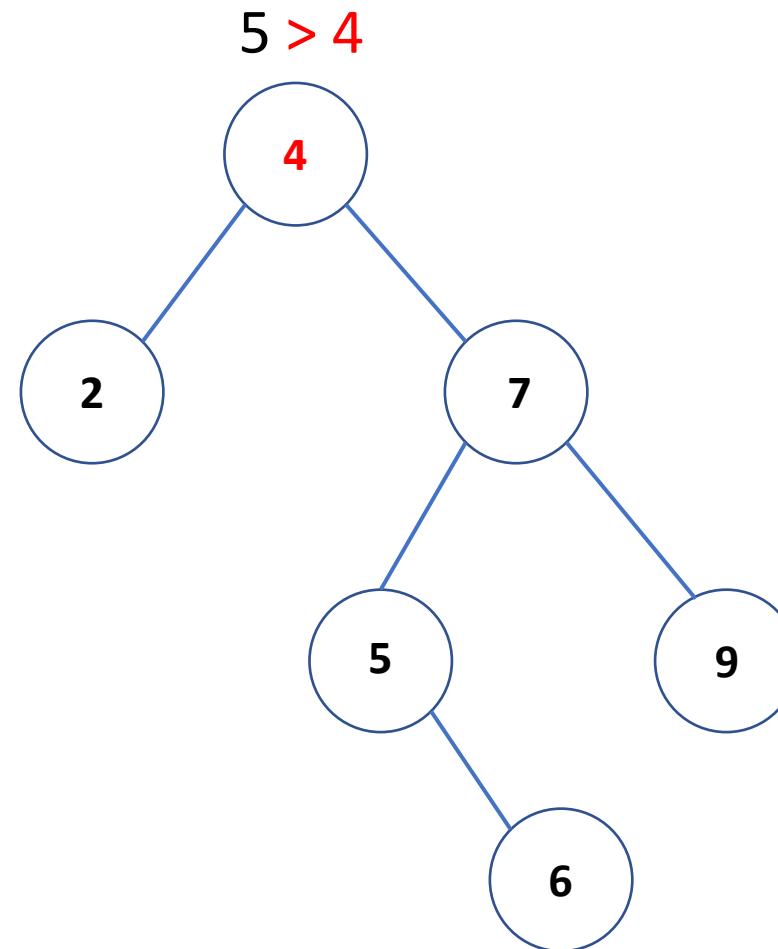
We show how they work by examples.



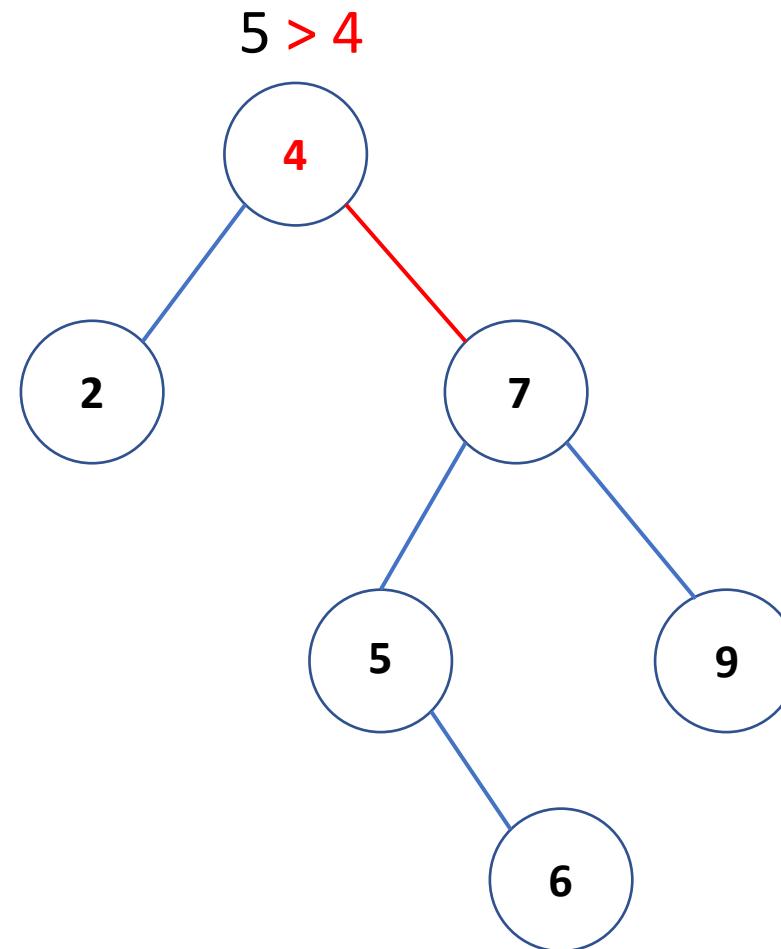
Search(T , 5)



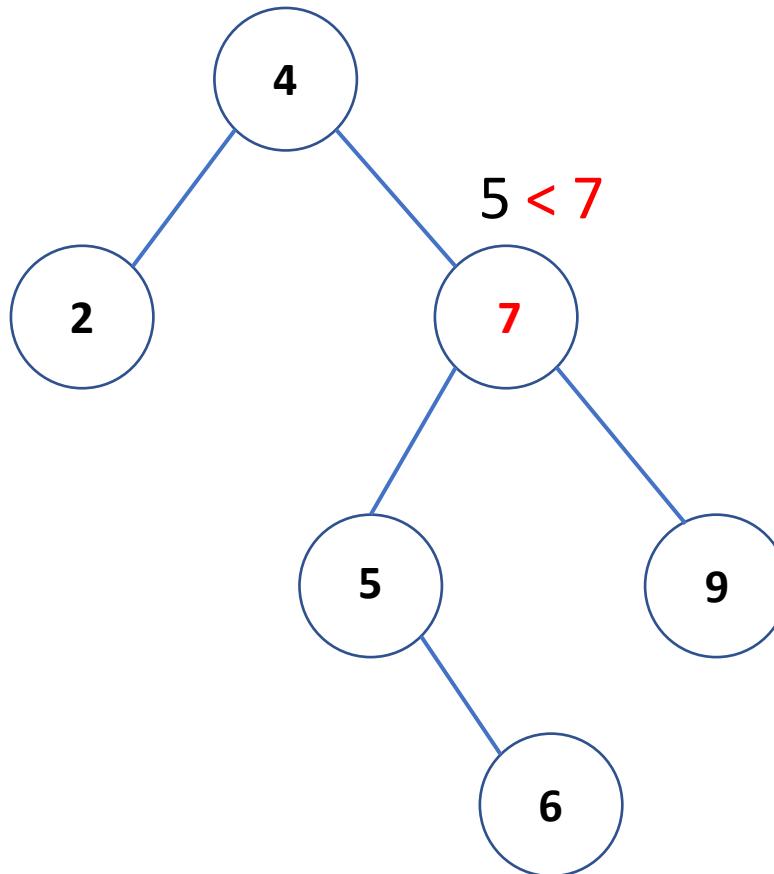
Search(T , 5)



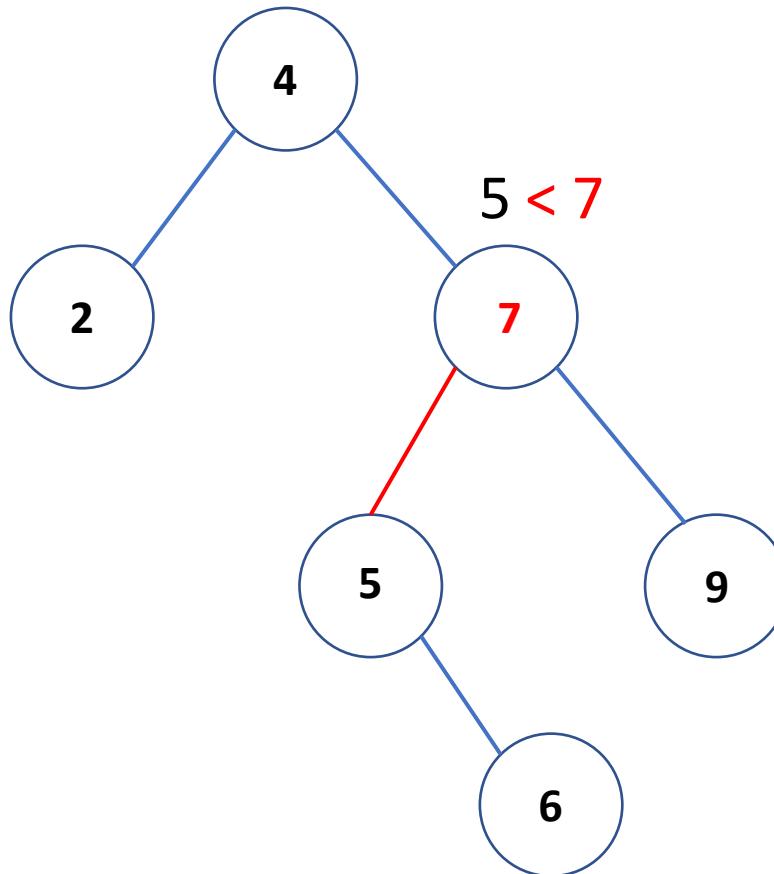
Search(T , 5)



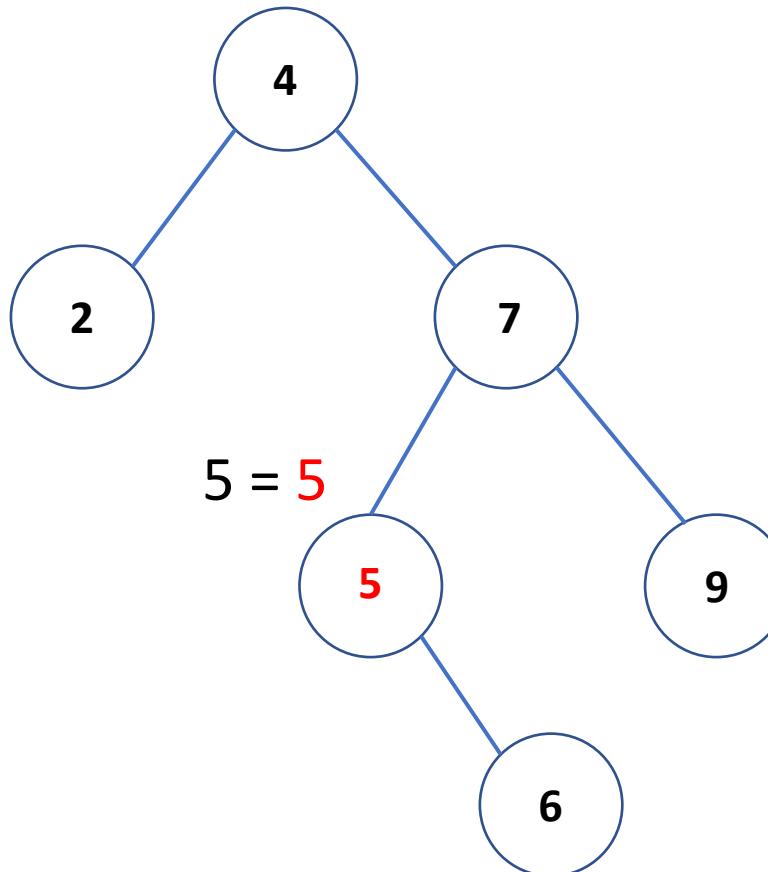
Search(T , 5)



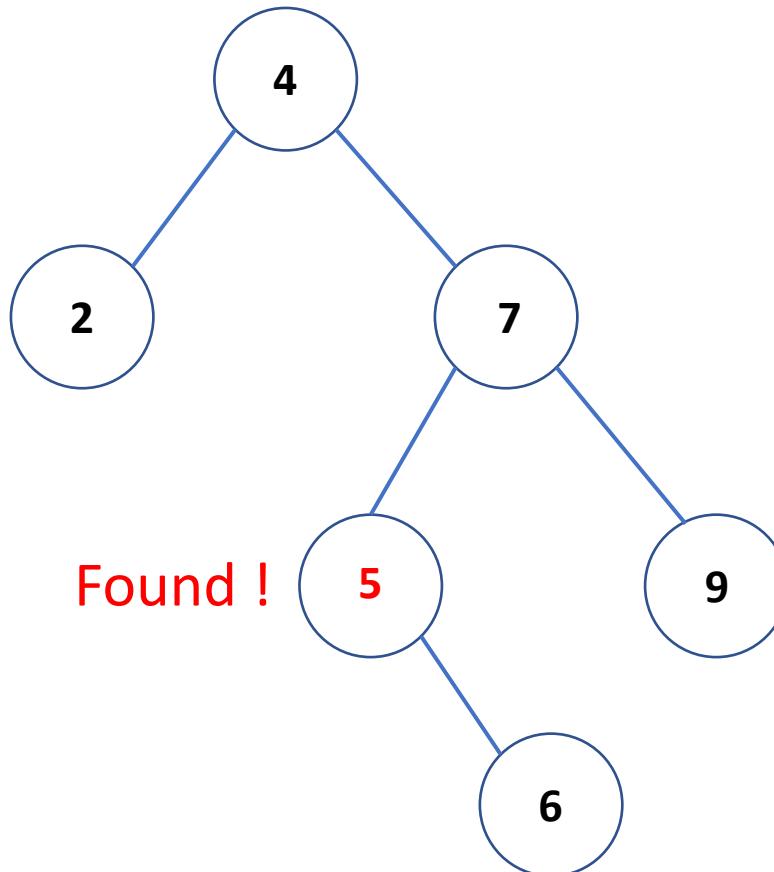
Search(T , 5)



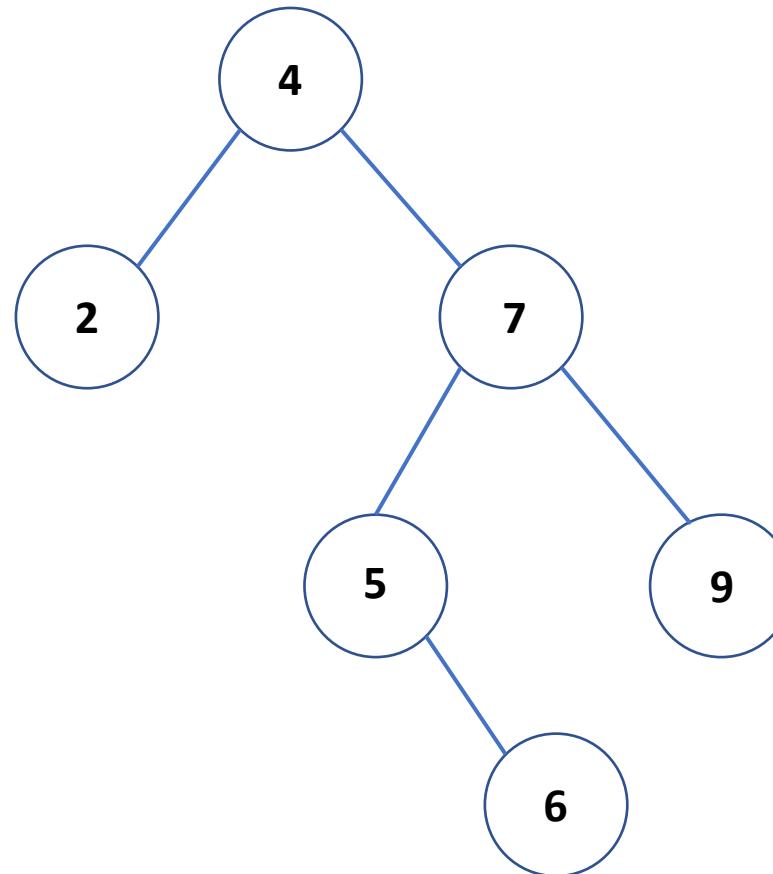
Search(T , 5)



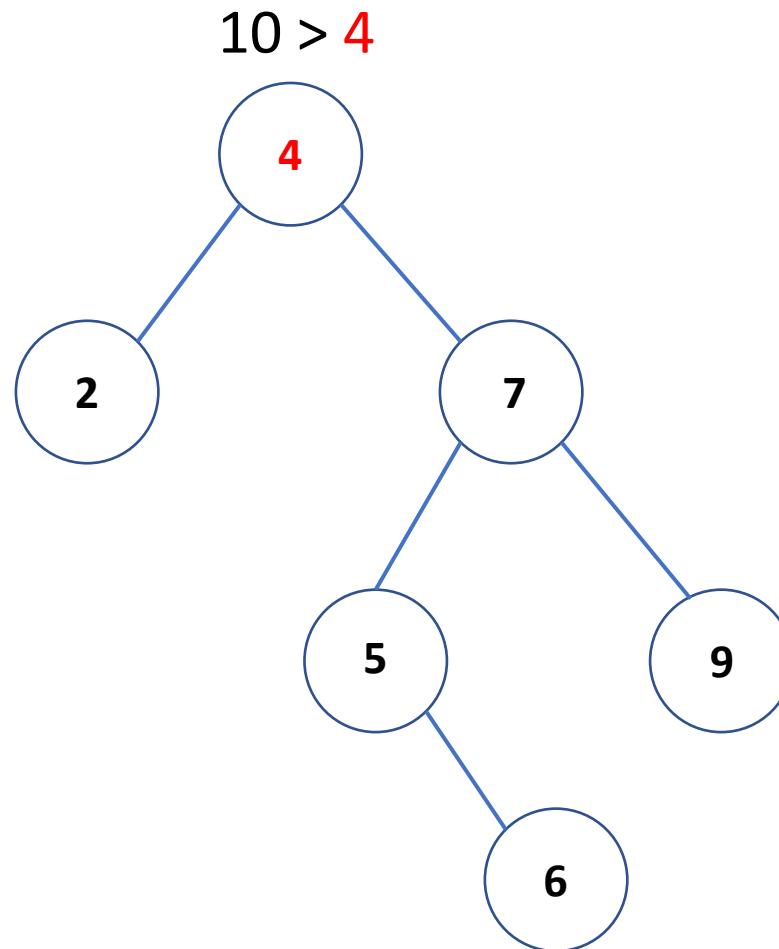
Search(T , 5)



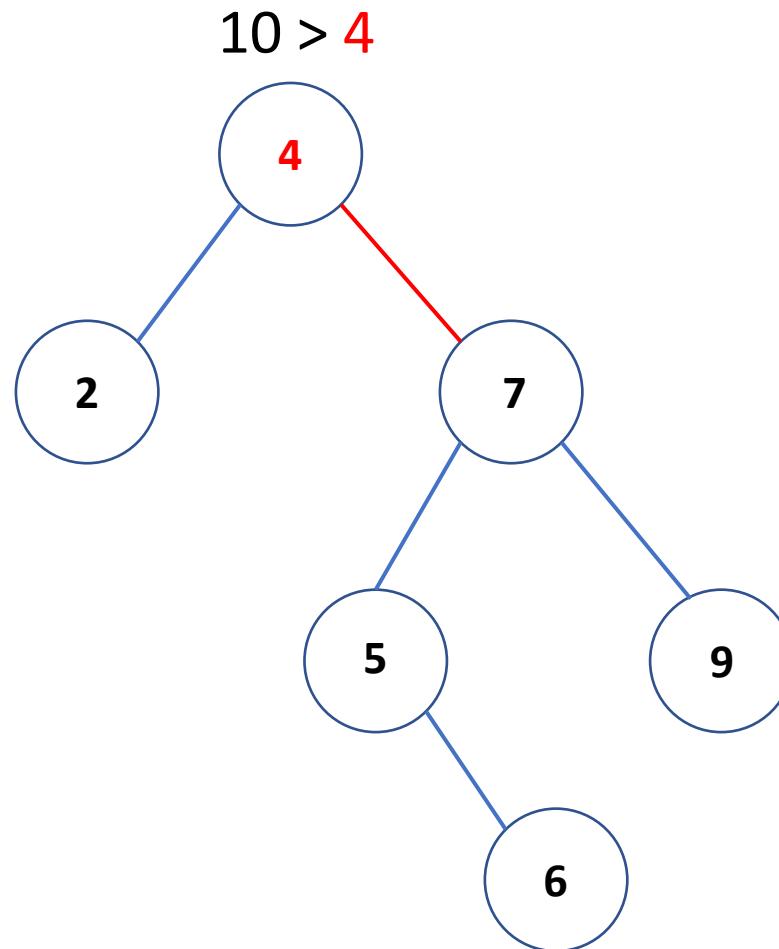
Insert(T , 10)



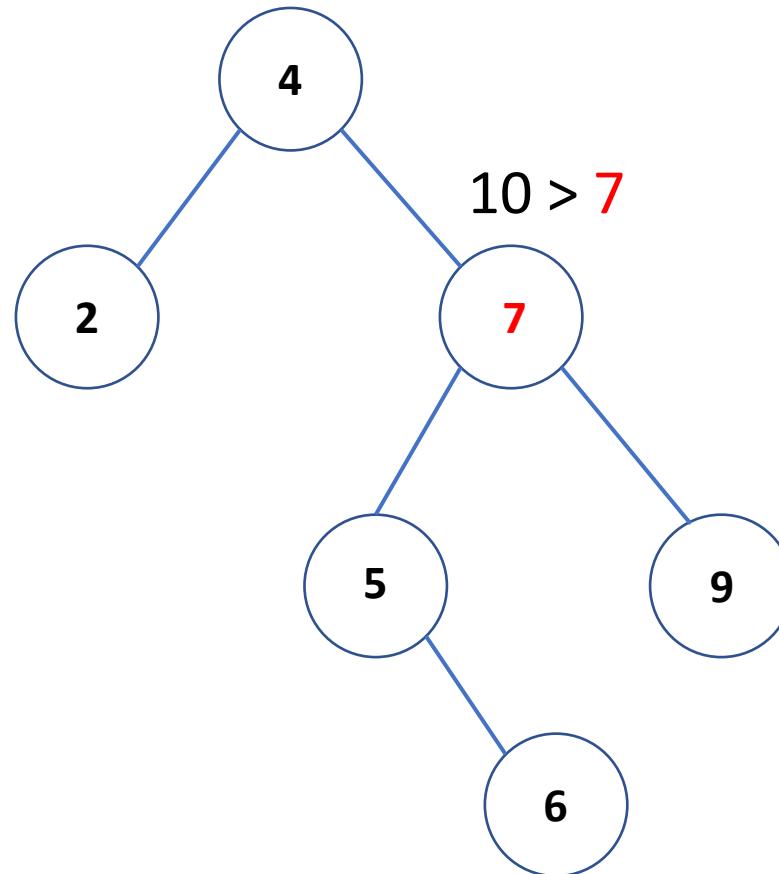
Insert(T , 10)



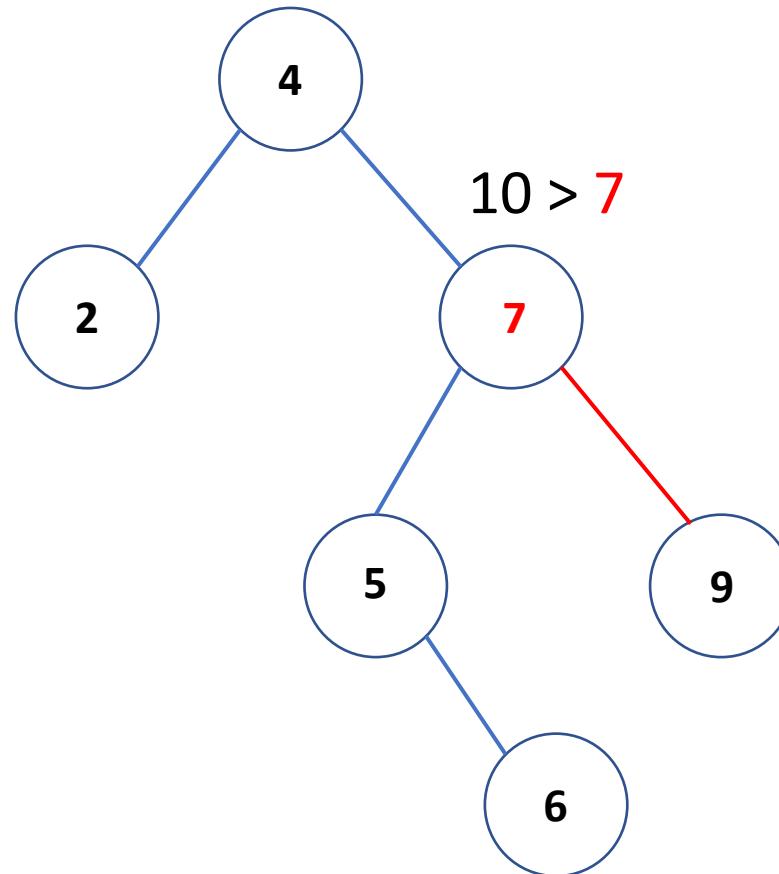
Insert(T , 10)



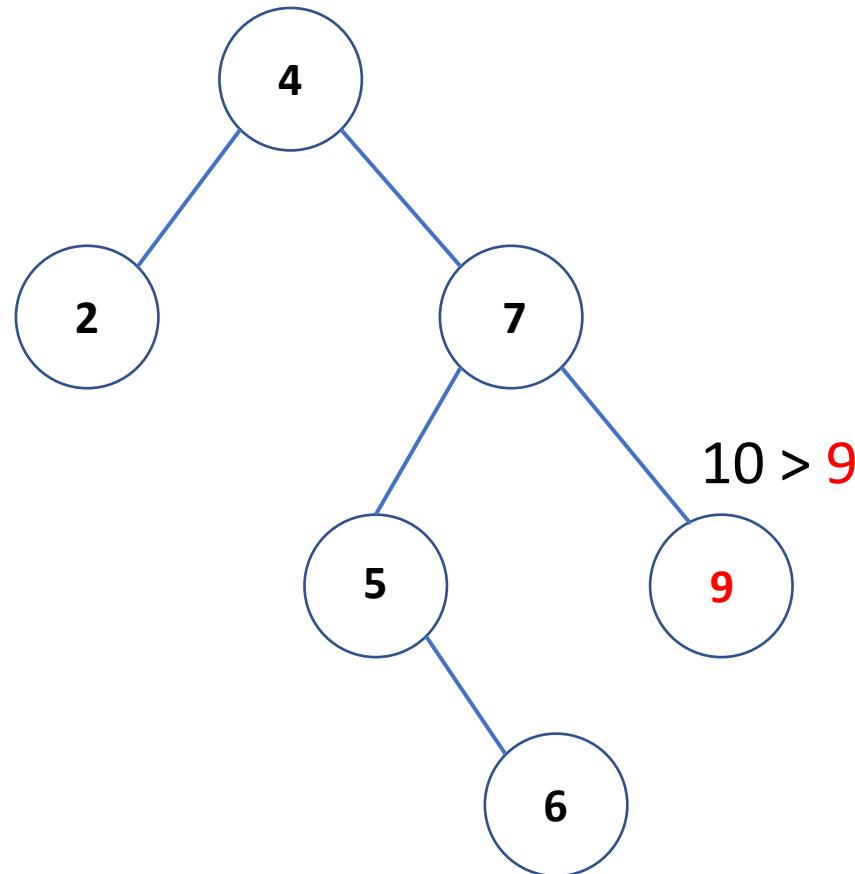
Insert(T , 10)



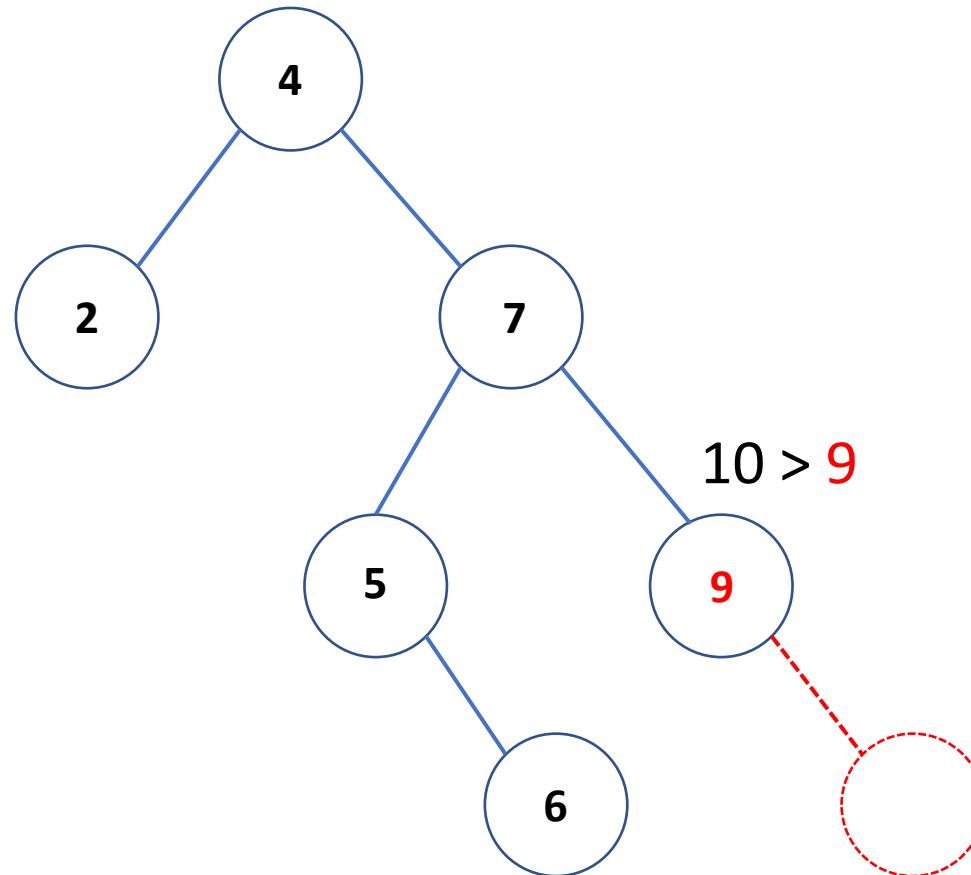
Insert(T , 10)



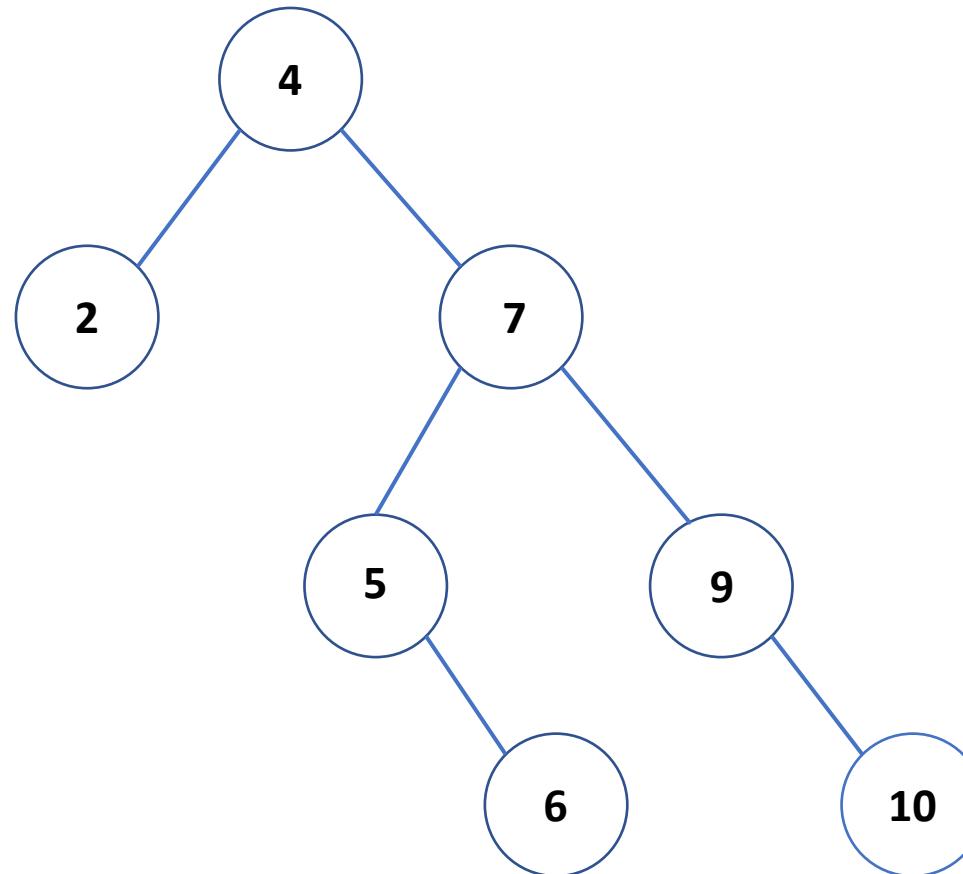
Insert(T , 10)



Insert(T , 10)



Insert(T , 10)

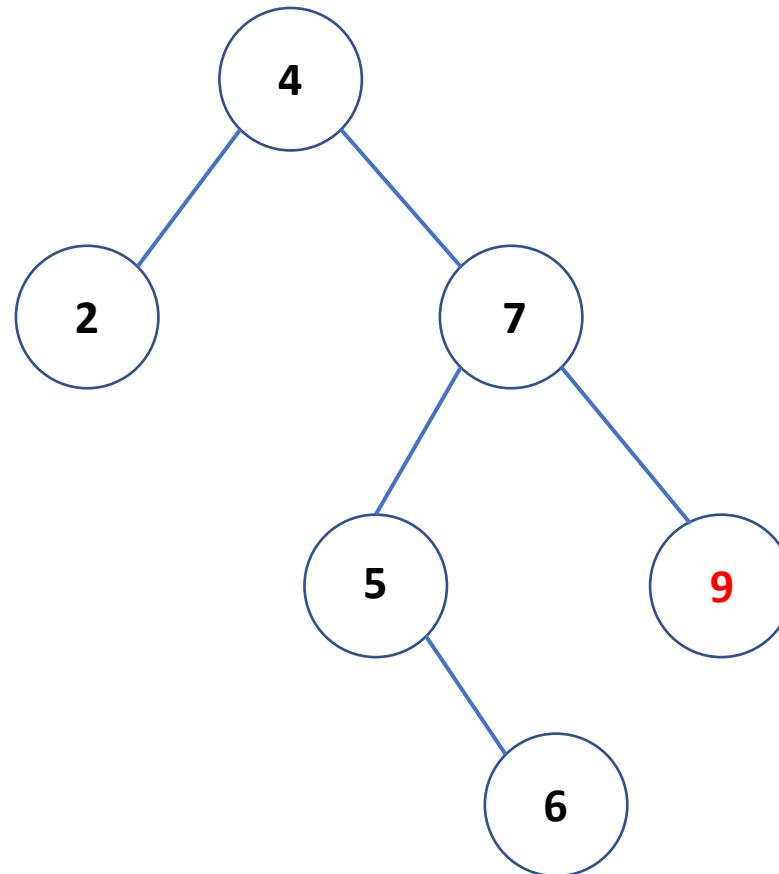


Delete(T, x)

Several possible cases



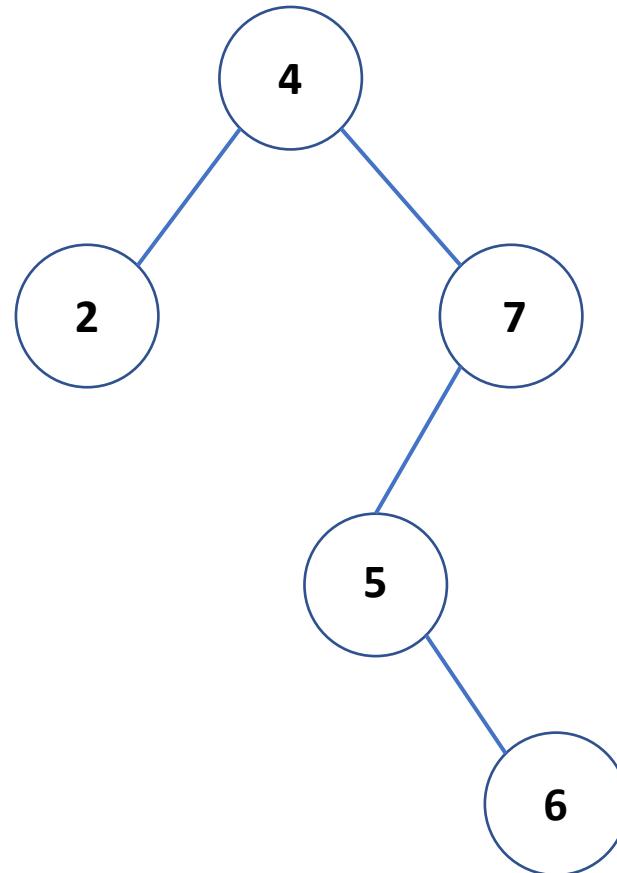
Delete(T , 9)



If node to be deleted is a leaf



Delete(T , 9)

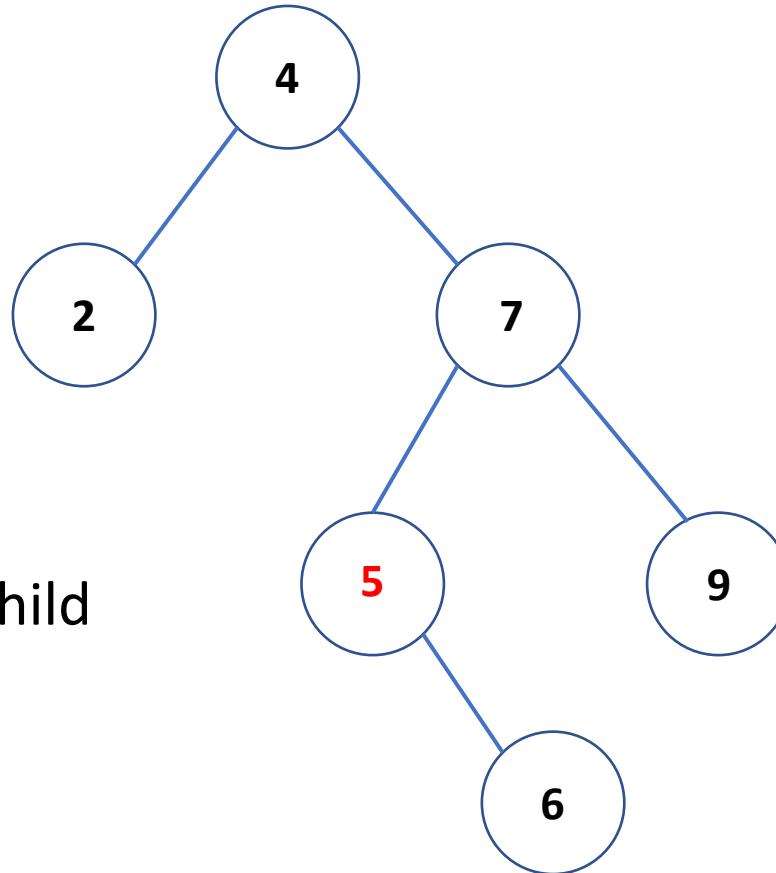


If node to be deleted is a leaf
remove the leaf

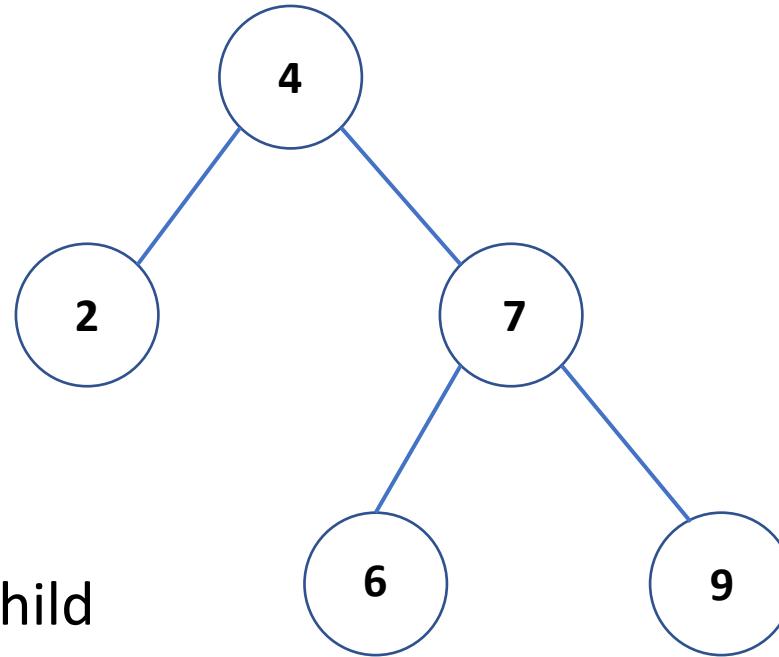


Delete($T, 5$)

If node to be deleted has one child



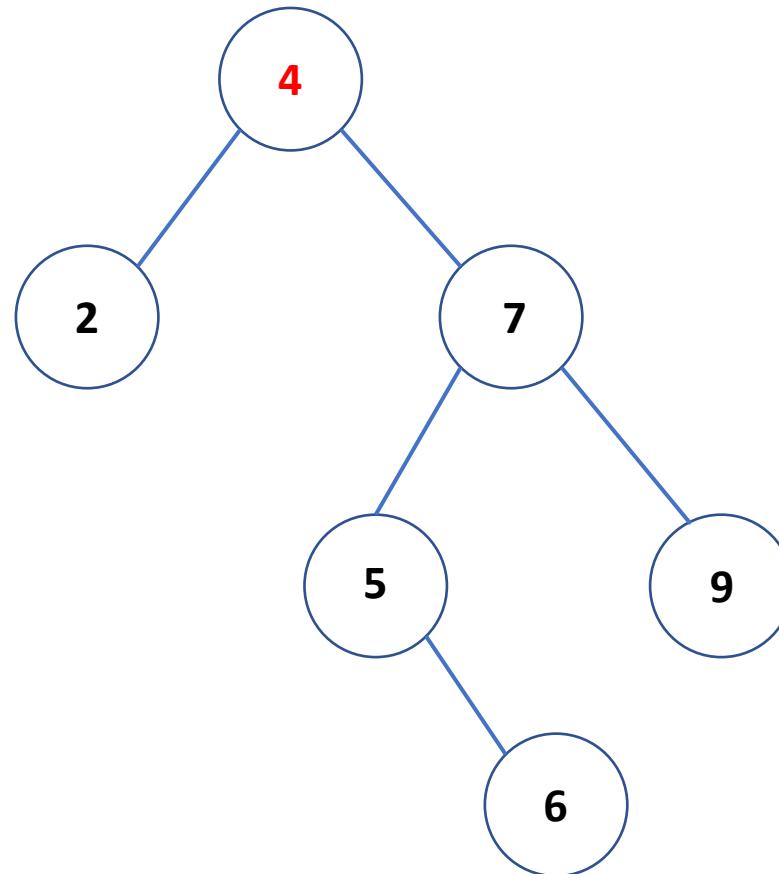
Delete($T, 5$)



If node to be deleted has one child
replace the node with child (and its subtree)

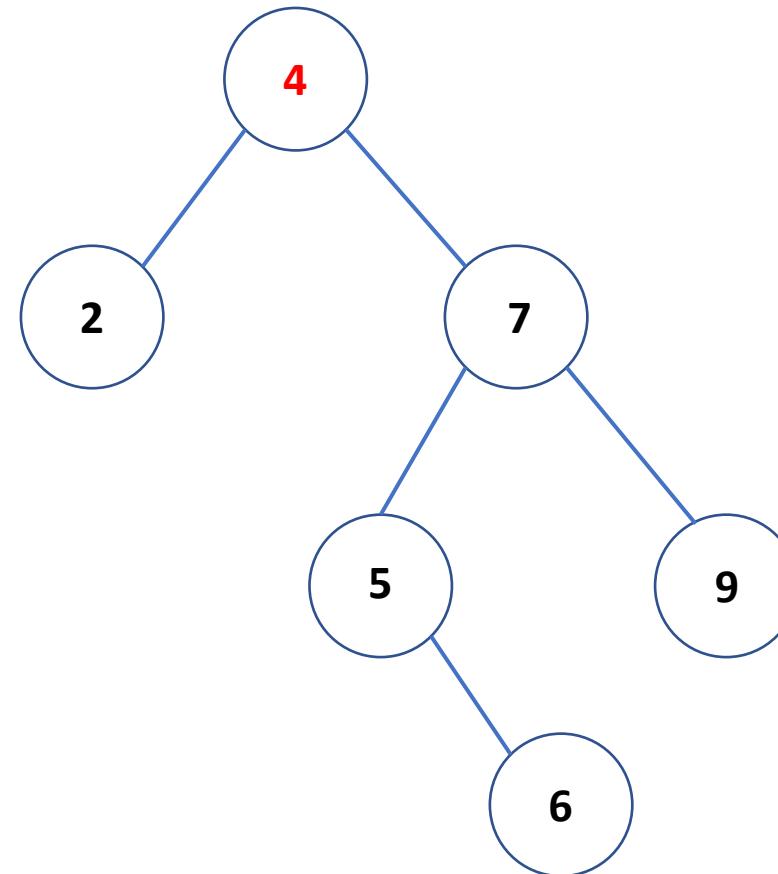


Delete(T , 4)



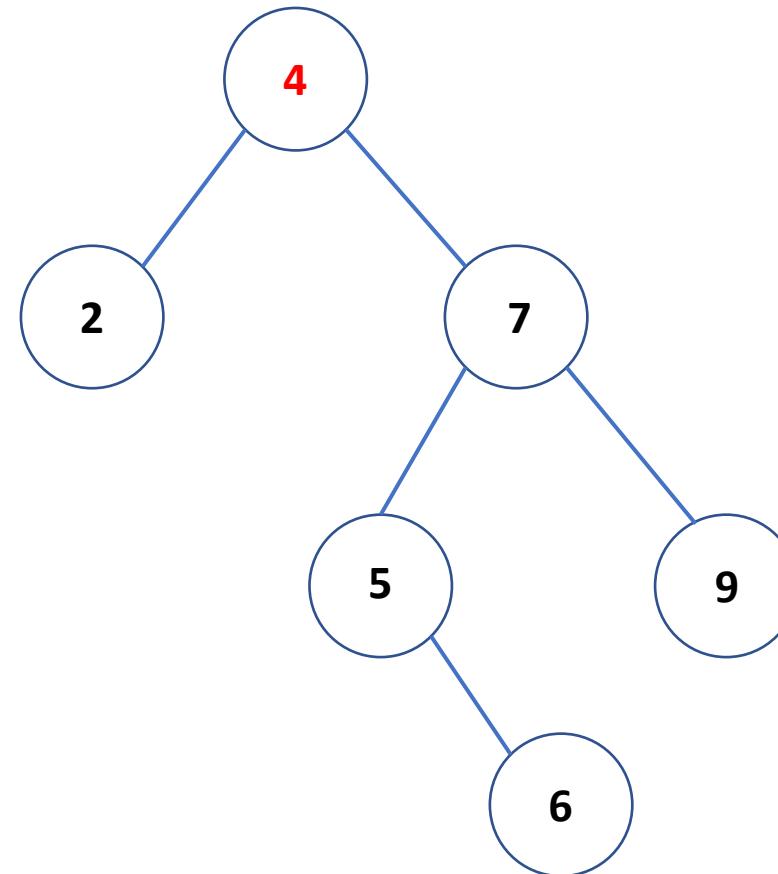
Delete(T , 4)

If node to be deleted has two children:



Delete(T , 4)

If node to be deleted has two children:



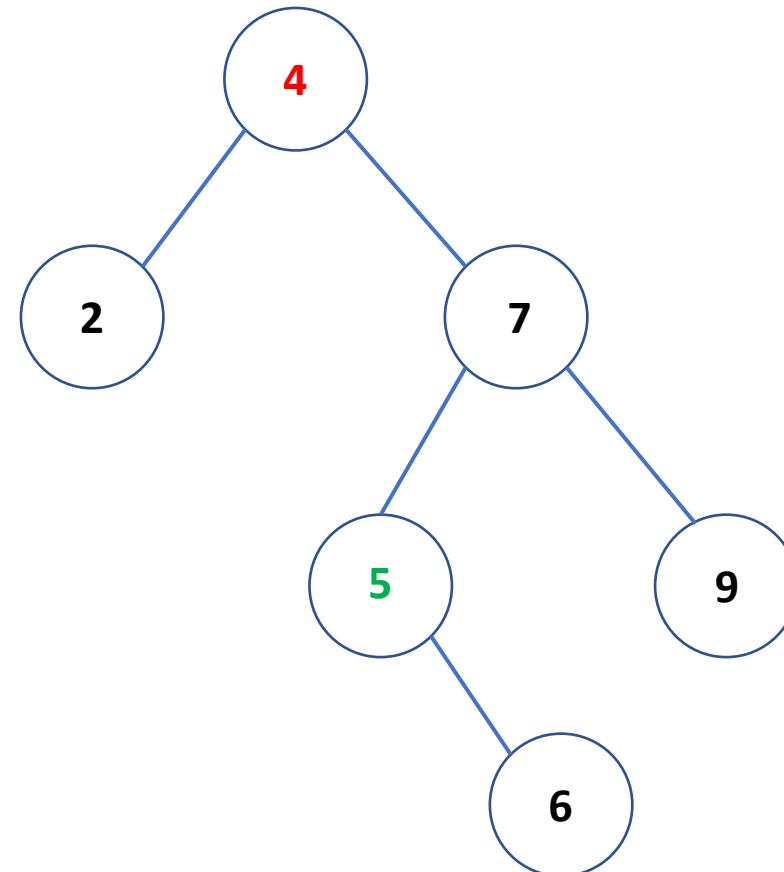
S : 2 4 5 6 7 9

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted
on the internet without the written permission of the copyright owners.



Delete(T , 4)

If node to be deleted has two children:



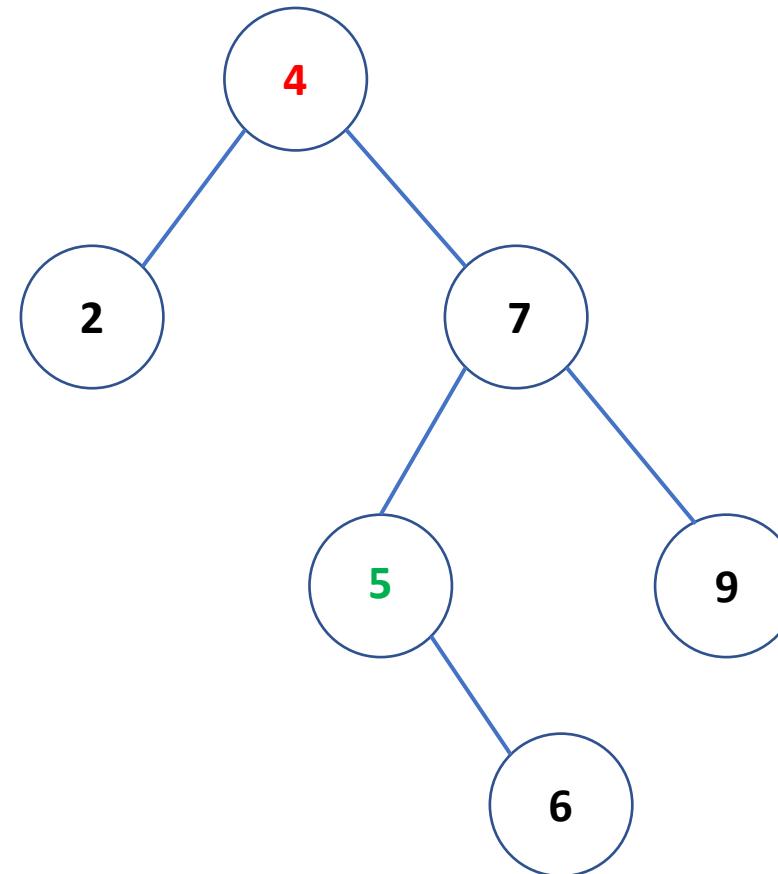
S : 2 4 5 6 7 9



Delete(T , 4)

If node to be deleted has two children:

- Find the “successor” of the node



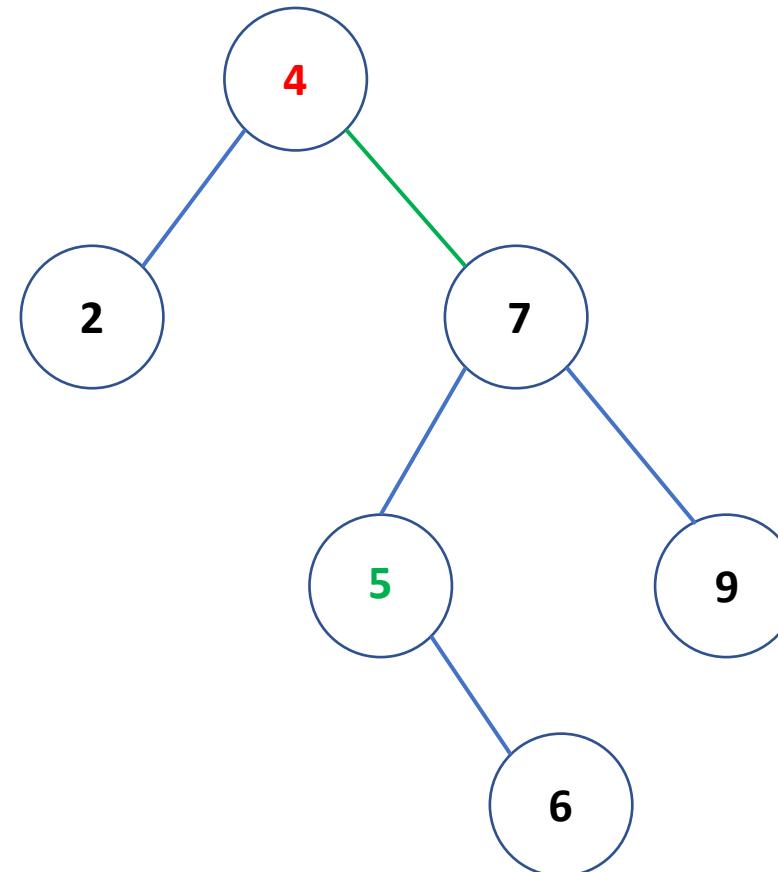
S : 2 4 5 6 7 9

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Delete(T , 4)

- If node to be deleted has two children:
- Find the “successor” of the node
 - Go one step down right

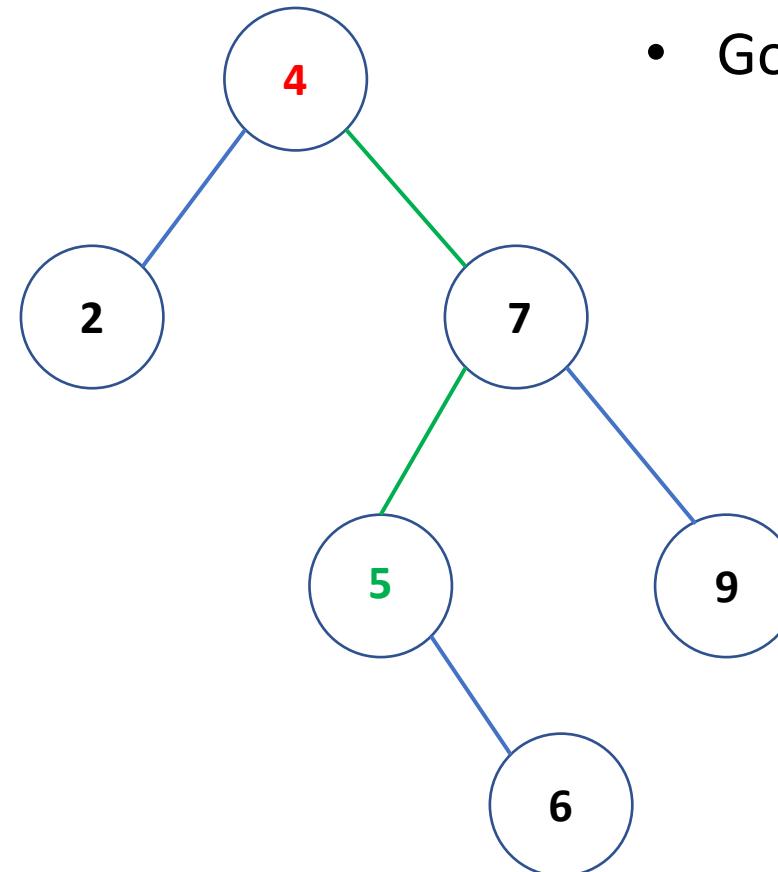


S : 2 4 5 6 7 9



Delete(T , 4)

- If node to be deleted has two children:
- Find the “successor” of the node
 - Go one step down right
 - Go all the way down to the left



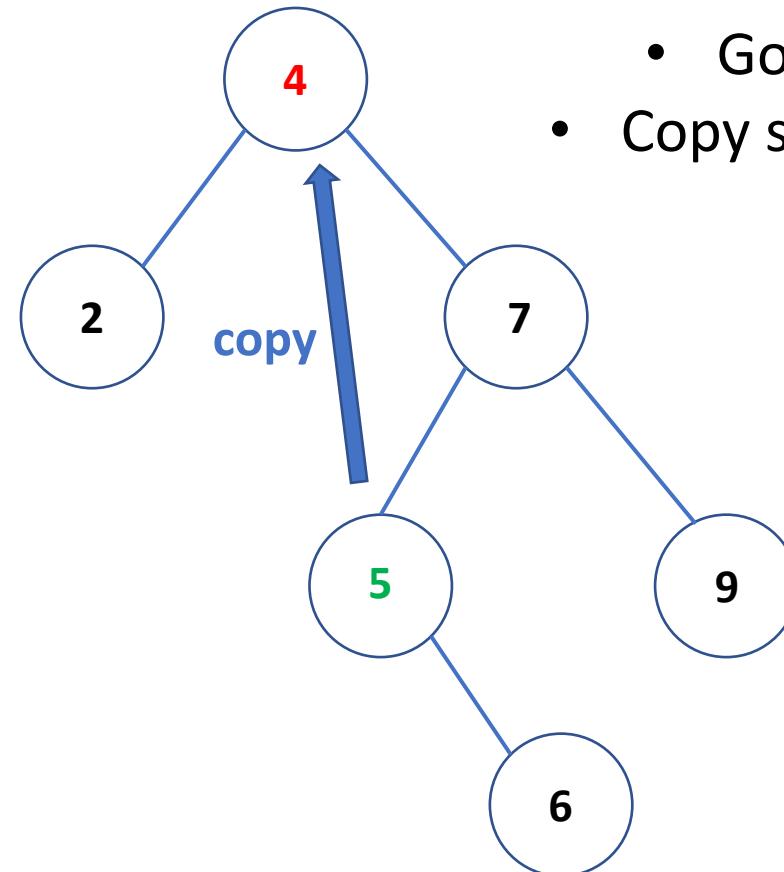
S : 2 4 5 6 7 9



Delete(T , 4)

If node to be deleted has two children:

- Find the “successor” of the node
 - Go one step down right
 - Go all the way down to the left
- Copy successor’s key into node’s key



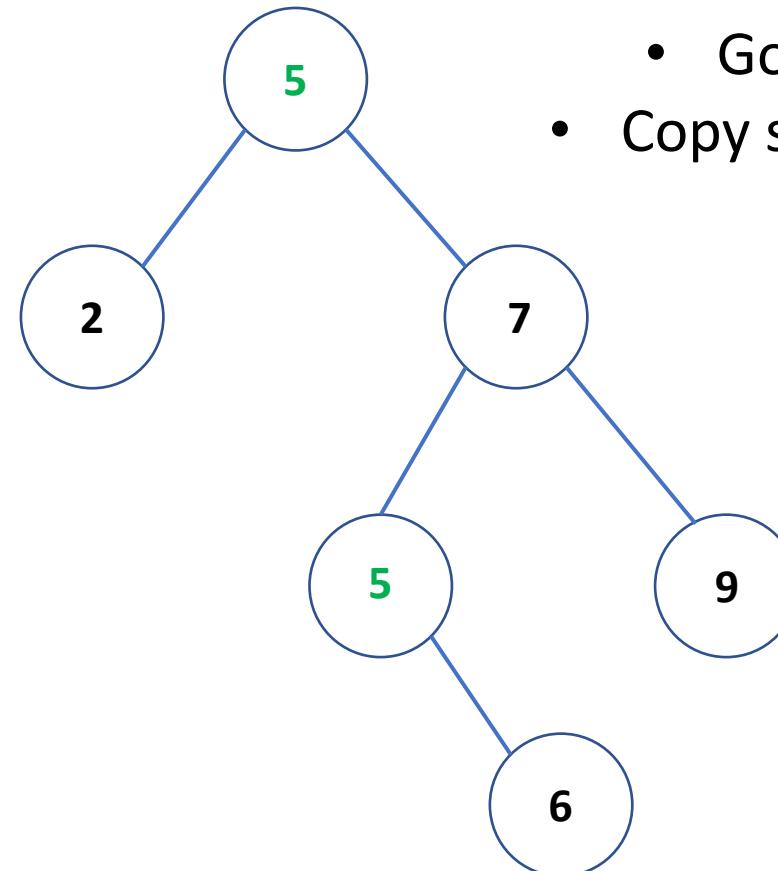
S : 2 4 5 6 7 9



Delete(T , 4)

If node to be deleted has two children:

- Find the “successor” of the node
 - Go one step down right
 - Go all the way down to the left
- Copy successor’s key into node’s key



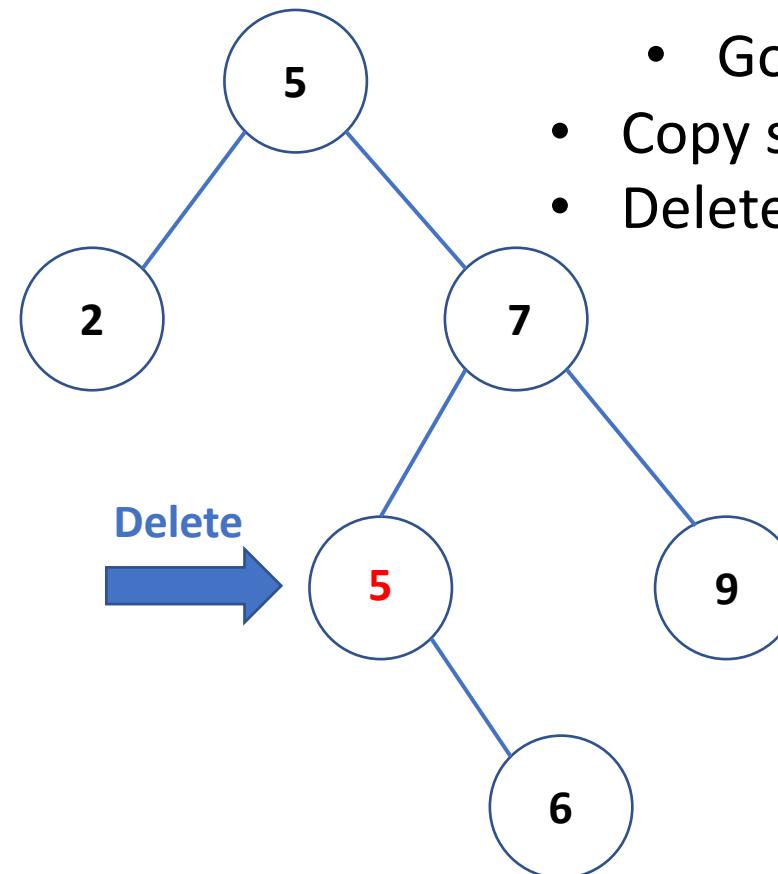
S : 2 5 5 6 7 9



Delete(T , 4)

If node to be deleted has two children:

- Find the “successor” of the node
 - Go one step down right
 - Go all the way down to the left
- Copy successor’s key into node’s key
- Delete successor node



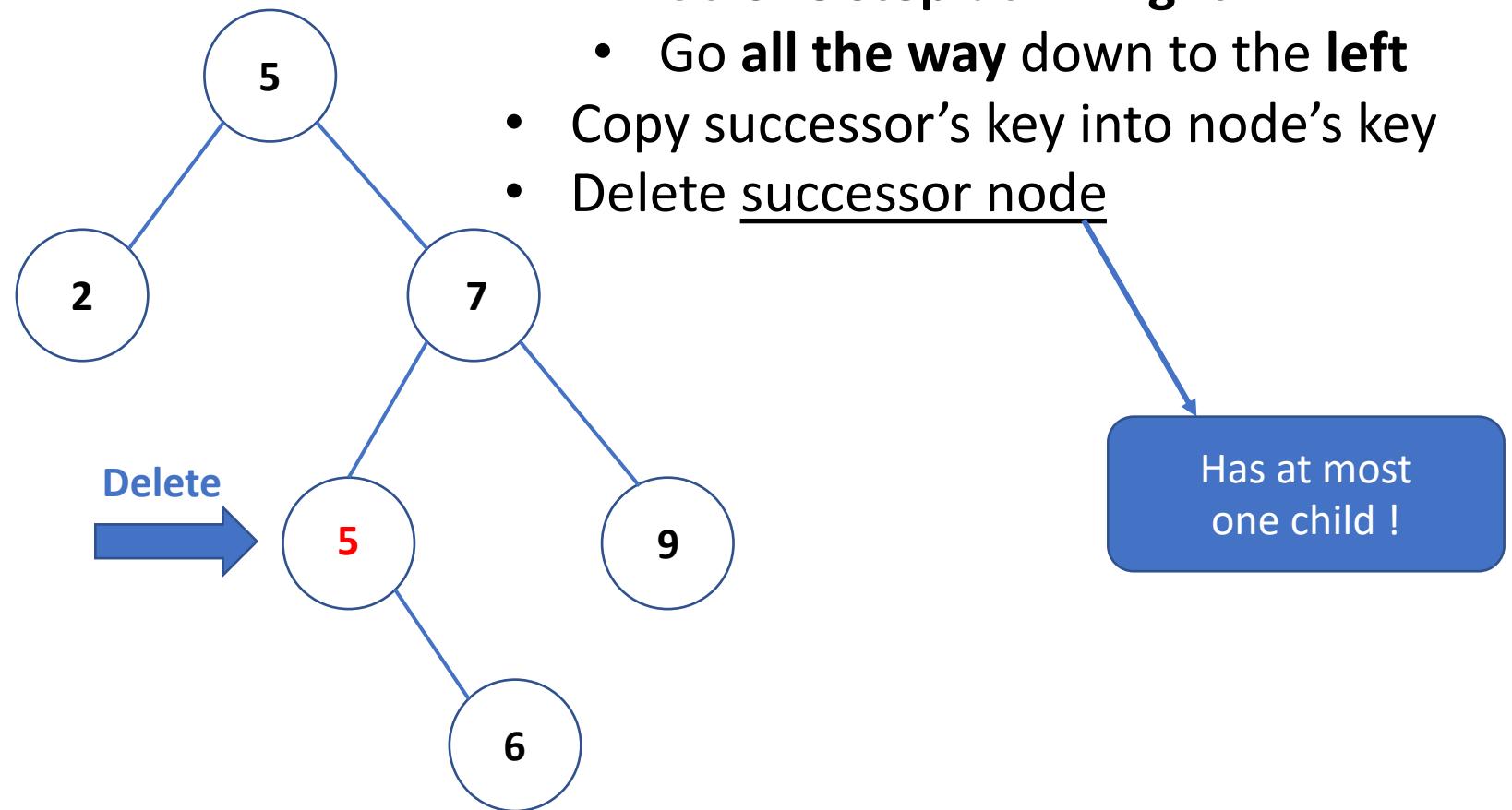
S : 2 5 5 6 7 9



Delete(T, 4)

If node to be deleted has two children:

- Find the “successor” of the node
 - Go **one step** down **right**
 - Go **all the way** down to the **left**
 - Copy successor’s key into node’s key
 - Delete successor node

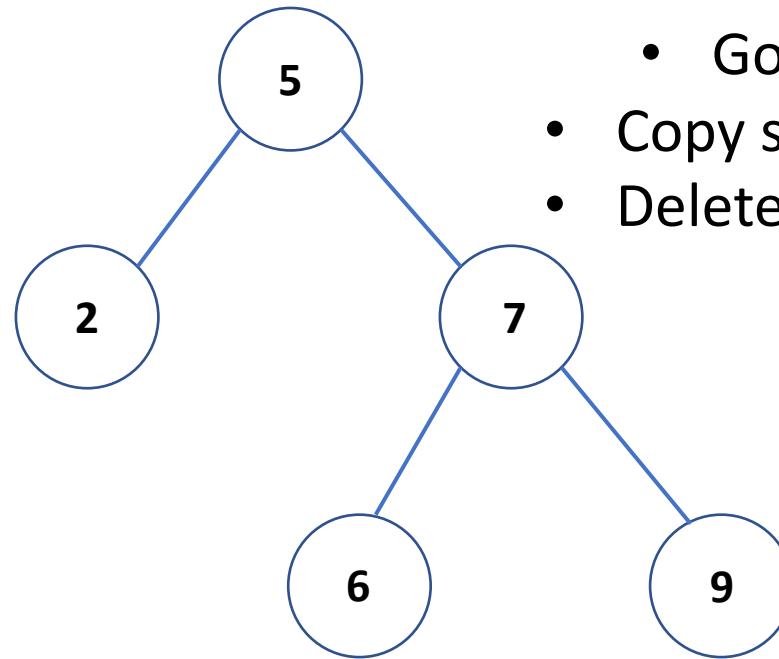


S : 2 5 5 6 7 9

© 2019 by Jay Balasundaram and Sam Toueg. This document may not be posted on the internet without the written permission of the copyright owners.



Delete(T , 4)



- If node to be deleted has two children:
- Find the “successor” of the node
 - Go **one step down right**
 - Go **all the way** down to the **left**
 - Copy successor’s key into node’s key
 - Delete successor node

Has at most
one child !

S : 2 5 6 7 9



Delete(T , x)

- Case 1: **x is a leaf** : Remove x
- Case 2: **x has one child** : Replace x with child and its subtree
- Case 3: **x has two children** :
 - $y \leftarrow \text{Successor}(x)$ [Leftmost child in the right subtree of x]
 - Replace x with y
 - Delete **successor node containing y** (using Case 1 or 2)



Has at most
one child !



Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\quad)$?



Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$



Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?



Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?

1



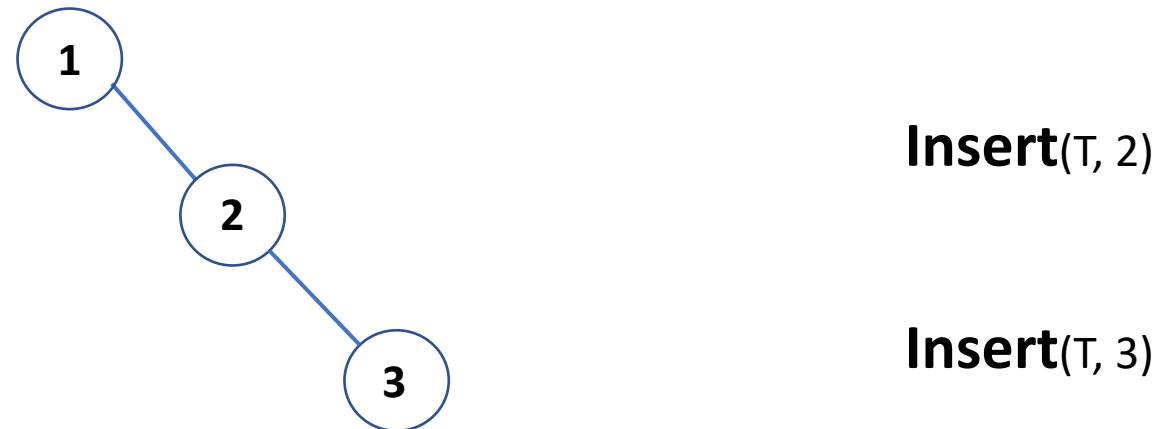
Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?



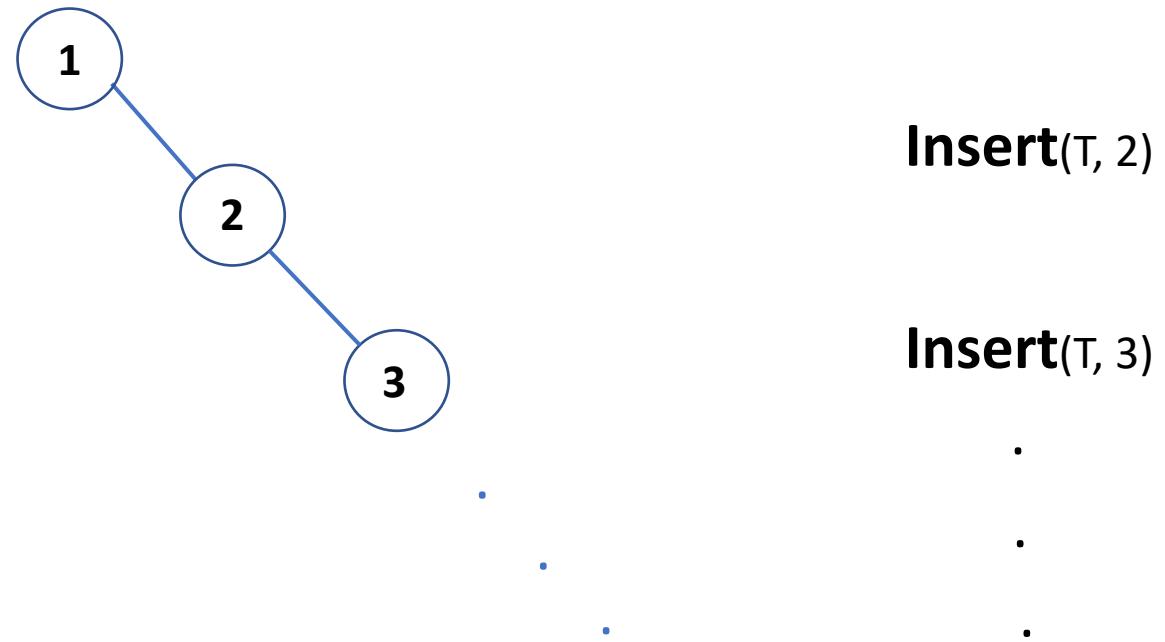
Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?



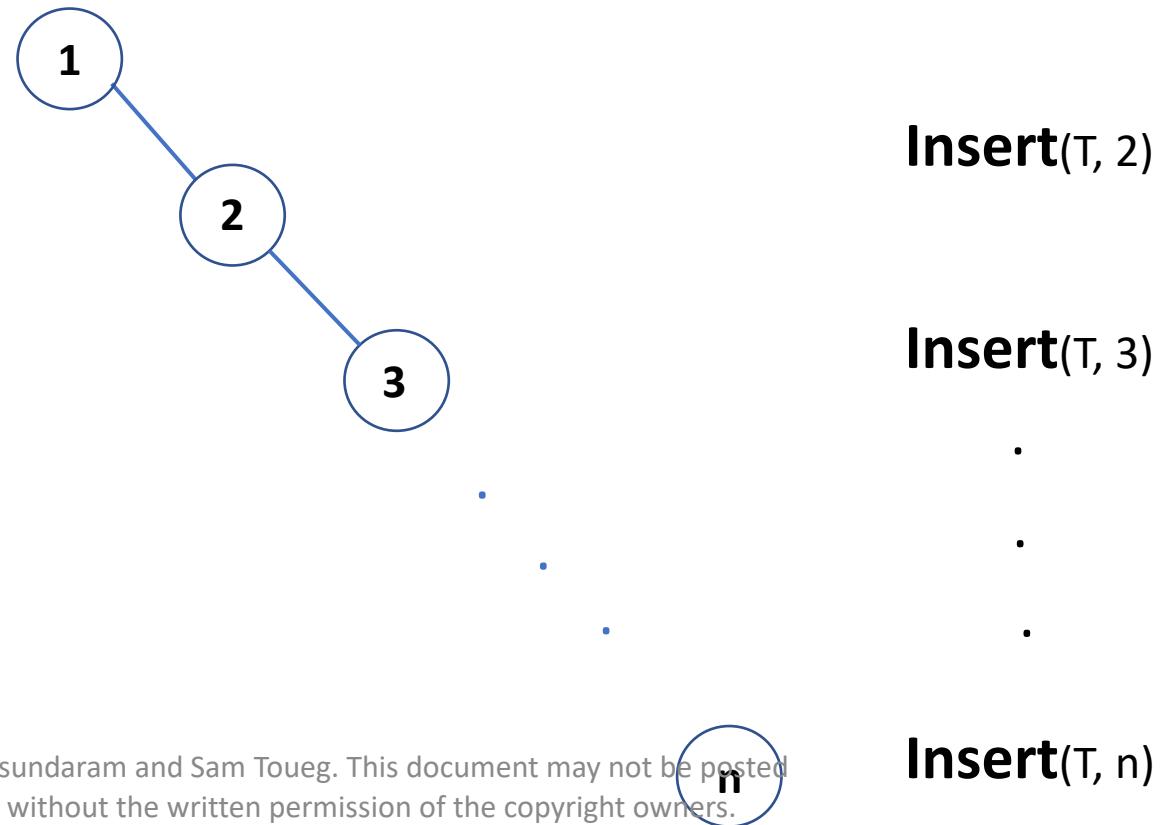
Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?



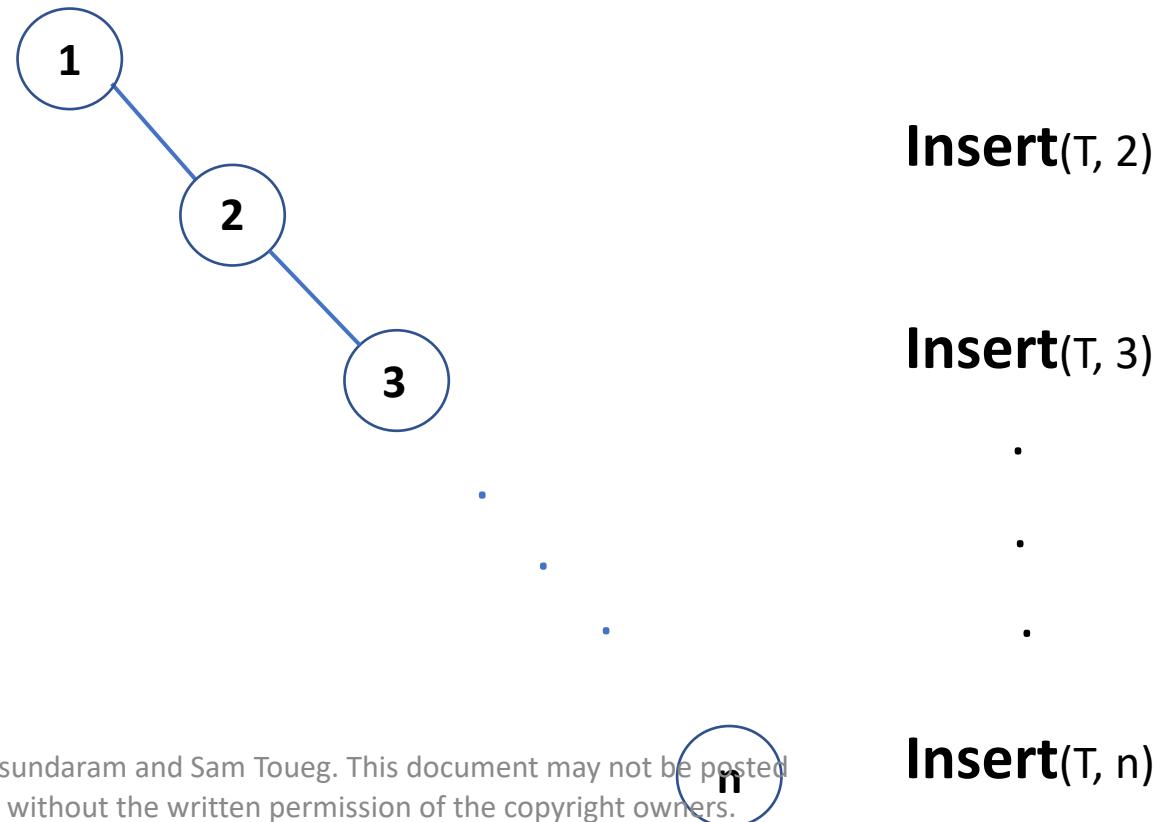
Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes?



Worst-Case Time Complexity of Search/Insert/Delete

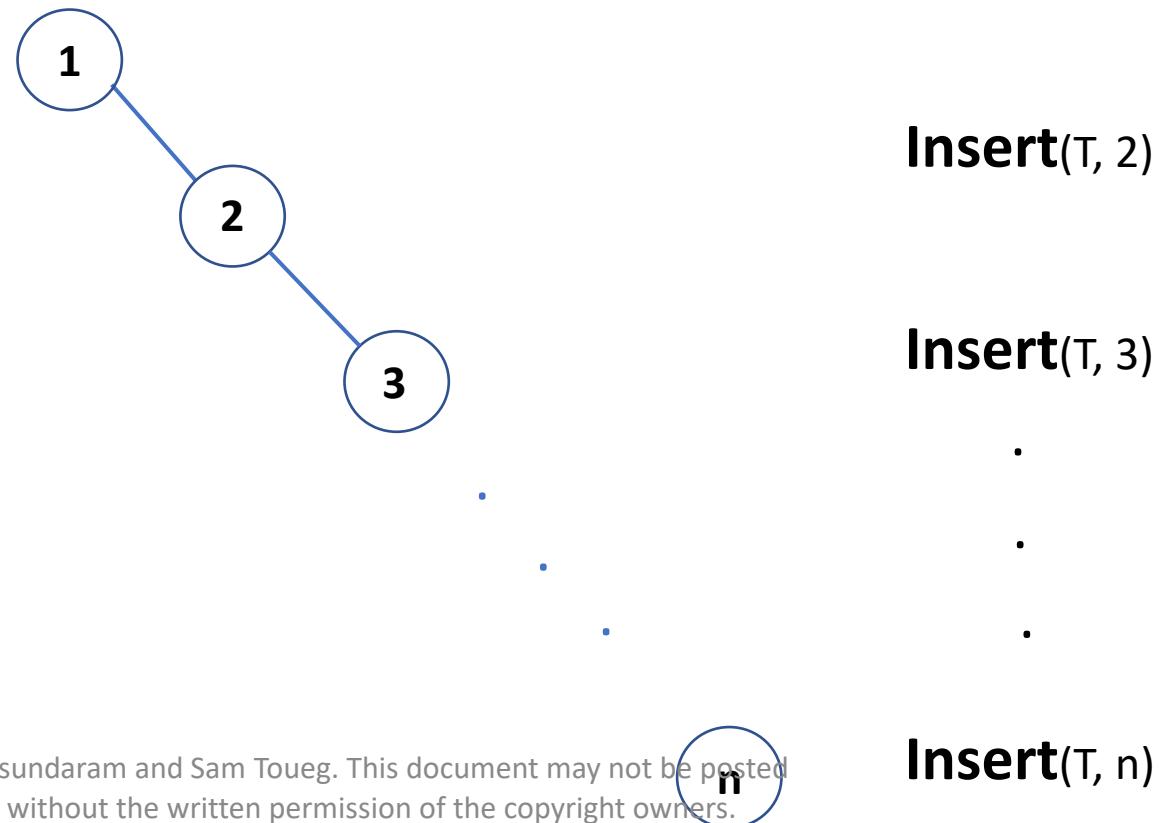
- Worst-Case time complexity of each operation is $\Theta(\text{height of tree})$
- Maximum height of a BST with n nodes? $n-1$



Worst-Case Time Complexity of Search/Insert/Delete

- Worst-Case time complexity of each operation is $\Theta(n)$
- Maximum height of a BST with n nodes? $n-1$

BSTs are not very good
in the worst case!



Next Week...

Variant of BST that ensures maximum height is $O(\log n)$!

