Binary Search Trees

Instructor: Meng-Fen Chiang

COMPCSI220: WEEK 9

THE UNIVERSITY OF AUCKLAND
Te Whare Wananga o Tamaki Makaurau
NEW ZEALAND

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz and Tanya Gvozdeva

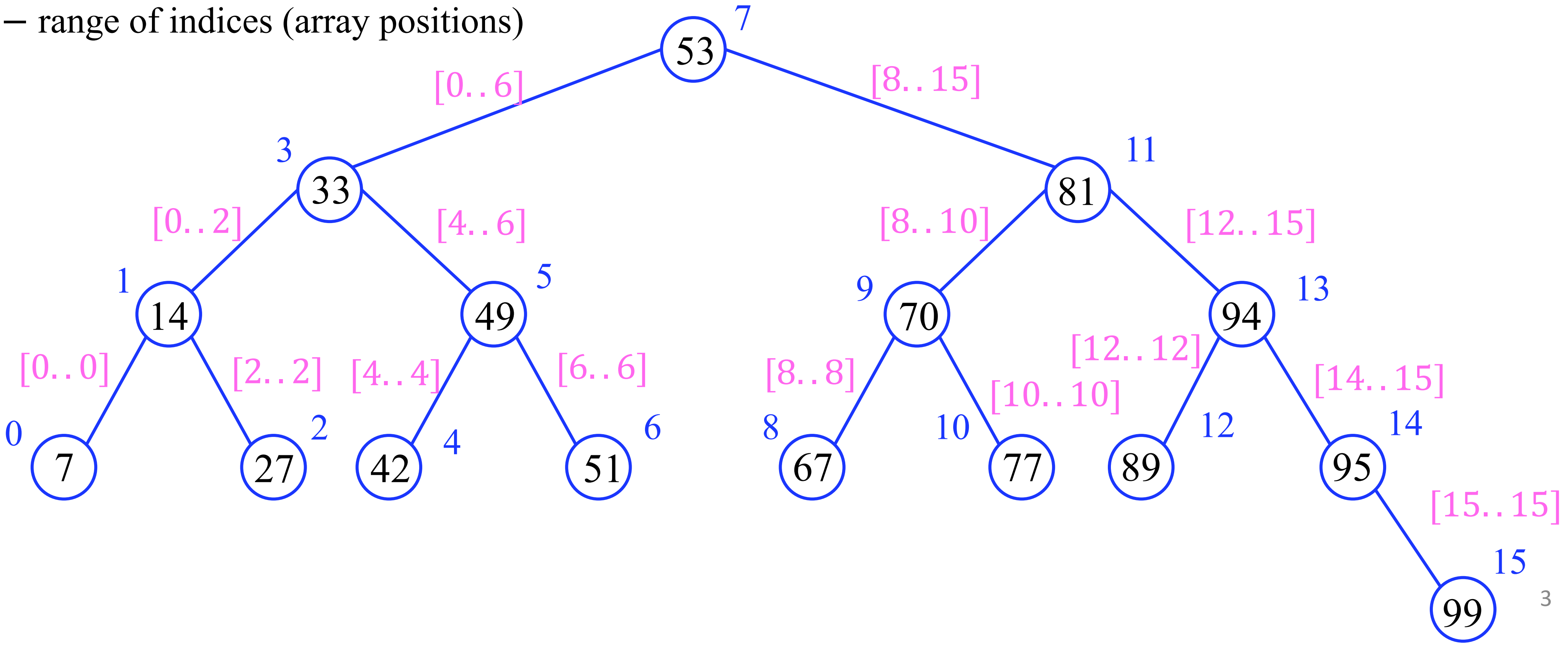
OUTLINE

- Tree Data Structure
- Binary Search Tree Operations
- Time Complexity Analysis

Tree Structure of Binary Search

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	14	27	33	42	49	51	53	67	70	77	81	89	94	95	99

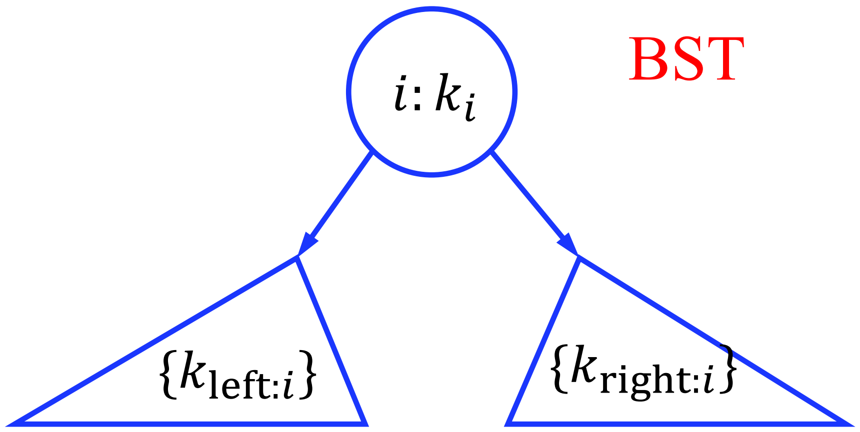
$[l..r]$ – range of indices (array positions)

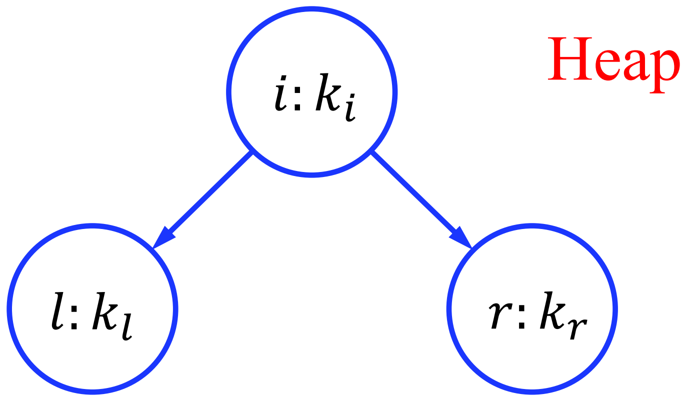


Binary Search Tree: Left-Right Ordering of Keys

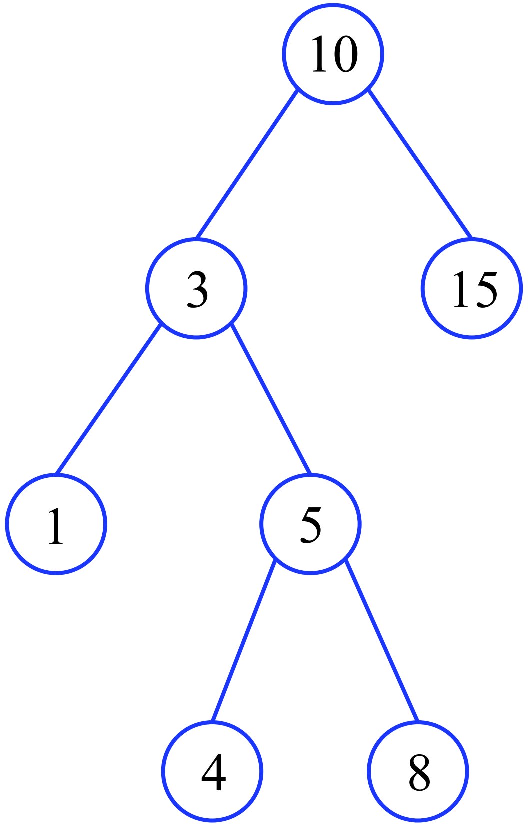
- Left-to-right numerical ordering in a BST: for every node i,
  - the values of all the keys $k_{\text{left}:i}$ in the left subtree are smaller than the key k_i in i and
  - the values of all the keys $k_{\text{right}:i}$ in the right subtree are larger than the key k_i in i:

$$\{k_{\text{left}:i}\} \ni l < k_i < r \in \{k_{\text{right}:i}\}$$

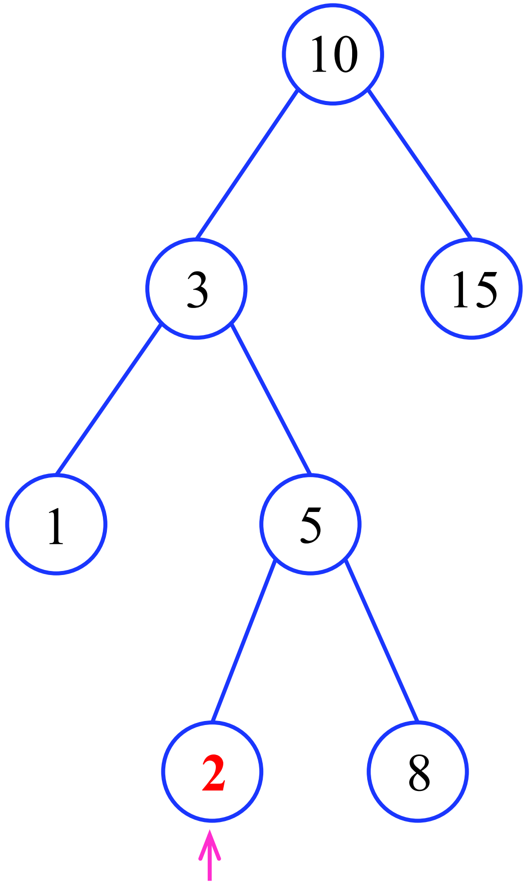




Binary Search Tree: Left-Right Ordering of Keys

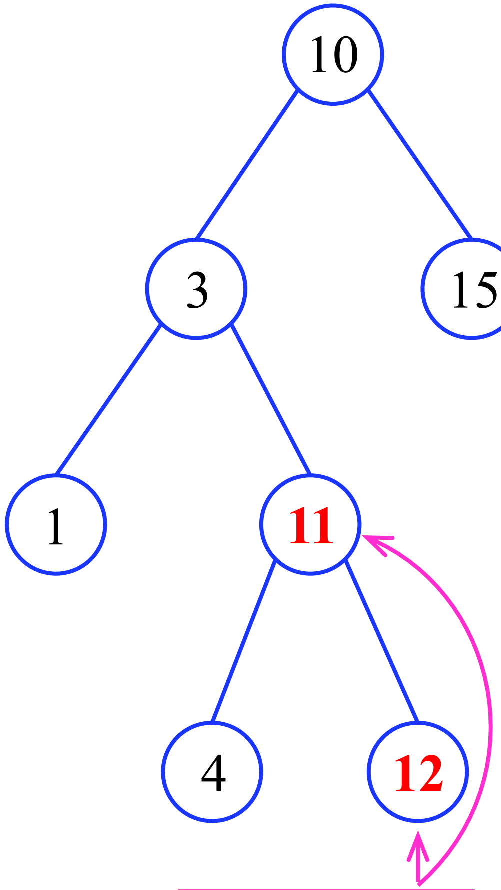


BST



Non-BST:

Key "2" cannot be in the right subtree of key "3".



Non-BST:

Key "11" and "12" cannot be in the left subtree of key "10".

Binary Search Tree Operations

- BST is an explicit data structure implementing the table ADT.
  - BST are more complex than heaps: any node may be removed, not only a root or leaves.
  - The only practical constraint: no duplicate keys (attach them all to a single node).

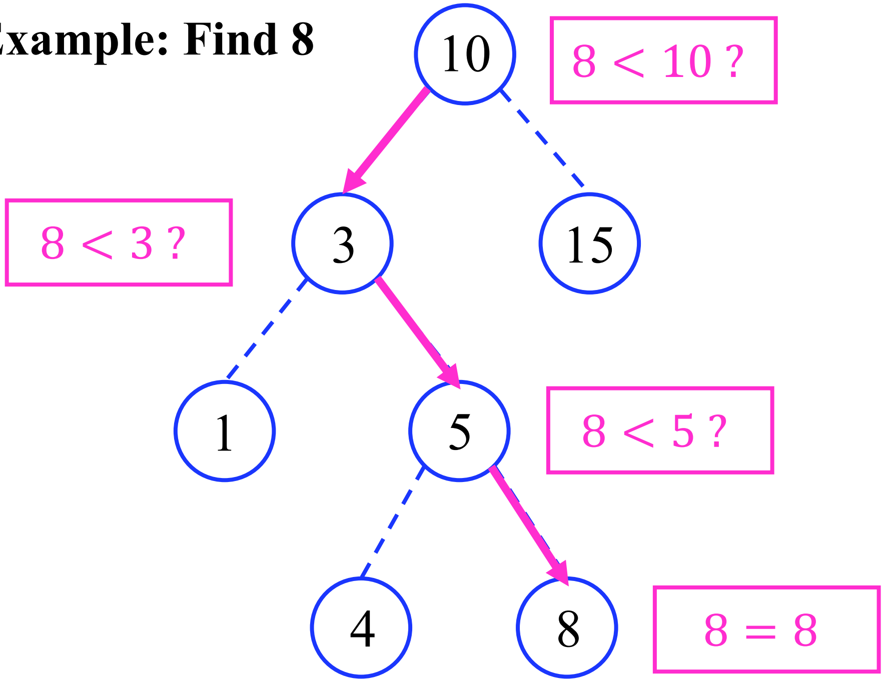
- Basic operations
  - Find a given search key or detect that it is absent in the BST.
  - Insert a node with a given key to the BST if it is not found.
  - FindMin: find the minimum key.
  - findMax: find the maximum key.
  - Remove a node with a given key and restore the BST if necessary.

BST Operations: Find / Insert a Node

find: a successful binary search

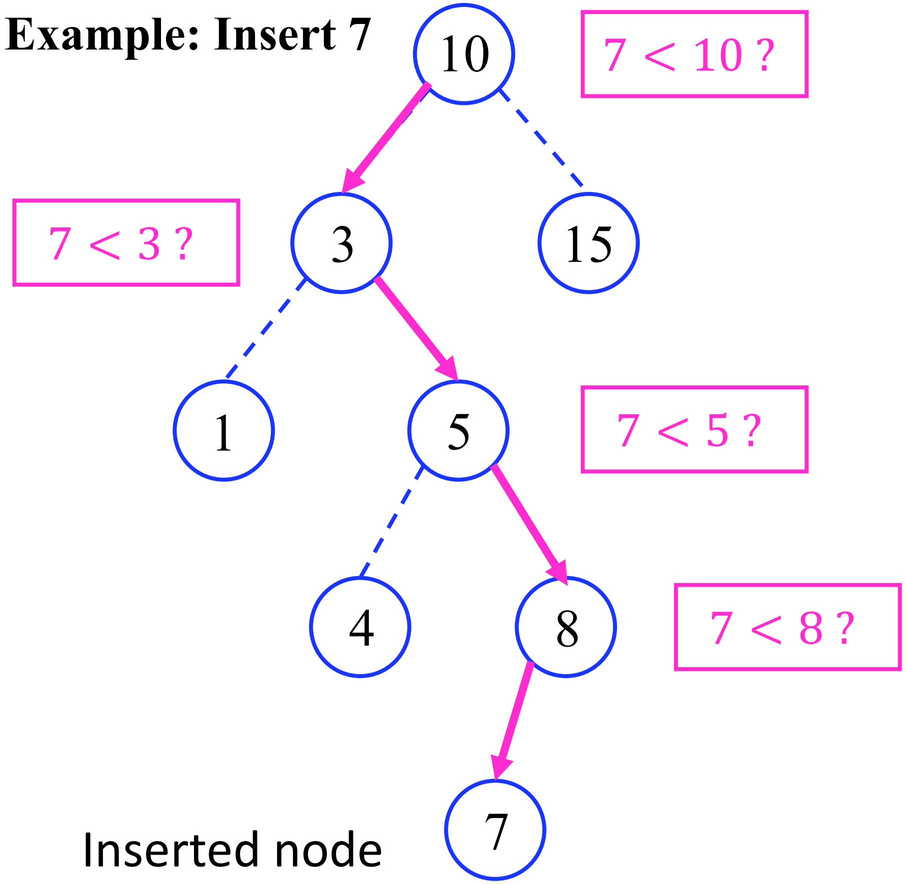
insert: creating a new node at the point where an unsuccessful search stops.

Example: Find 8



Found node

Example: Insert 7



Inserted node

BST Operations: FindMin / FindMax

- Extremely simple: starting at the root, branch repeatedly left (findMin) or right (findMax) as long as a corresponding child exists.

- The root of the tree plays a role of the pivot in quicksort and quickselect.
- As in quicksort, the in-order traversal of the tree can sort the items:
  - First visit the left subtree;
  - Then visit the root, and
  - Then visit the right subtree.

- $O(\log n)$ average-case and $O(n)$ worst-case running time for find, insert, findMin, and findMax operations, as well as for selecting a single item

BST Operations: Remove a Node

- The most complex because the tree may be disconnected. Need to reconnect some nodes!
  - Reconnection must retain the ordering condition.
  - Reconnection should not needlessly increase the tree height.

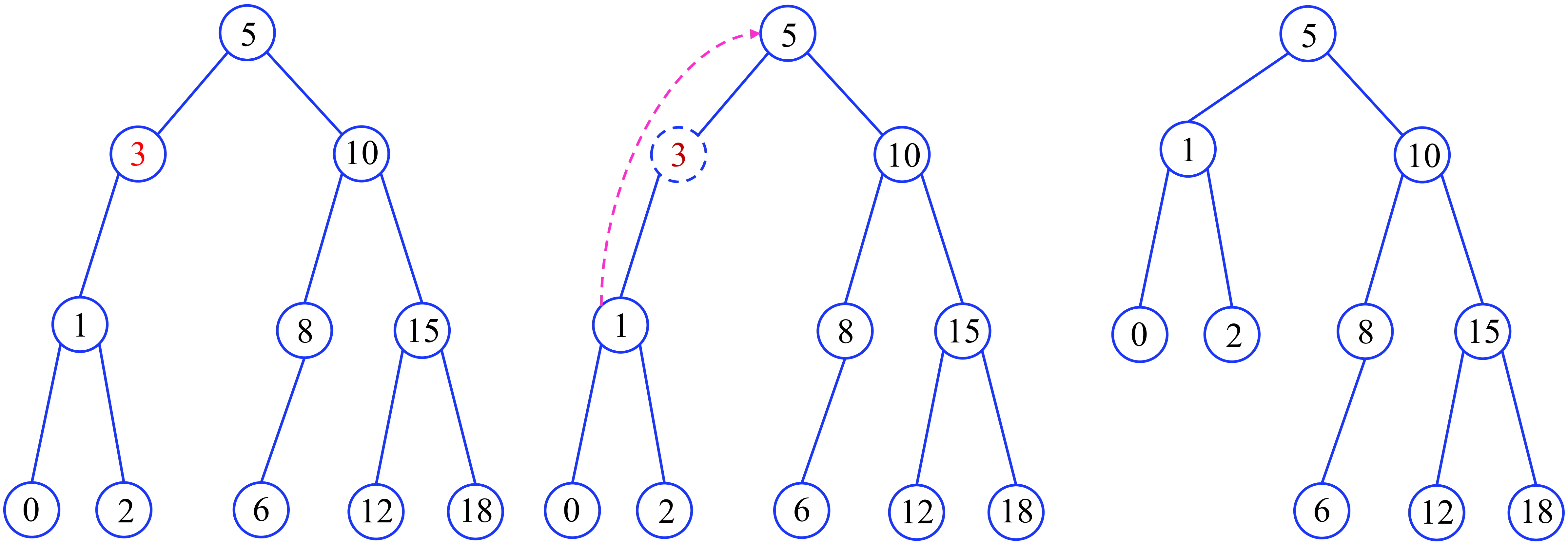
BST Operations: Remove a Node

- Standard method of removing a node i with c children:

c	ACTION
0	Simply remove the leaf i.
1	Remove the node i after linking its child to its parent node.
2	Swap the node i with the node j having the smallest key k_j in the right subtree of the node i. After swapping, remove the node i (as now it has at most one right child).

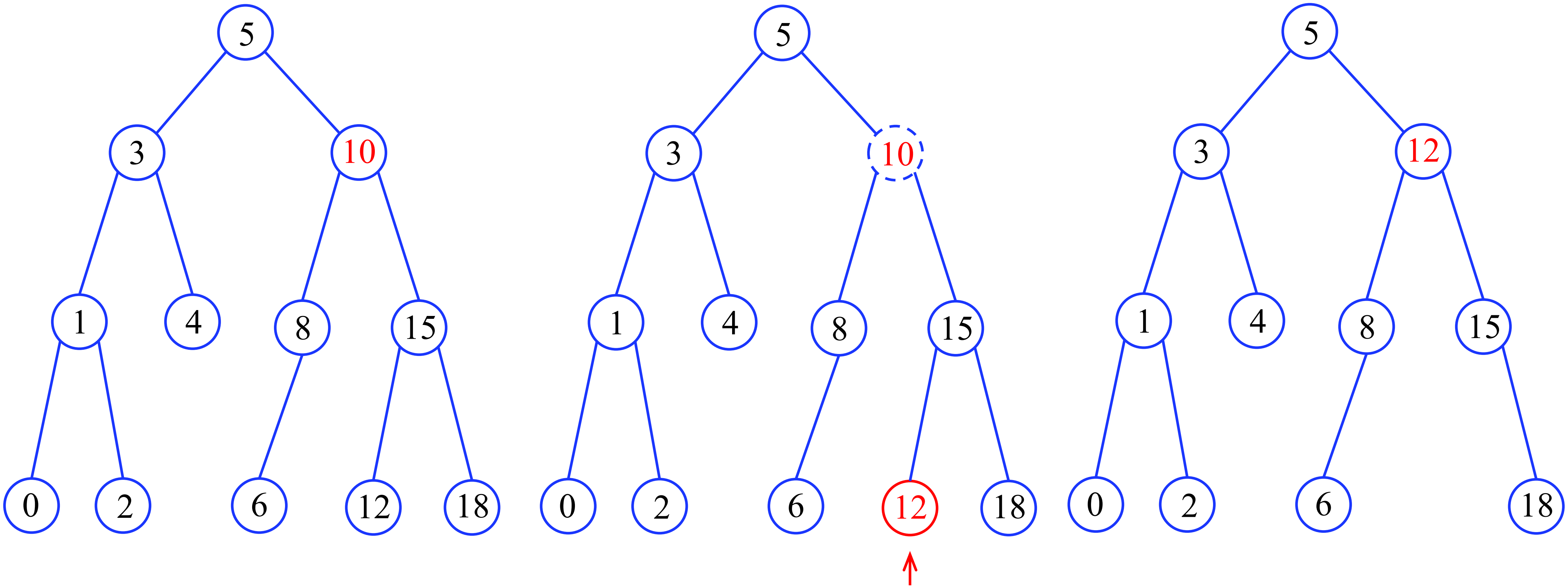
BST Operation: Remove a Node

Remove 3 $\Rightarrow$ Link 1 to the left branch of 5



BST Operation: Remove a Node

Remove 10 ⇒ Replace 10 (swap with 12 and delete)



Minimum key in the right subtree

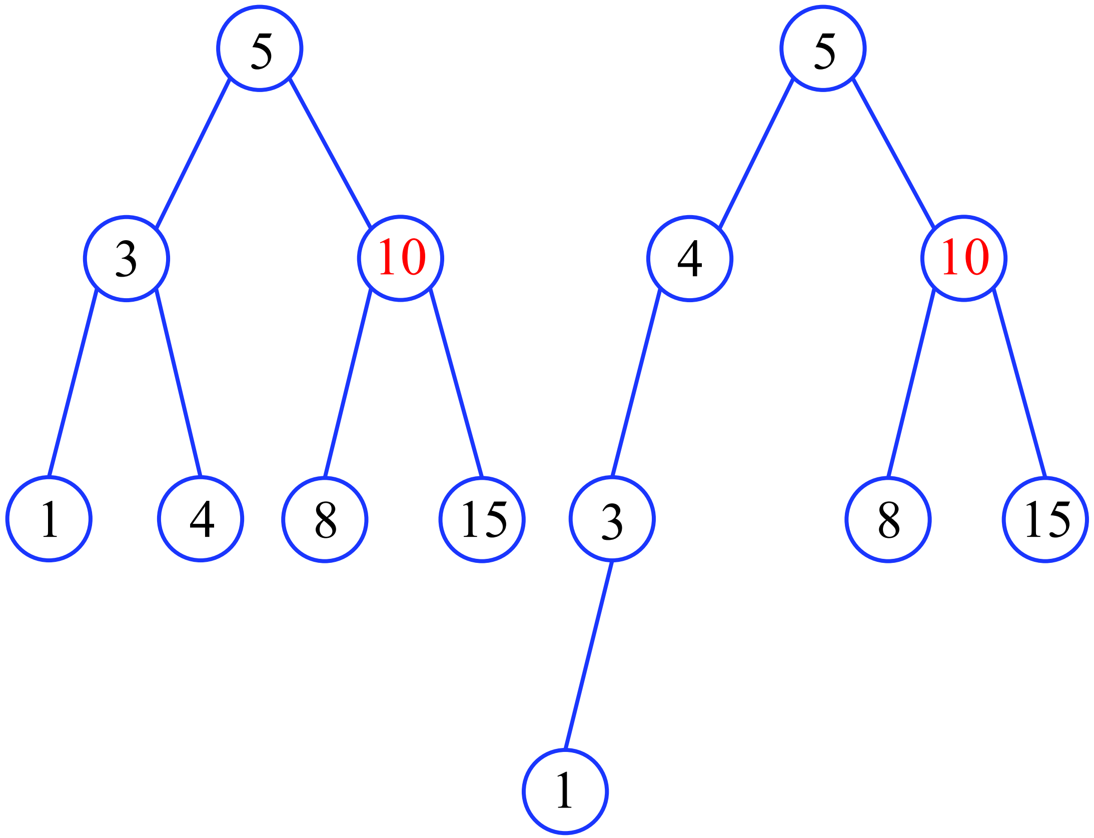
The Worst-Case Time Complexity

- The find, insert, and remove operations in a BST all take time in $O(h)$ in the worst case, where h is the height of the tree.

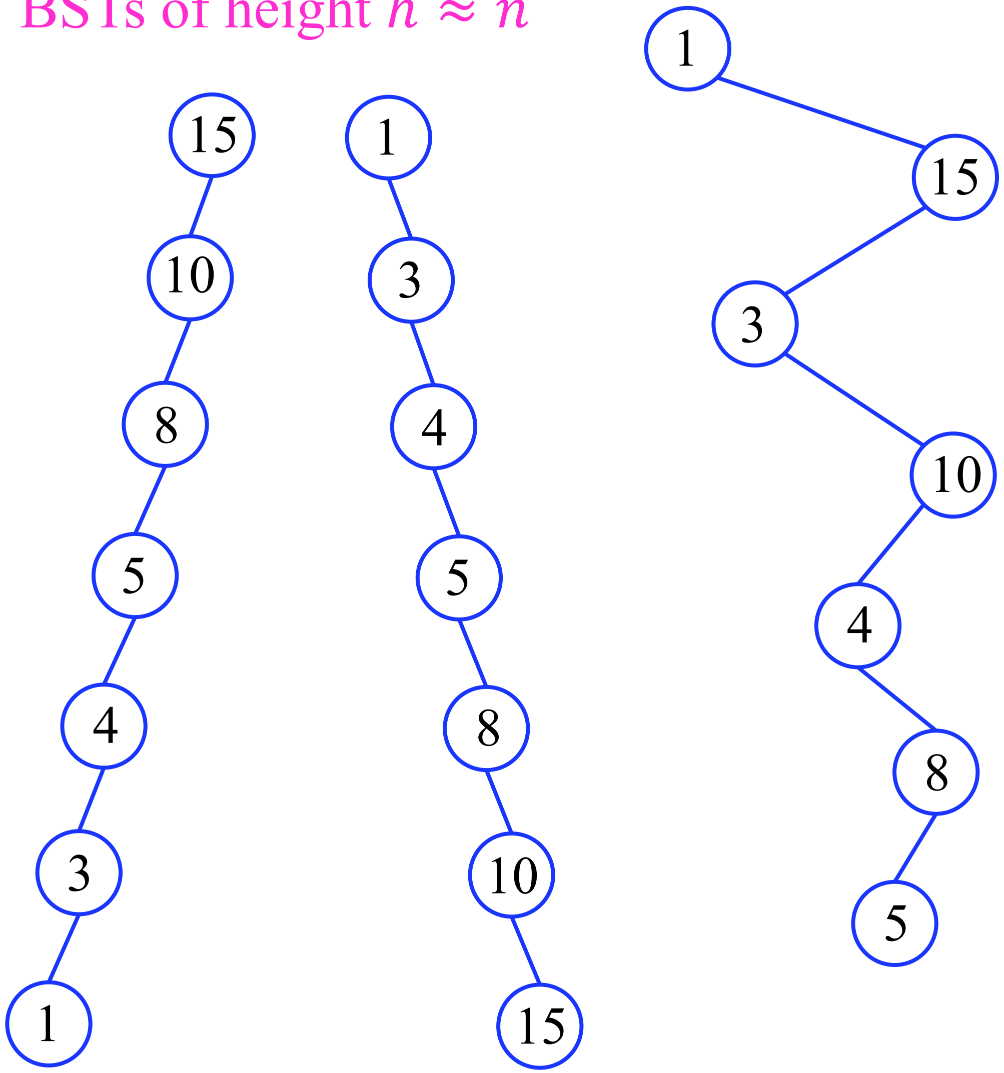
- **Proof**: The running time $T(n)$ of these operations is proportional to the number of nodes visited.
  - Find / insert: 1+h.
  - Remove: "1 + the depth of the node + the height of its highest subtree" →1+h.
  - In each case $T(n) = \Theta(h)$.
  - For a well-balanced BST, $T(n) \in O(\log n)$ (logarithmic time).
  - In the worst case $T(n) \in \Theta(n)$ (linear time) because insertions and deletions may heavily destroy the balance.

The Worse-Case Time Complexity

BSTs of height $h \approx \log n$



BSTs of height $h \approx n$



The Average-Case Time Complexity

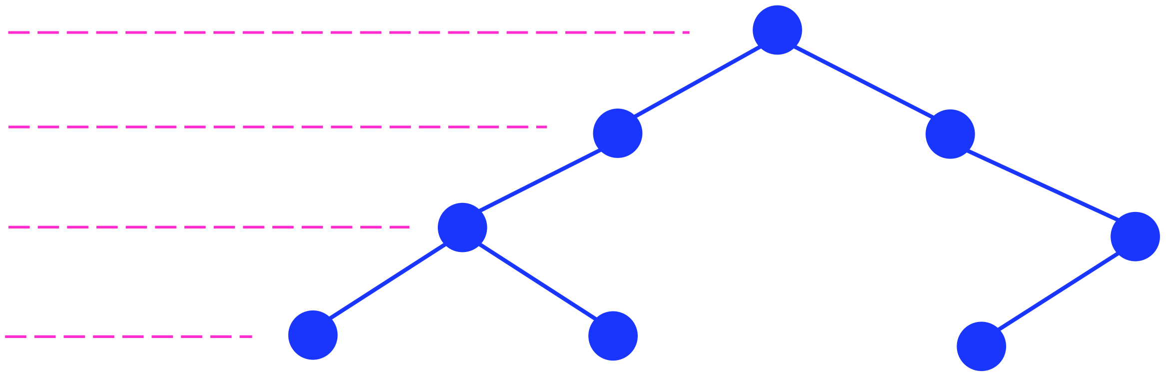
- More balanced trees are more frequent than unbalanced ones.
- **Definition** (**Internal Path Length**): The total internal path length, $S_\tau(n)$, of a binary tree τ is the sum of the depths of all its nodes.

Depth 0

1

2

3



$S_\tau(8) = 0 +$

$+ 1 + 1$

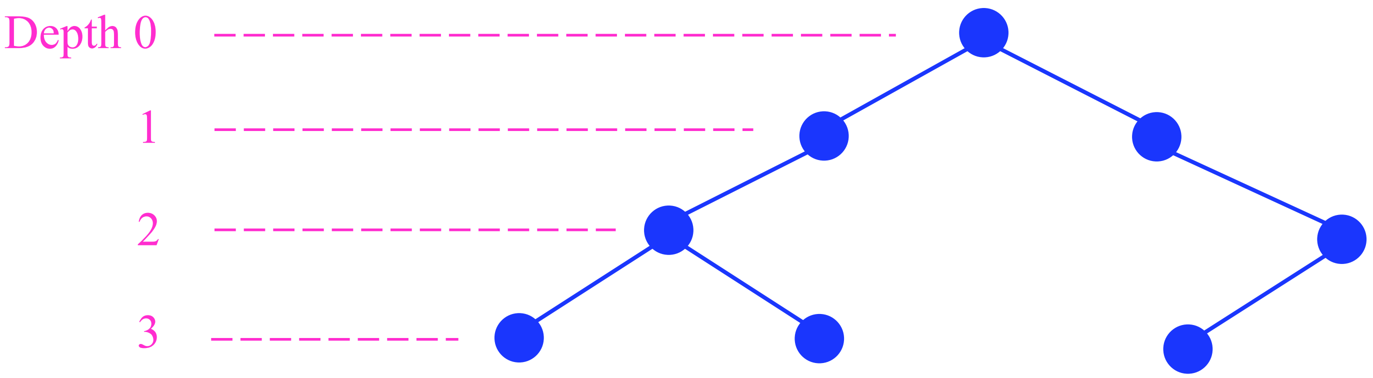
$+ 2 + 2$

$+ 3 + 3 + 3 = \mathbf{15}$

- Average complexity of a successful search in τ: the average node depth, $1/n\ S_\tau(n)$, e.g. 1/8 $S_\tau(8)$ =15/8=1.875 in this example.

The Average-Case Time Complexity (Contd.)

- Average-case complexity of searching:
  - Averaging $S_\tau(n)$ for all the trees of size n, i.e. for all possible $n!$ Insertion orders, occurring with equal probability , $\frac{1}{n!}$



$$S_\tau(8) = 0 + \\ +1+1 \\ +2+2 \\ +3+3+3 = \mathbf{15}$$

The $\Theta(\log n)$ Average-case BST Operations

- Let $S(n)$ be the average of the total internal path length, $S_\tau(n)$, over all BST τ created from an empty tree by sequences of n random insertions, each sequence considered as equally possible.

- The expected time for successful and unsuccessful search (insertion and deletion) in such BST is $\Theta(\log n)$
- **Proof**: It should be proven that $S(n) \in \Theta(n \log n)$
  - Obviously, $S(1) = 0$.
  - Any n-node tree, n>1, contains a left subtree with i nodes, a root at level 0, and a right subtree with $n - i - 1$ nodes; $0 \le i \le n - 1$.
  - For a fixed i, $S(n) = (n-1) + S(i) + S(n-i-1)$, as the root adds 1 to the path length of each other node.

The $\Theta(\log n)$ Average-case BST Operations (Contd.)

- After summing these recurrences for $0 \leq i \leq n-1$ and averaging, just the same recurrence as for the average-case quicksort analysis is obtained:

$$S(n) = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} S(i)$$

- Therefore, $S(n) \in \Theta(n \log n)$, and the expected depth of a node is $\frac{1}{n}S(n) \in \Theta(\log n)$.
- Thus, the average-case search and insertion time is in $\Theta(\log\ n)$.
- It is possible to prove (but in a more complicate way) that the average-case deletion time is also in $\Theta(\log n)$.
- The BST allow for a special balancing, which prevents the tree height from growing too much, i.e. avoids the worst cases with linear time complexity $\Theta(n)$.

SUMMARY

- Tree Data Structure
- Binary Search Tree Operations
  - find, insert, and remove operations
- Time Complexity Analysis