



CS202 – Data Structures

LECTURE-08

Binary Trees and Search Trees

Tree properties, traversal, Binary Search Trees

Dr. Maryam Abdul Ghafoor

Assistant Professor

Department of Computer Science, SBASSE

Topics

- Binary Trees
- Binary Tree Properties
- Local Search and Binary Search Trees (BSTs)

Binary Tree

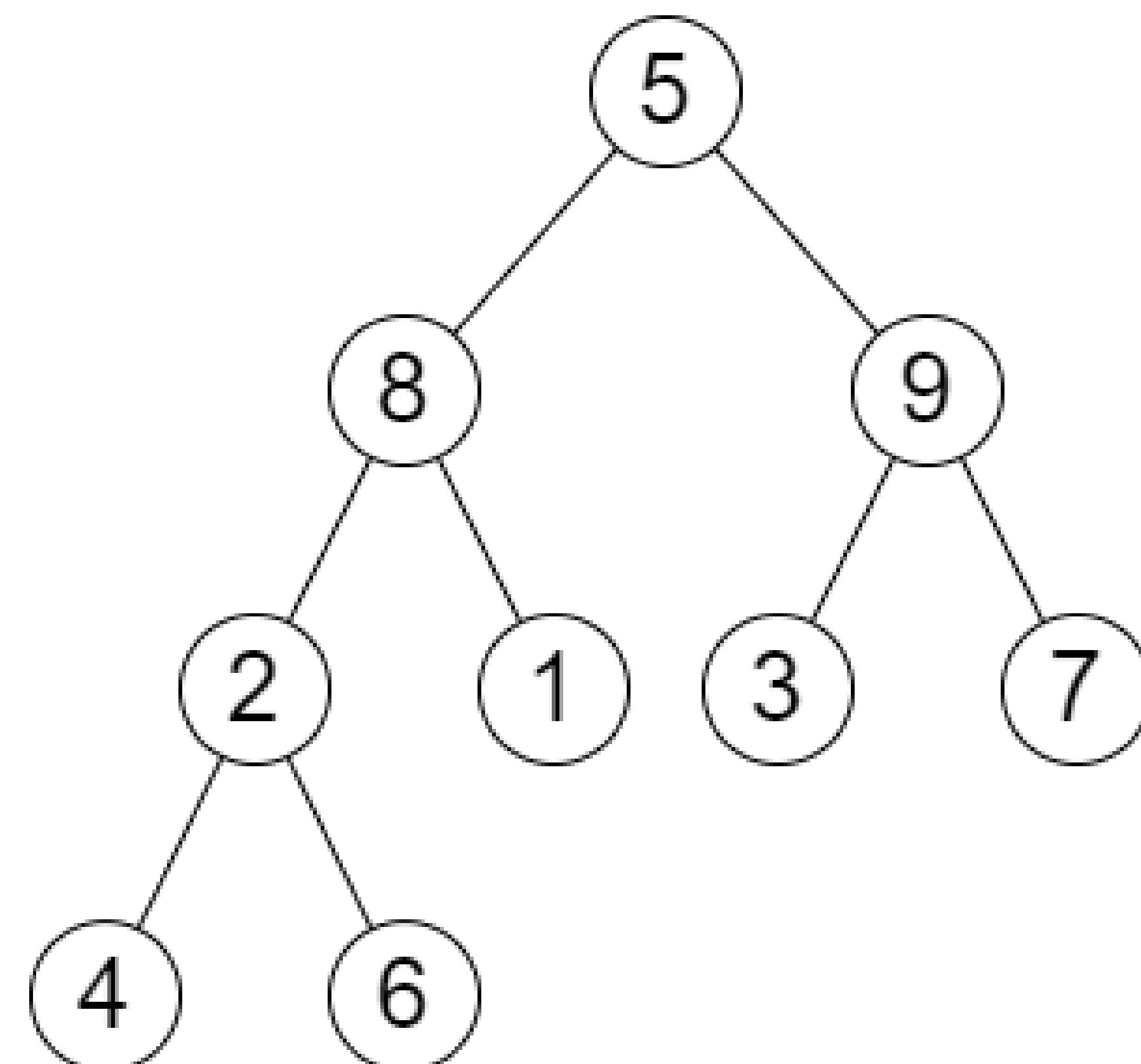


Binary Trees – Definitions Revisited

- A binary tree is an **ordered tree** in which every node has at most two children nodes
 - A left child **precedes** the right child in the ordering (by convention)
 - A node is called an **internal node** if it has one or more children and **external node** if it has no children
- A binary tree is **proper** if each node has either zero or two children (it is called improper otherwise)

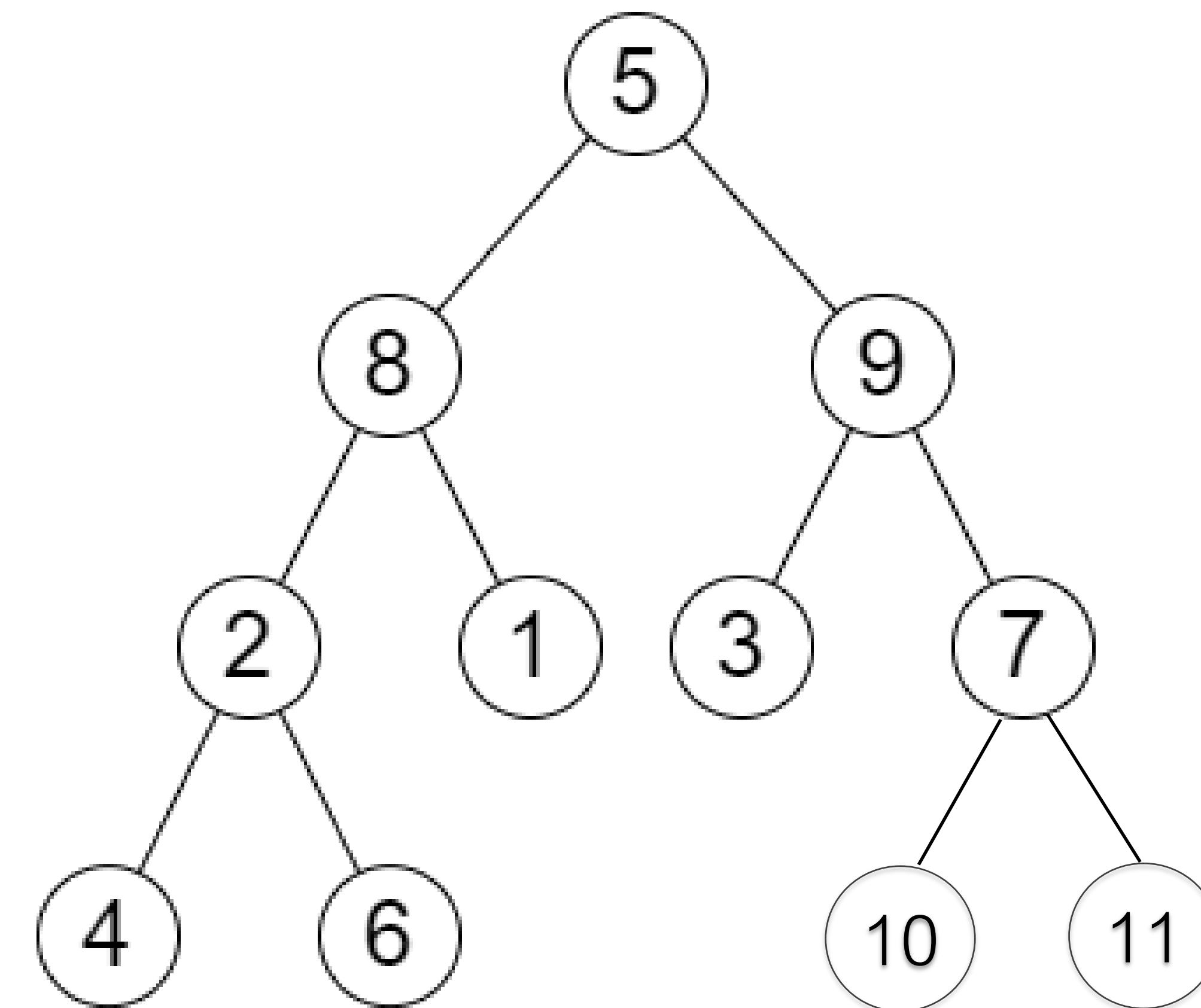
Binary Trees – More Definitions

- A **complete binary tree** is a binary tree in which all levels are **fully filled except** possibly the **last level**, which is filled from left to right.



Binary Trees – More Definitions

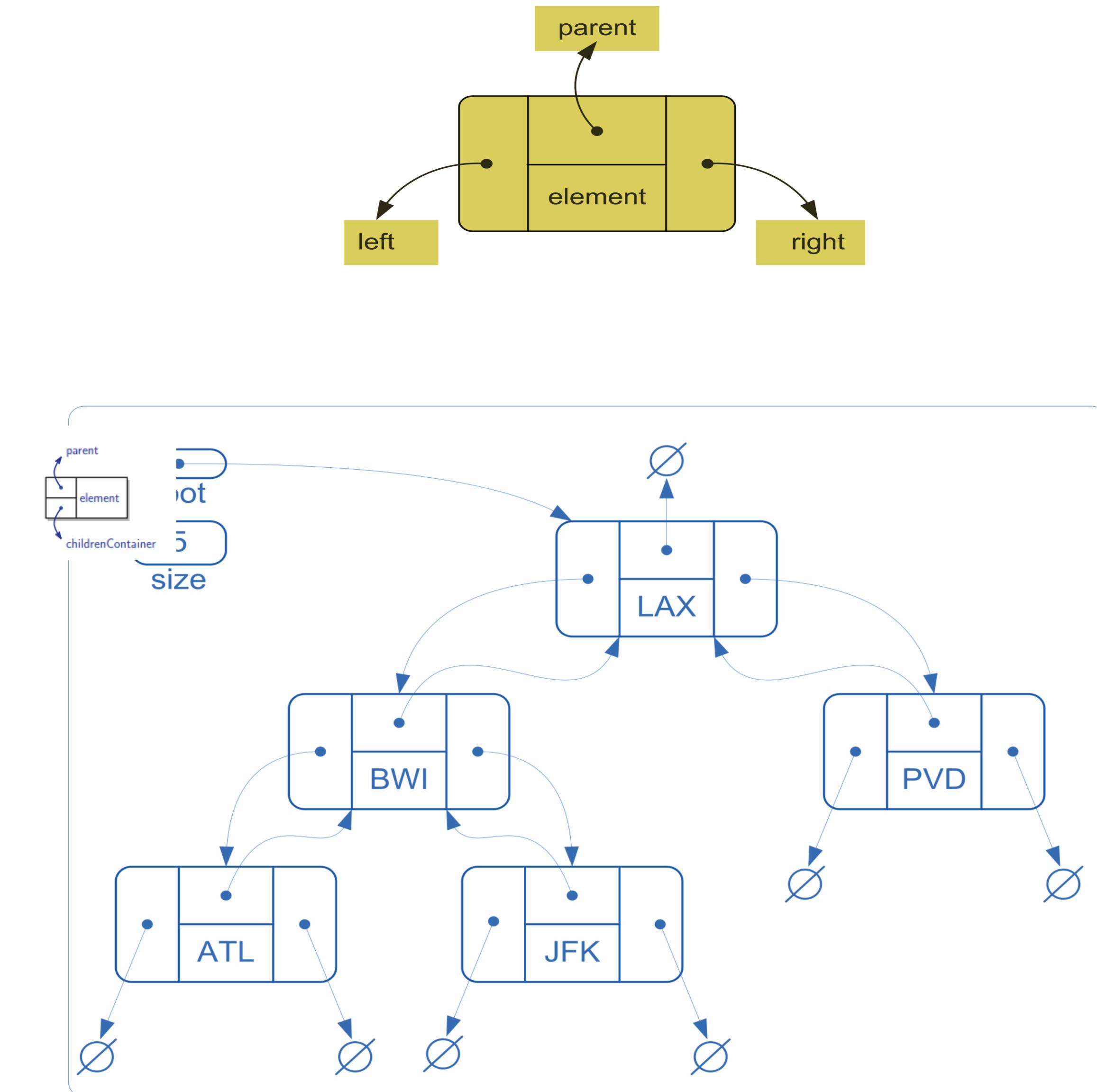
- A **full binary tree** is a binary tree in which every node has either 0 or 2 children.



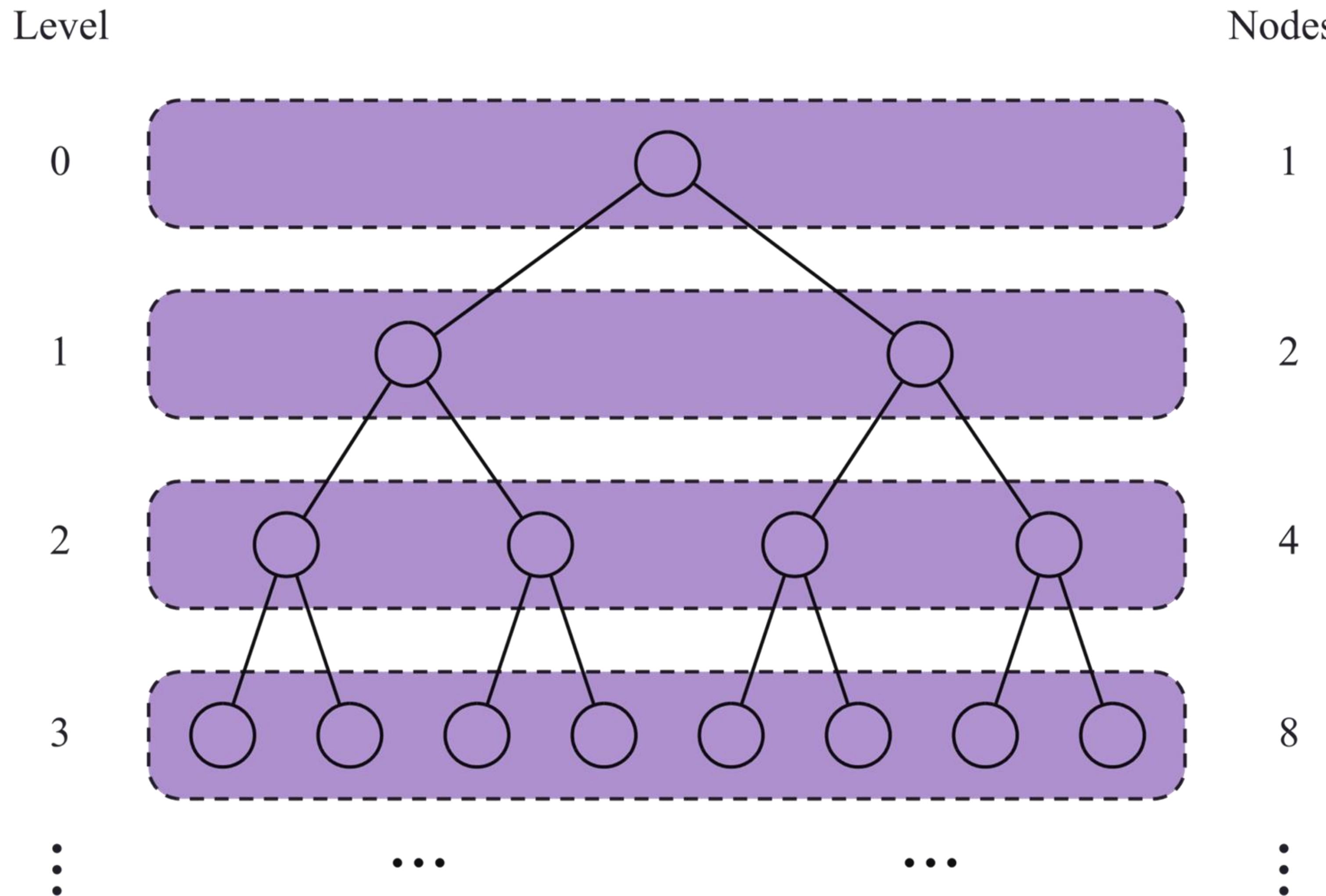
Representing Binary Trees

```
struct TreeNode {  
    string element;  
    TreeNode* parent;  
    TreeNode* left;  
    TreeNode* right;  
}
```

```
class BinaryTree{  
public:  
    BinaryTree();  
    void insert(string data);  
    TreeNode *search(int data);  
  
private:  
    int size;  
    TreeNode* root;  
};
```



Height vs. Maximum Number of Nodes



More Properties

- Let T be a non-empty binary tree, and let n , n_E , n_I and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T , respectively.
- Then T has the following properties:

1. $a \leq n \leq b$
2. $c \leq h \leq d$

What is a and b (in terms of h)?
What is c and d (in terms of n)?

$$a \leq n \leq b$$

If I give you a binary tree of **height h** then what is the minimum and maximum number of nodes can it have?

$$c \leq h \leq d$$

If I give you a binary tree with **n nodes**, then what is the minimum and maximum height can it assume?

More Properties

- Let T be a non-empty binary tree, and let n , n_E , n_I and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T , respectively.
- Then T has the following properties:

$$1. h + 1 \leq n \leq 2^{h+1} - 1$$

$$2. \log(n + 1) - 1 \leq h \leq n - 1$$

More Properties

- Let T be a non-empty binary tree, and let n , n_E , n_I and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T , respectively.
- Then T has the following properties:

$$\begin{aligned} A &\leq n_E \leq B \\ C &\leq n_I \leq D \end{aligned}$$

More Properties

- Let T be a non-empty binary tree, and let n , n_E , n_I and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T , respectively.
- Then T has the following properties:

3. $1 \leq n_E \leq 2^h$
4. $h \leq n_I \leq 2^h - 1$

Tree Traversals – Summary

- Preorder Traversal → root – left – right
- Post order traversal → left – right – root
- In-order Traversal → left – root – right
- Level order traversals → level by level

Implementing Level Order Traversal

- Level order traversal is naturally not recursive!
- Use a **queue**, which initially only contains the root

Initially, the queue contains the root node

Repeat:

 Dequeue a node

 Visit it

 Enqueue its children nodes(left→right)

Until queue is empty

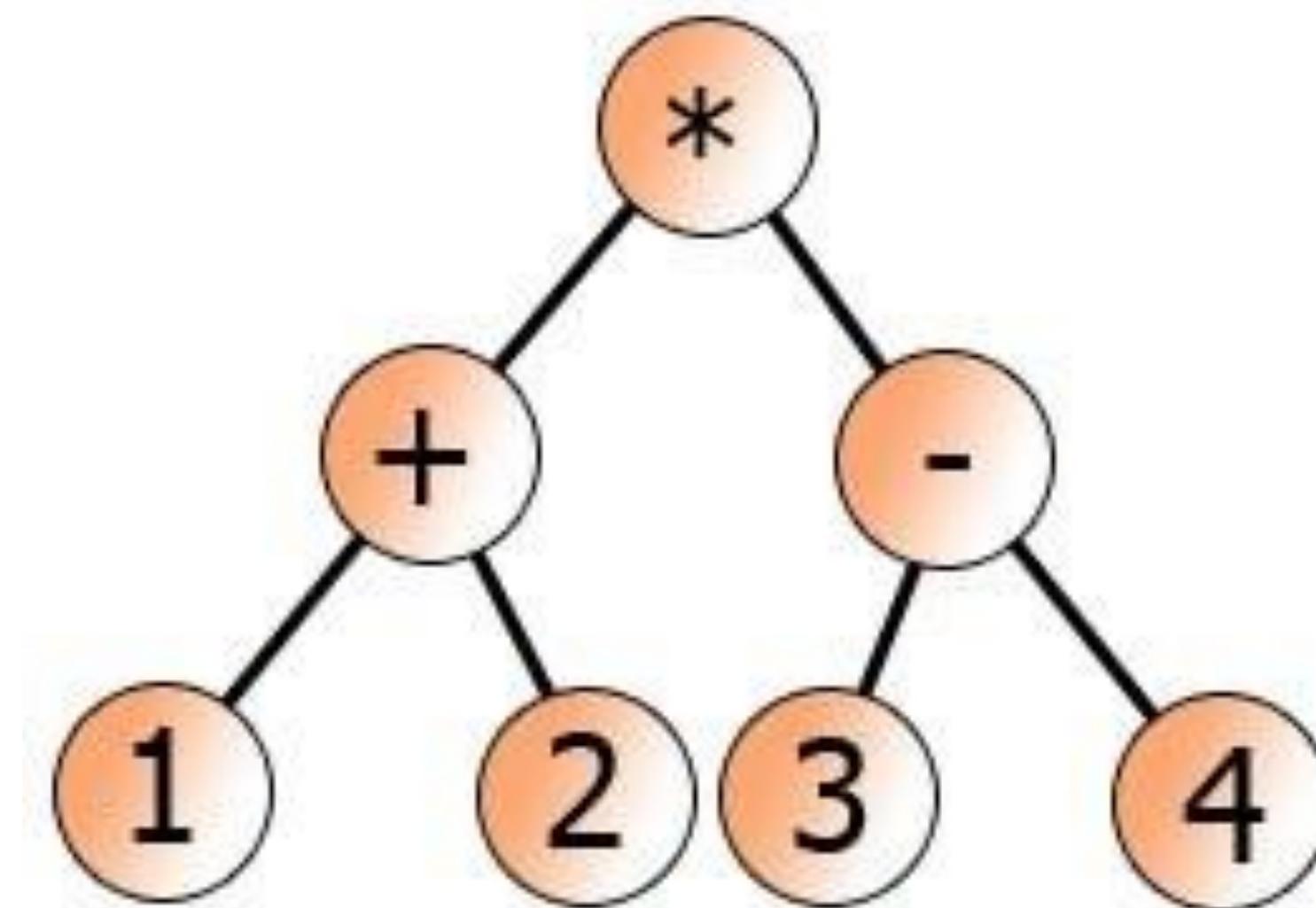
Applications of Tree Traversal

Customize and move the “do something,” and that’s the basis for dozens of algorithms and applications

```
void BinaryTree::traverse(TreeNode *node) {  
    if (node != NULL) {  
        traverse(node->left);  
        // “do something”  
        traverse(node->right);  
    }  
}
```

Evaluating Expression

- Inorder (infix): $1+2*3-4$
- Preorder (prefix): $*+12-34$
- Postorder (postfix): $12+34-$ *



$((1+2)*(3-4))$

Natural way of writing expression

Local Search and Binary Trees

Price Ranges

- Retrieve headsets matching the specified price criteria

The screenshot shows the Amazon search results for "gaming headsets". The search bar at the top has "gaming headsets" entered. Below the search bar, there are navigation links: All, Today's Deals, Customer Service, Registry, Gift Cards, and Sell. The main content area displays 1-16 of over 30,000 results for "gaming headsets". On the left, there is a sidebar with "Popular Shopping Ideas" like Video Game, Wireless, Desktop, and Animal. Under "Customer Reviews", there is a red box highlighting a price filter: "Price \$1 - \$550" with a slider and a "Go" button. Below this are sections for "Deals & Discounts" (All Discounts, Today's Deals) and "Brands" (Razer, Logitech G, SteelSeries, HyperX, Turtle Beach, Corsair, JBL, See more). There are also filters for "Headphones Connectivity Technology" (Wired, Wireless). The main results section shows two products:

Product 1: Gaming Headset for PC, Ps4, Ps5, Xbox Headset with 7.1 Surround Sound, Gaming Headphones with Noise Cancelling Mic RGB Light Over Ear Headphones for Xbox Series X/S, Switch
Top Reviewed for Sound quality
★★★★★ 5,046
10K+ bought in past month
\$23⁹⁹ List: \$29.99
Save \$4.00 with coupon
Delivery Thu, Feb 20
Ships to Pakistan
Add to cart
More Buying Choices
\$19.19 (2+ used & new offers)
Color options: White, Black, Blue, Red, Green, Other +1

Product 2: Overall Pick
2.4GHz Wireless Gaming Headset for PS5, PS4 Fortnite & Call of Duty/FPS Gamers, PC, Nintendo Switch, Bluetooth 5.3 Gaming Headphones with Noise Canceling Mic, Stereo Sound, 40+Hr Batter...
Top Reviewed for Sound quality
★★★★★ 8,883
10K+ bought in past month
\$34⁹⁹ List: \$49.99
Save \$4.00 with coupon
Delivery Thu, Feb 20

Local Search – Definition

- A local search **data structure** stores elements each with a **key** coming from an **ordered** set. It supports operations:
 - **RangeSearch(x, y):** Returns all elements with keys between x and y (including x and y).
 - **NearestNeighbors(z):** Returns the element with keys on either side of z .

Dynamic Updates: Insertions and Deletions

- We would also like to be able to **modify** the data structure as we go
 - **Insert(x)**: Adds an element with key x
 - **Delete(x)**: Removes the element with key x

Possible Design Choices so far

- How good are structures we have learnt so far for implementing local search?
 - Arrays
 - Sorted Arrays
 - Linked Lists

Array

- RangeSearch(1, 15) $O(n)$ X
- NearestNeighbors(15) $O(n)$ X
- Insert(12) $O(1)$ ✓
- Delete(11) $O(n)$ X

7	10	4	13	1	6	15	
---	----	---	----	---	---	----	--

RangeSearch(5, 12)

NearestNeighbors(3)

2 – Sorted Array

- RangeSearch(1,15) $O(\log n)$ ✓
- NearestNeighbors(15) $O(\log n)$ ✓
- Insert(12) $O(n)$ ✗
- Delete(11) $O(n)$ ✗

1	4	6	7	10	13	15	
---	---	---	---	----	----	----	--

RangeSearch(5, 12)

NearestNeighbors(3)

3 – Linked List

- RangeSearch(1,15) $O(n)$ X
- NearestNeighbors(15) $O(n)$ X
- Insert(12) $O(1)$ ✓
- Delete(11) $O(n)$ X

RangeSearch(5, 12)

NearestNeighbors(3)



Need something new

- Sorted arrays can **search efficiently** but are inefficient with insertion/deletion
 - Therefore, none of the existing data structures work
- We need a **new** data structure for local search

RangeSearch:
NearestNeighbors:
Insert:
Delete:

Array

$O(n)$

$O(n)$

$O(1)$

$O(n)$

Sorted Arrays

$O(\log n)$

$O(\log n)$

$O(n)$

$O(n)$

Linked Lists

$O(n)$

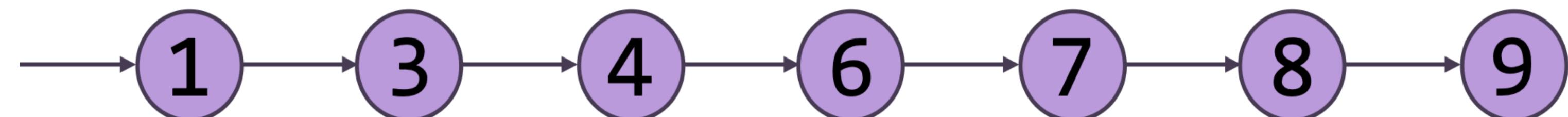
$O(n)$

$O(1)$

$O(n)$

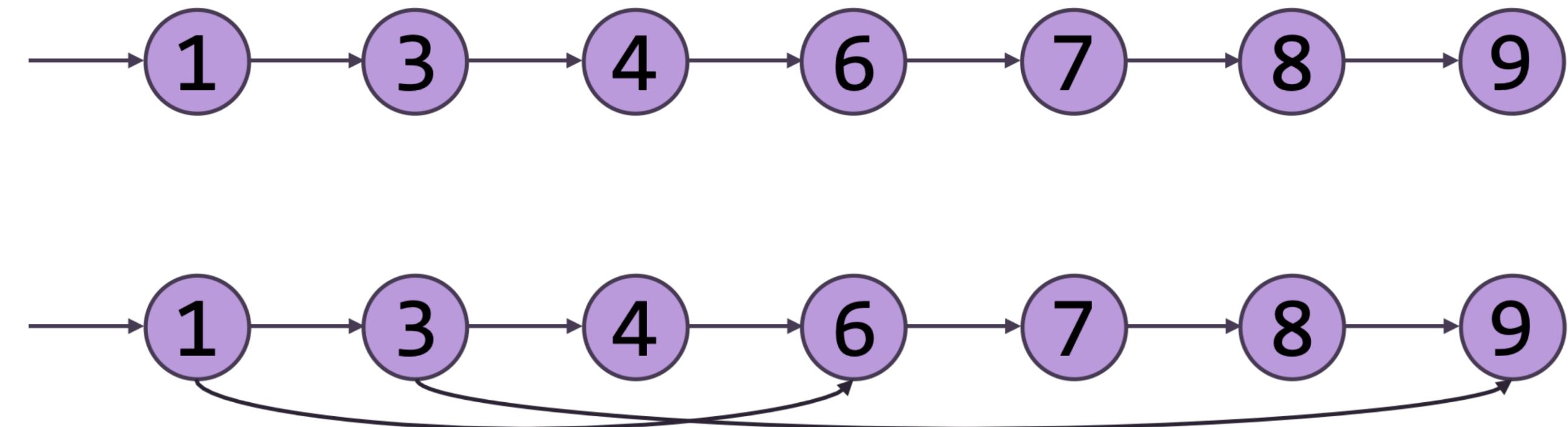
Let's consider Sorted Linked List

- **Fundamental problem:** slow search even though it is in order
- How can we **speed up** search?



Speeding Up Search in Linked Lists

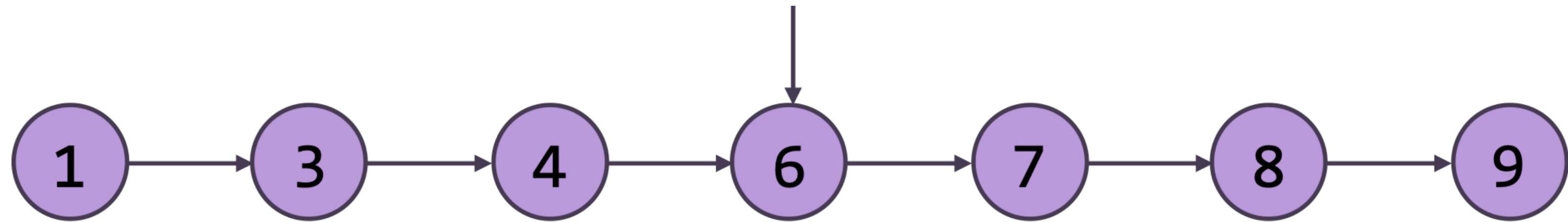
- **Fundamental problem:** slow search even though it is in order
- How can we **speed up** search?



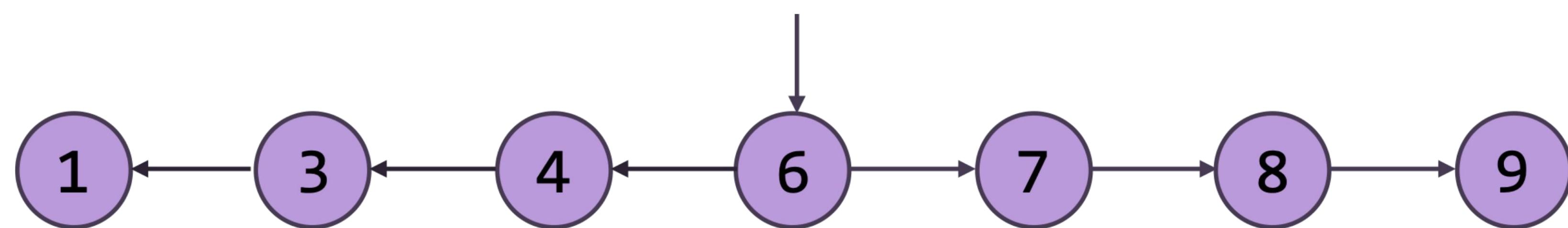
One idea: add (random) express lanes → Skip List

Optimizing Search in Sorted Linked Lists: Changing the Entry Point

- Move pointer to middle

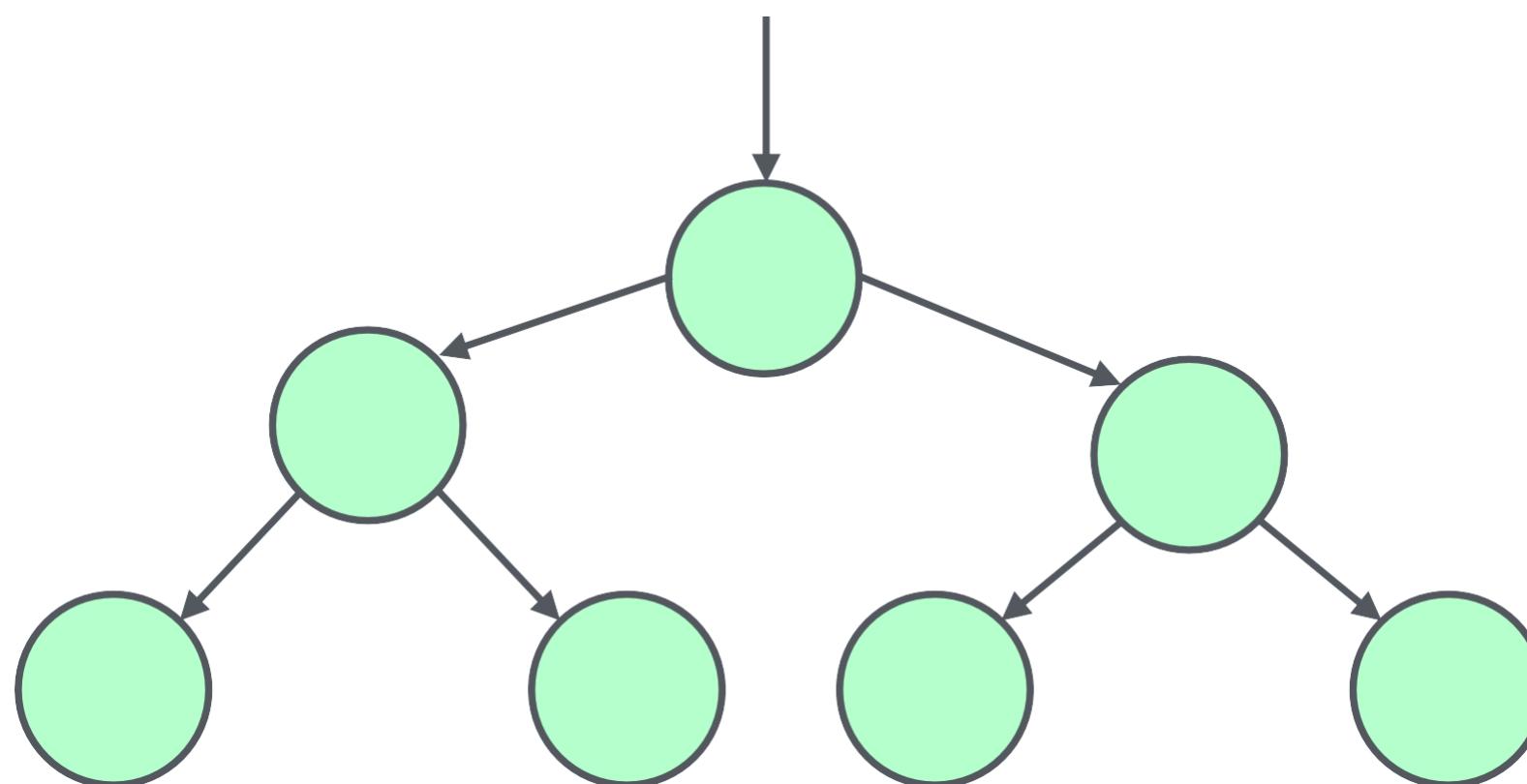
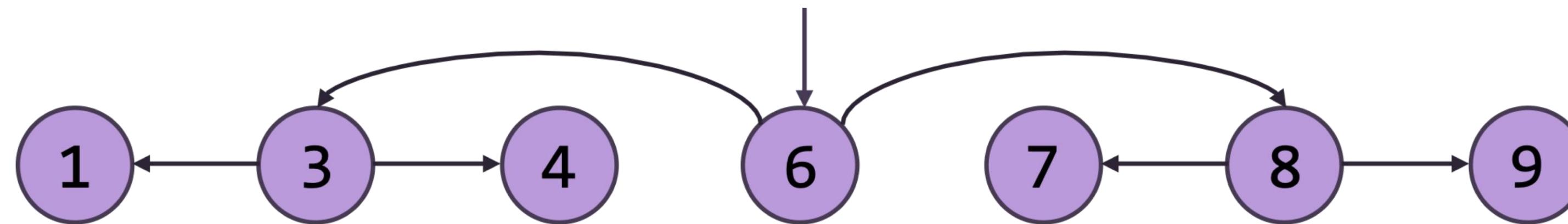


- Flip left links. Halves search time!



Can we do even Better?

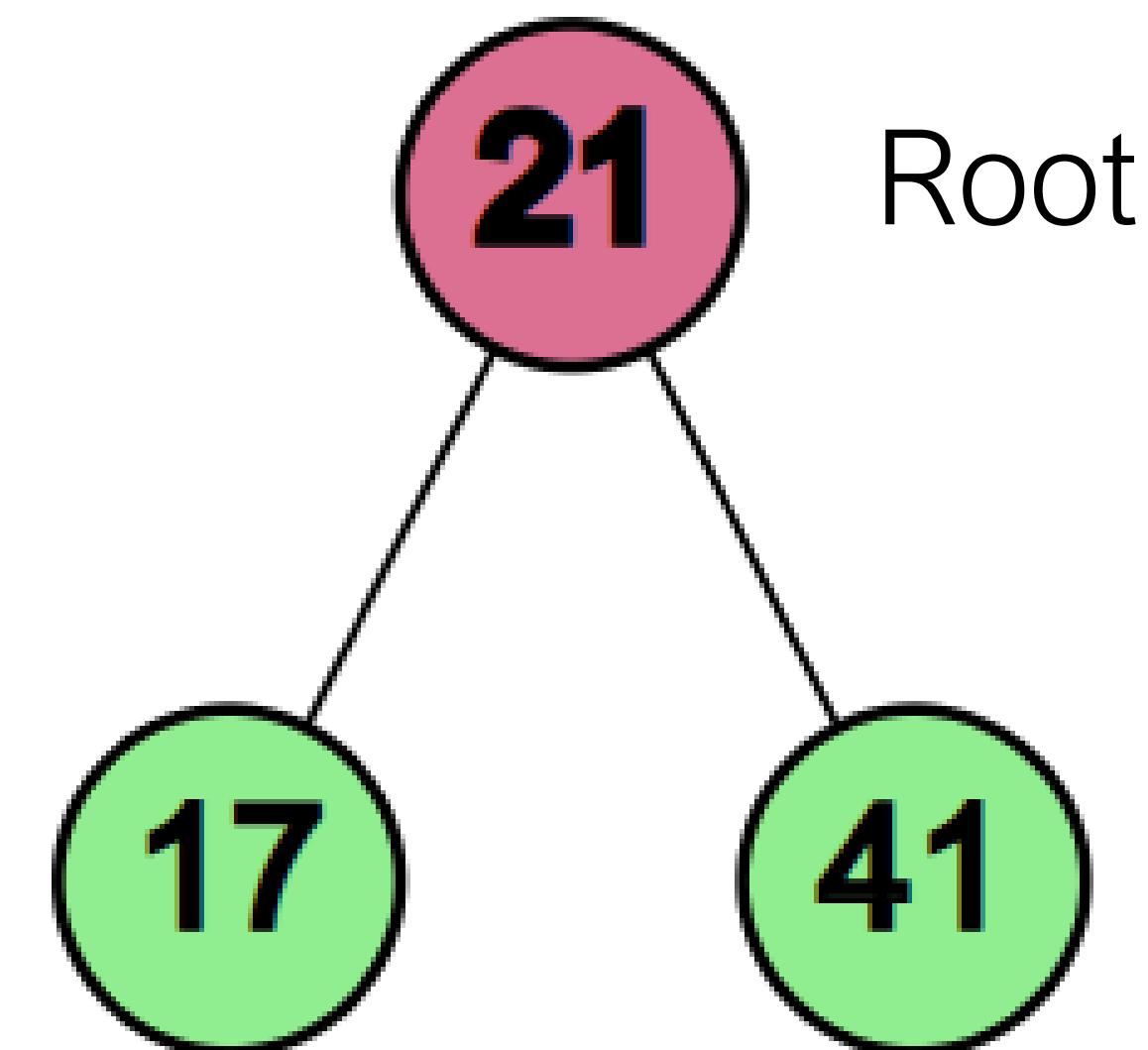
- Change entry points, flip links, allow big jumps



So, we can adapt our linked list to support array-style binary search

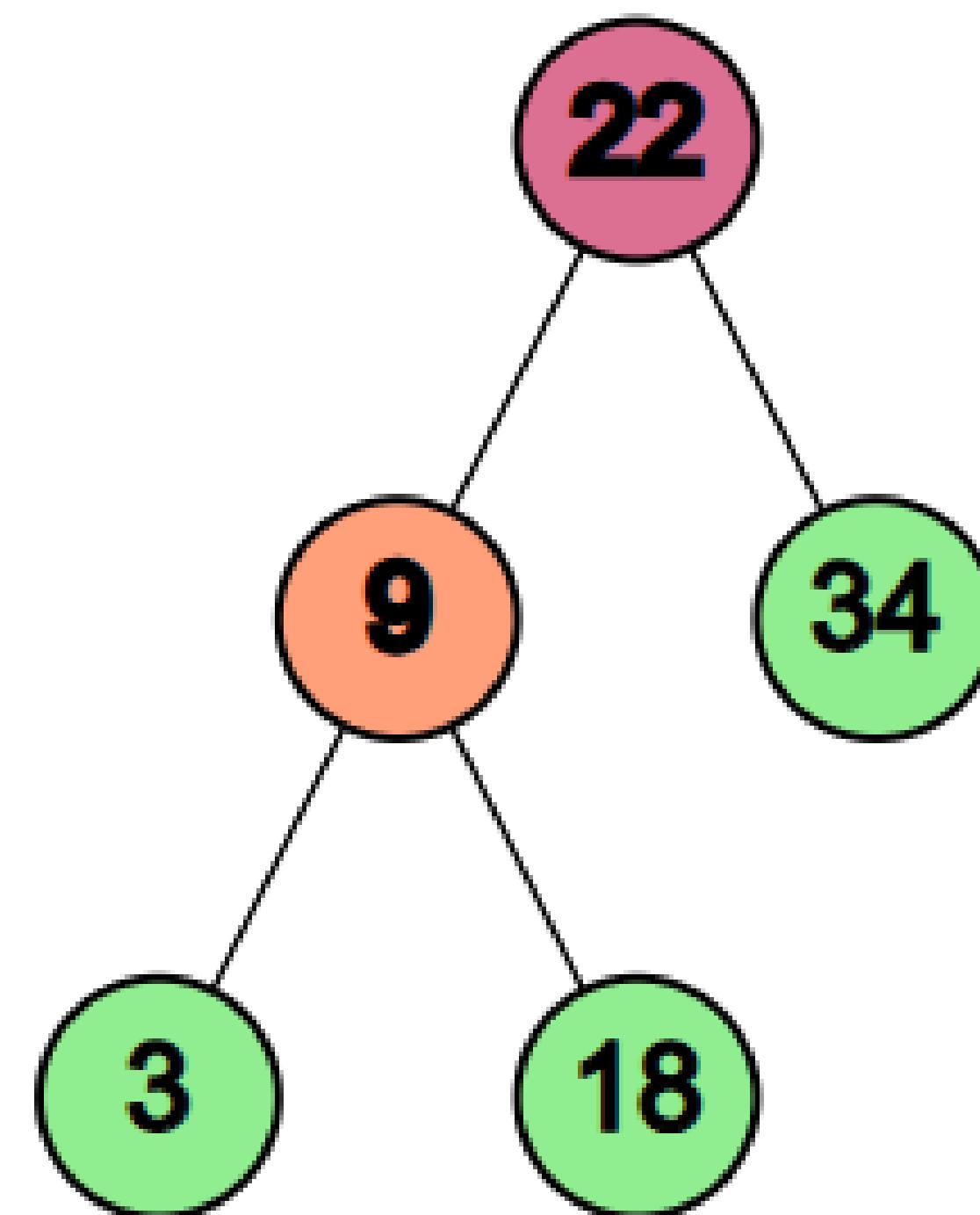
Binary Search Tree (BST)

- A **binary search tree (BST)** is a binary tree where (for all nodes):
 - Key of the **left** child is smaller than the parent's key
 - Key of the **right** child is greater than the parent's key
 - Each node stores a unique key



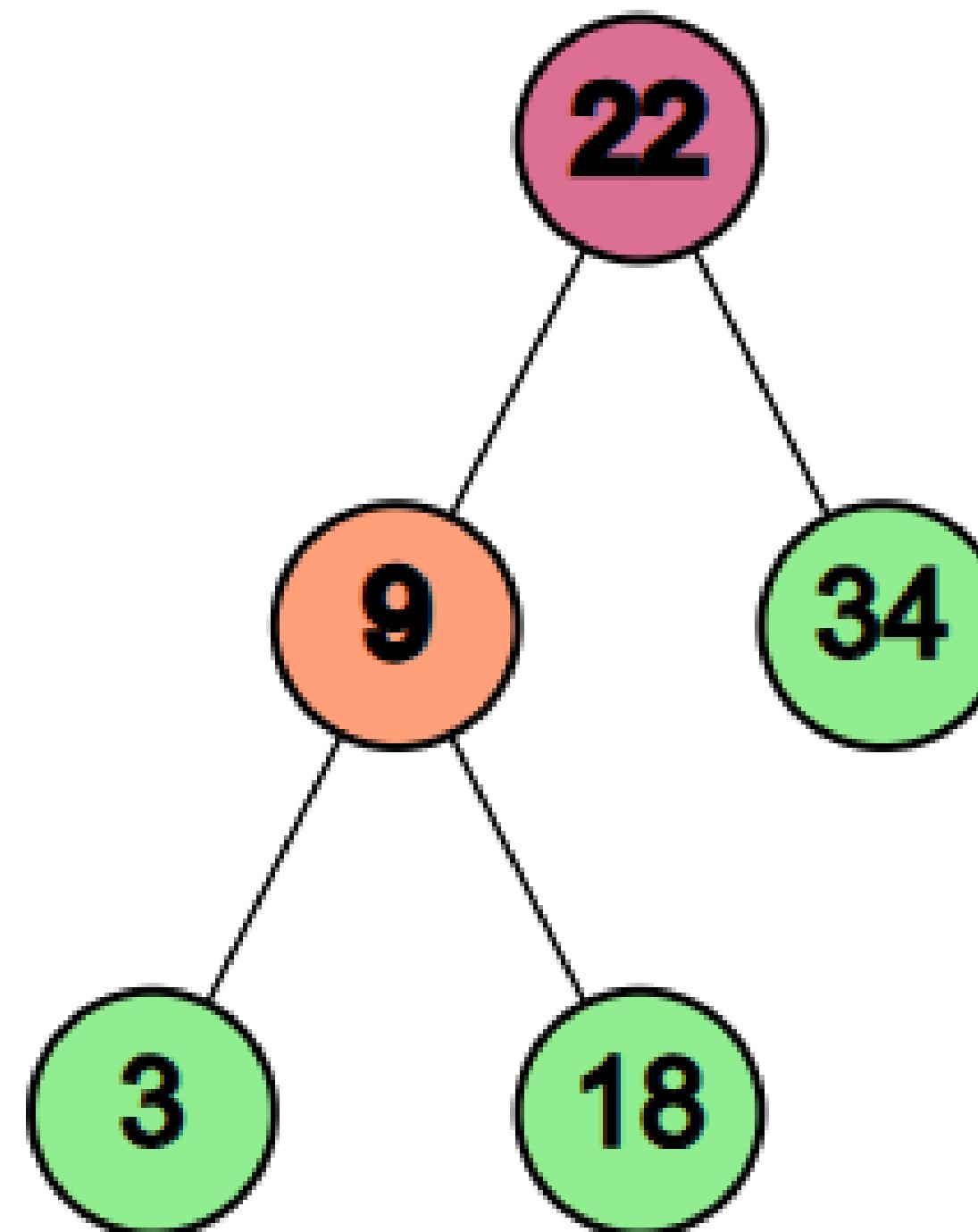
Can we have different BSTs for the same keys?

- Store keys: 22, 9, 34, 18, 3

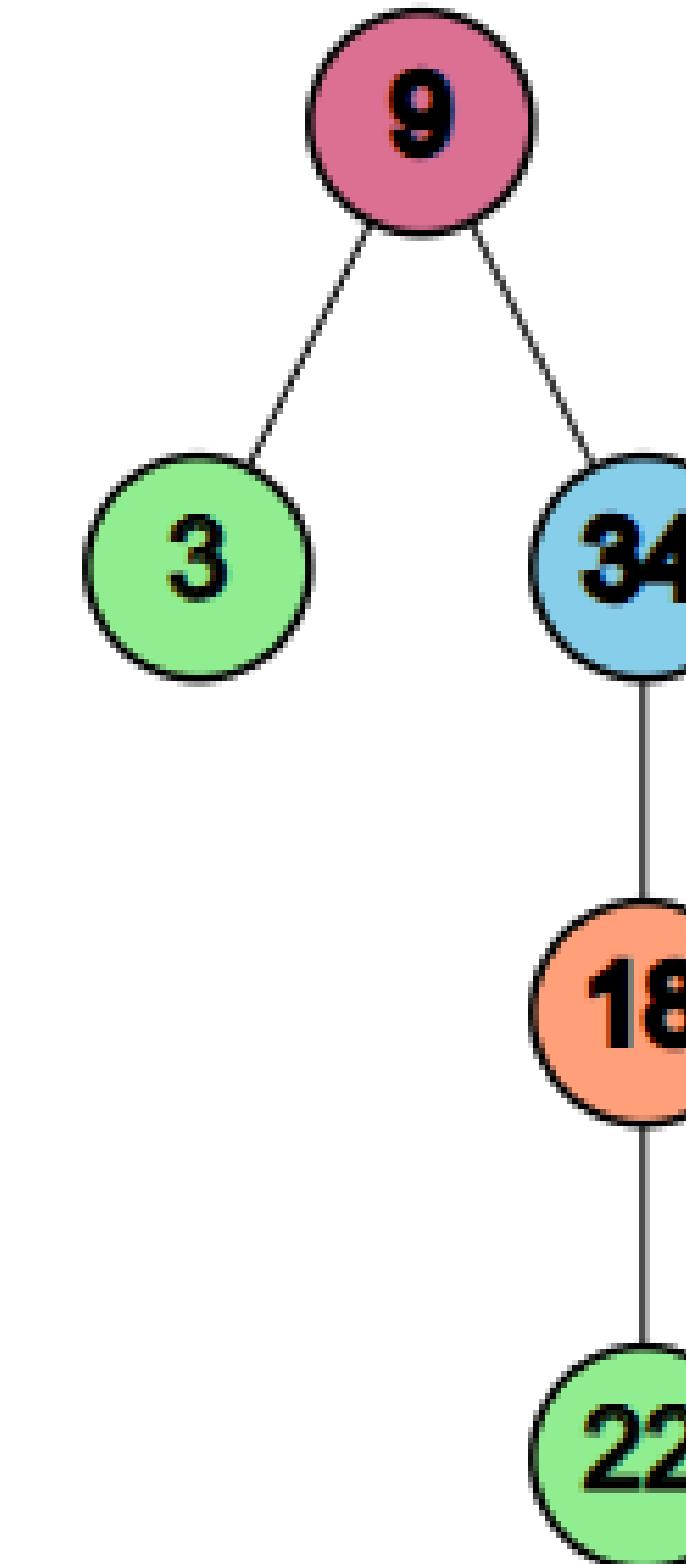


Can we have different BSTs for the same keys?

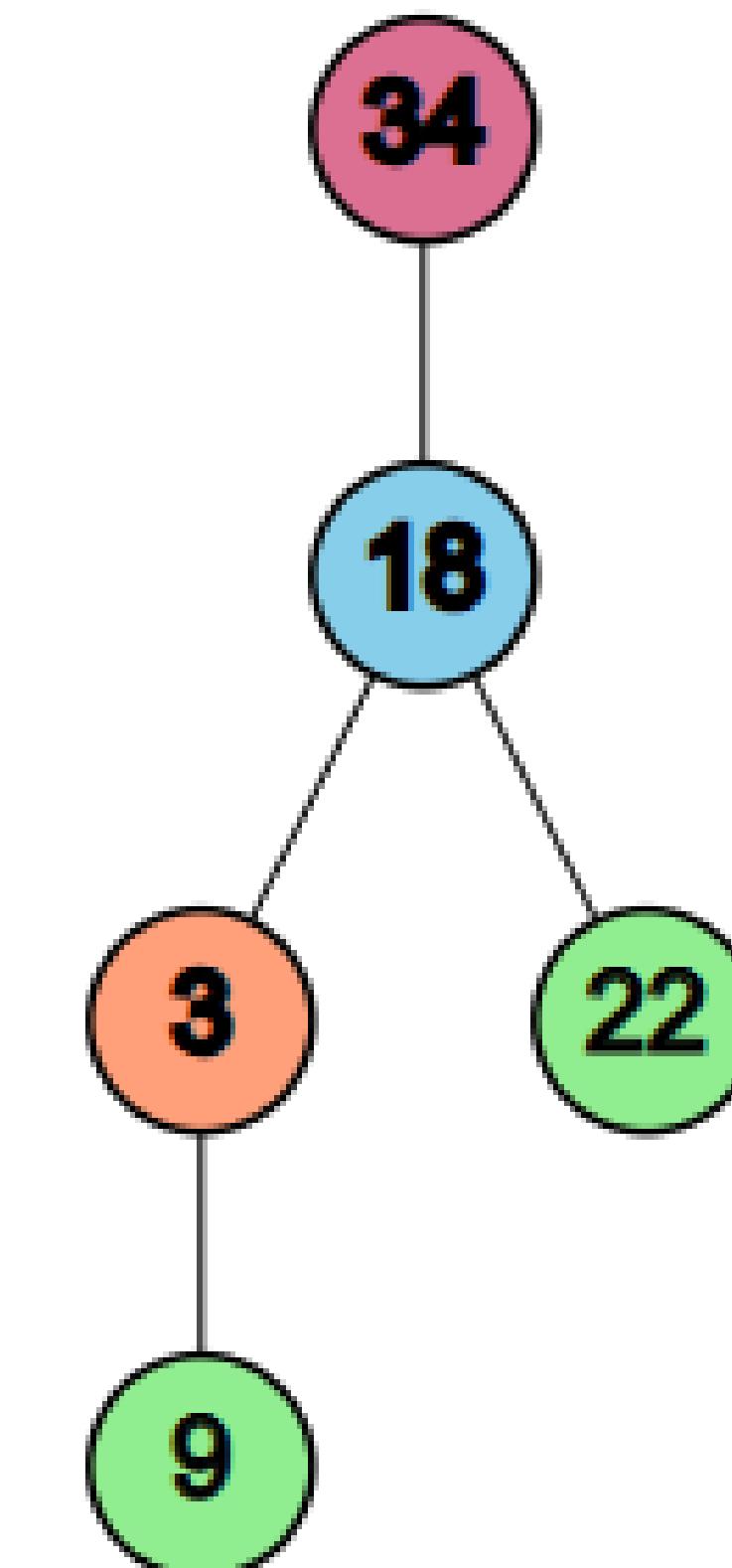
- Store keys: 22, 9, 34, 18, 3



22, 9, 34, 18, 3



9, 34, 18, 22, 3



34, 18, 22, 3, 9

Exercise

- Write iterative solution of tree traversals.
 - Pre-order, post-order, in-order, level-order
- Hint: you can use stacks (one or more), queues etc.

Questions

