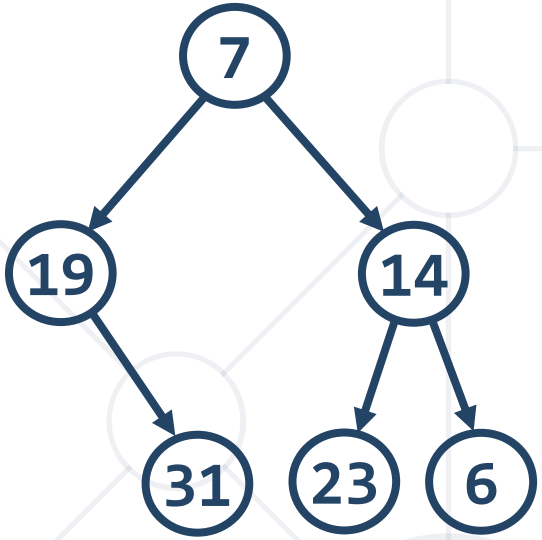
Binary Trees, Heaps and BST

Terminology, Traversal and Operations



SoftUni Team

Technical Trainers

Software University

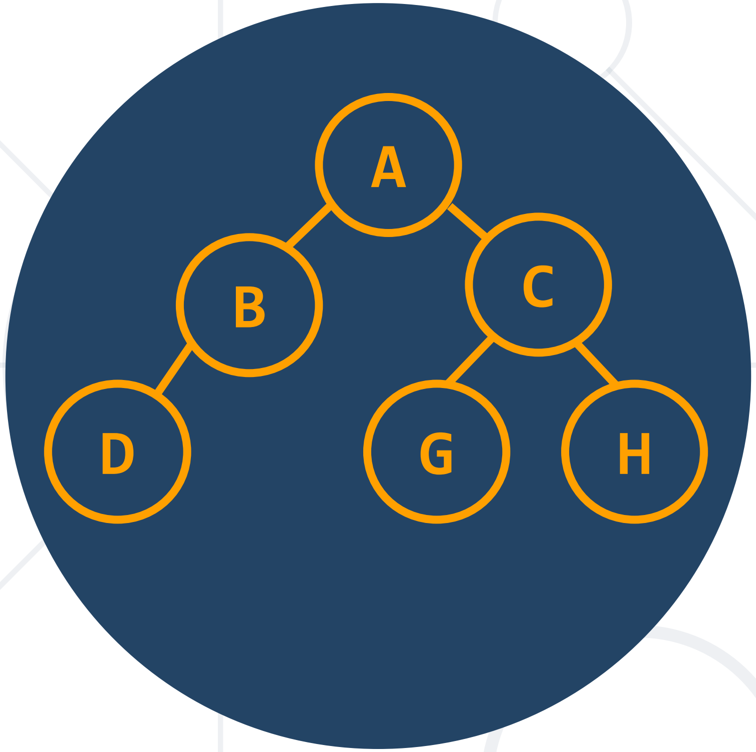
SoftUni

Software University

https://softuni.bg

Table of Contents

1. Binary Trees
   - Traversal algorithms
2. Binary Search Trees
3. Heaps
   - Binary heap, Min/Max heaps
4. PriorityQueue

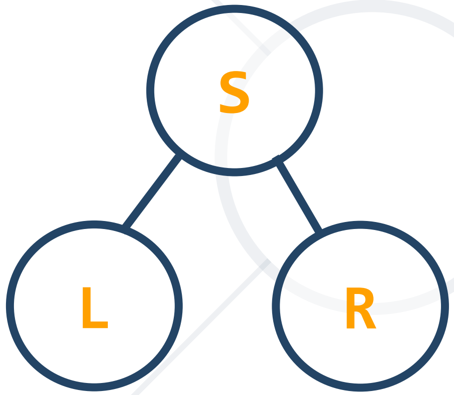


Binary Trees and BT Traversal

Preorder, In-Order, Post-Order

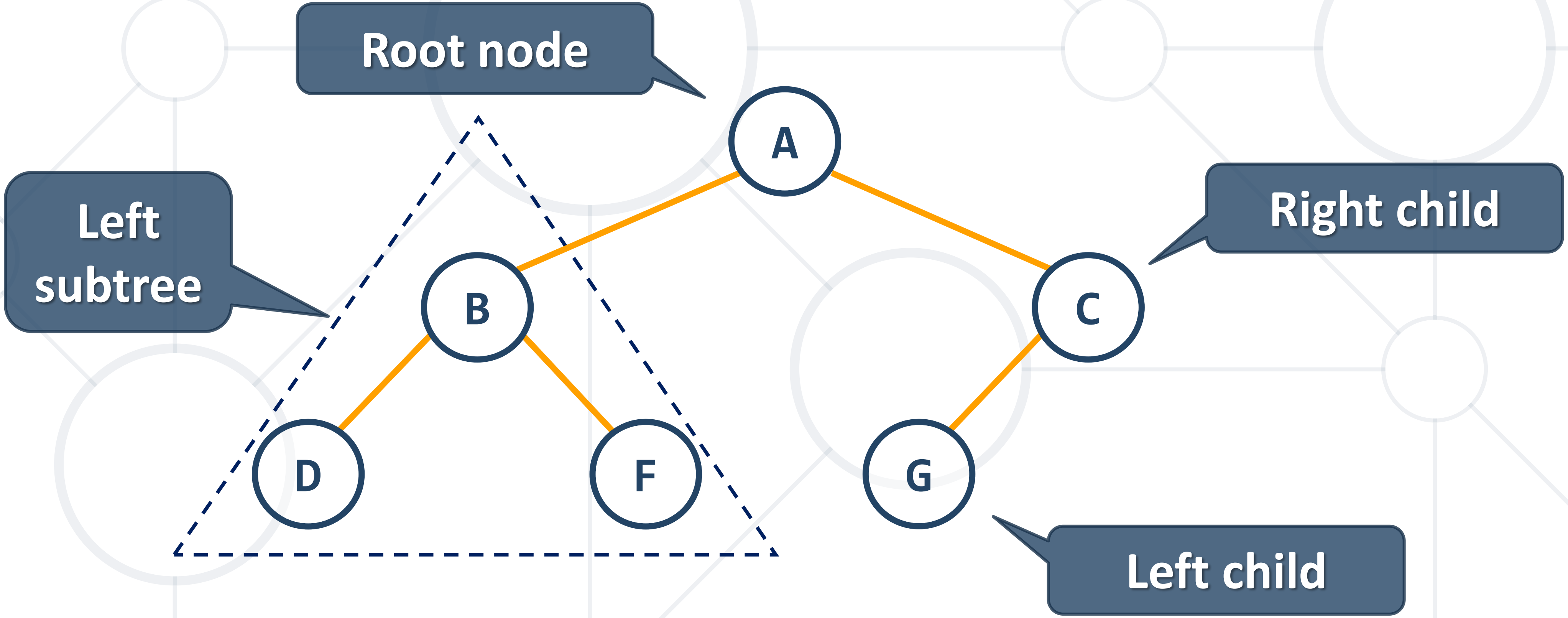
Binary Tree

- ADS representing tree like hierarchy
- Each node has **at most two** children
  - Children are called **left** and **right**



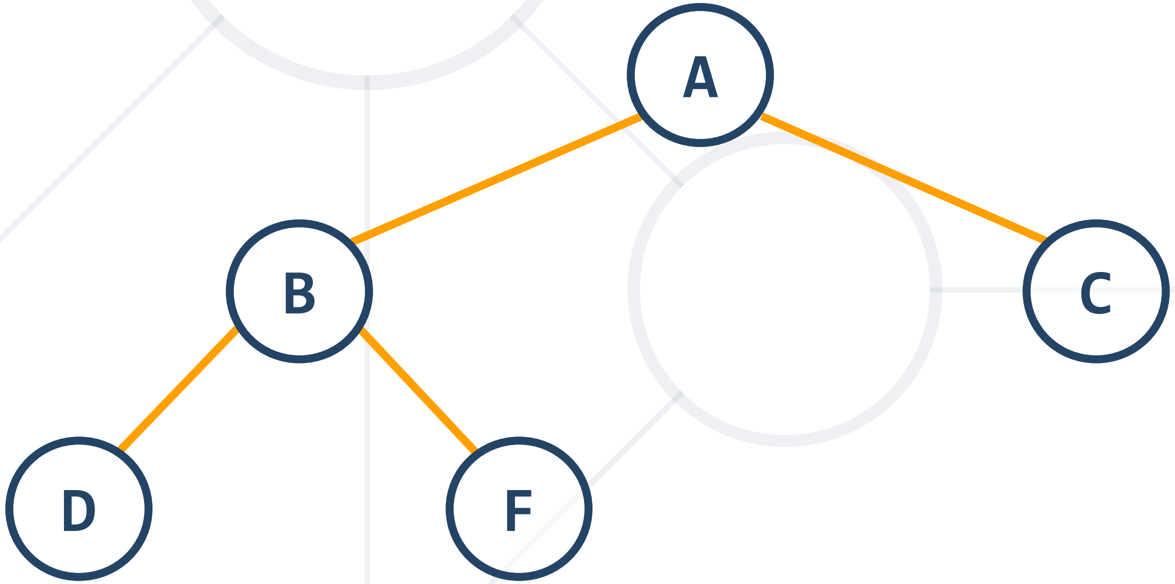
Binary Trees

- **Binary trees**: Each node **has at most 2 children** (**left** and **right**)



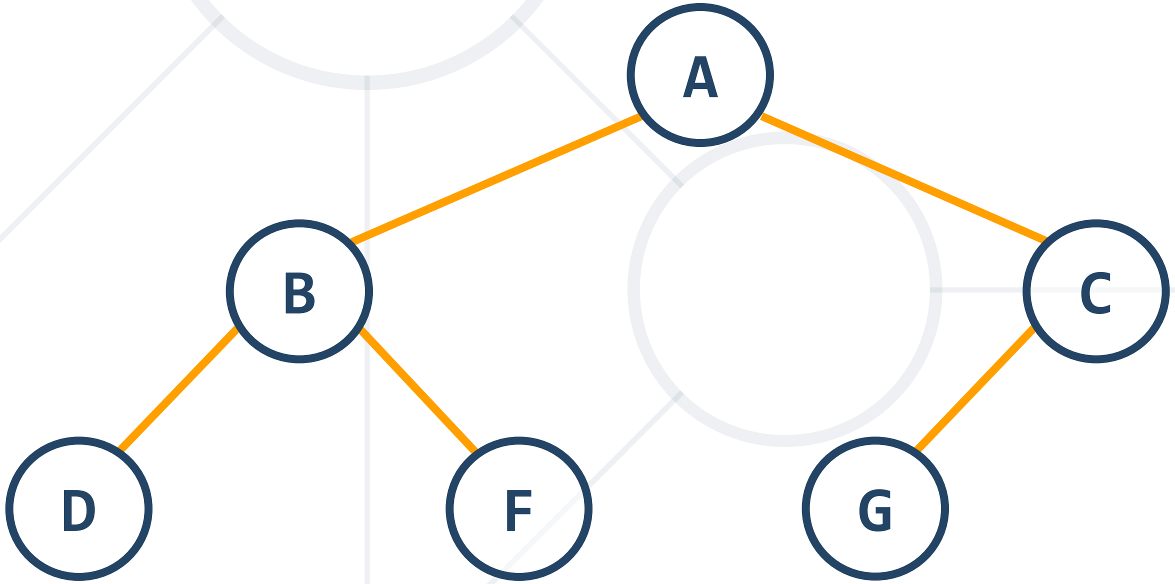
Types of Binary Trees

- **Full** – each node has **0** or **2** children



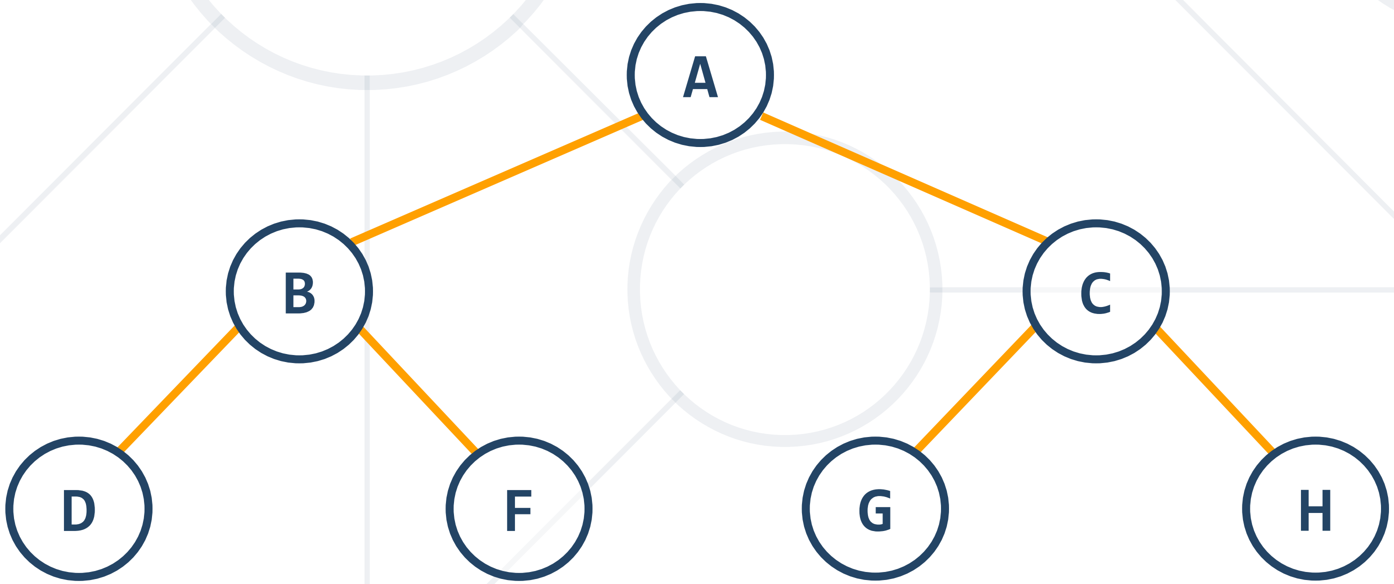
Types of Binary Trees

- **Complete** – nodes are filled **top** to **bottom** and **left** to **right**



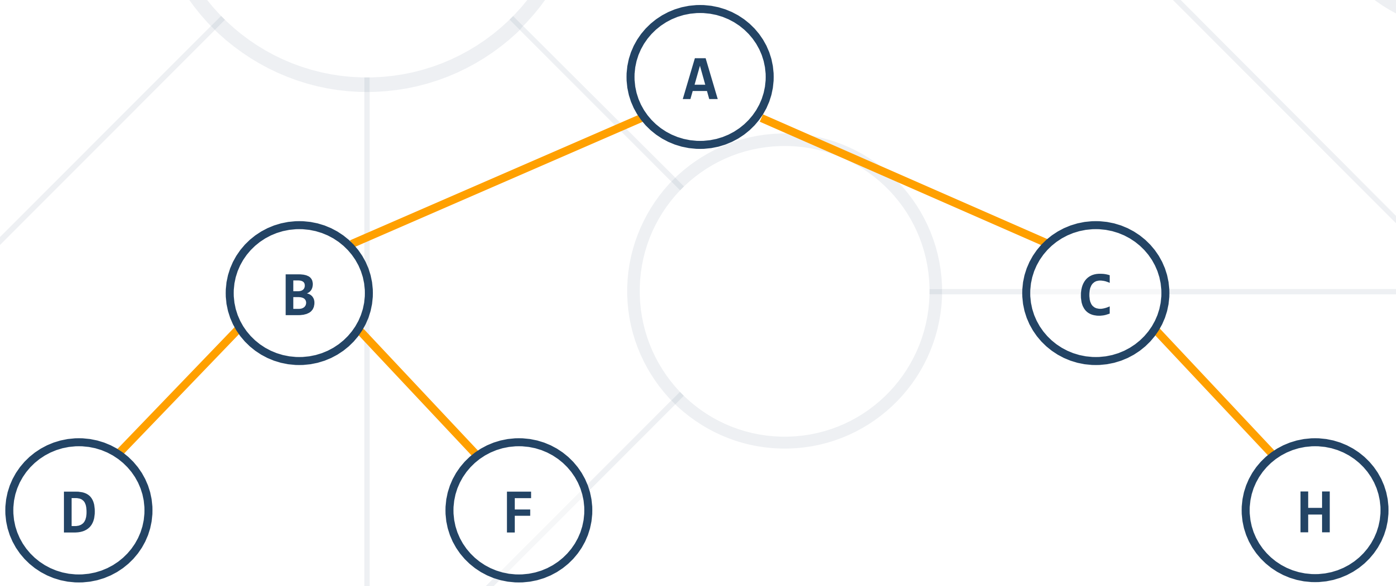
Types of Binary Trees

- **Perfect** – combines **complete** and **full**, leafs are at the **same level**, internal nodes have exactly **two** children



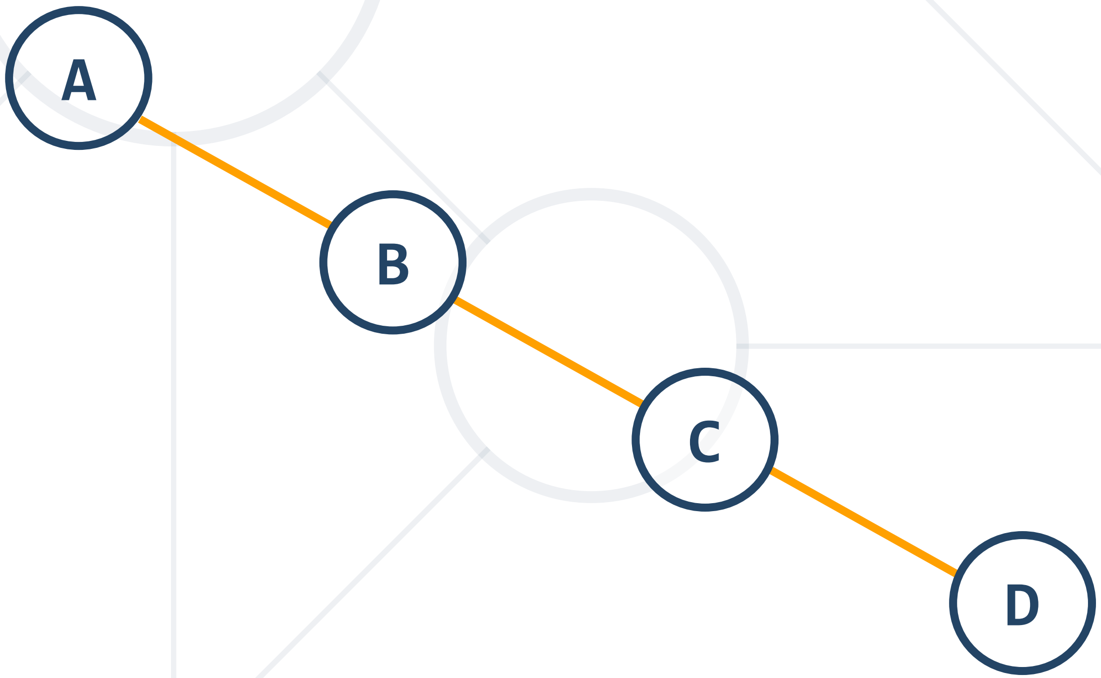
Types of Binary Trees

- **Balanced** – satisfies the following constraints
  - The left and right subtrees' heights differ by at most one
  - Left and right subtrees are balanced



Types of Binary Trees

- **Degenerate** – each parent has exactly one child. Behaves like a linked list

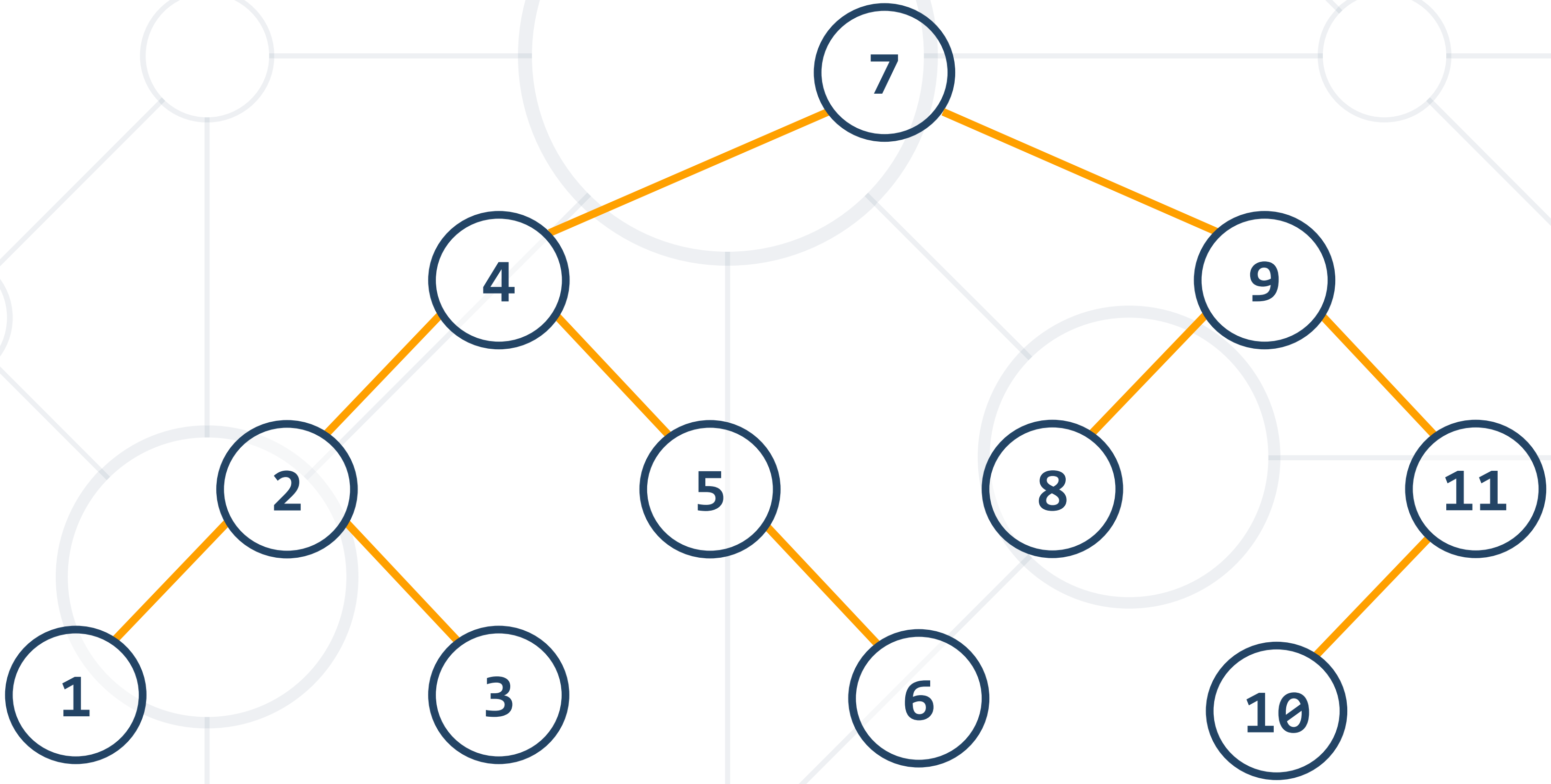


Binary Tree Traversal Algorithms

- Traversing a **binary tree** is similar to traversing normal trees
  - **Pre-Order** Traversal
    - Order -> Root, Left, Right
  - **In-Order** Traversal
    - Order -> Left, Root, Right
  - **Post-Order** Traversal
    - Order -> Left, Right, Root

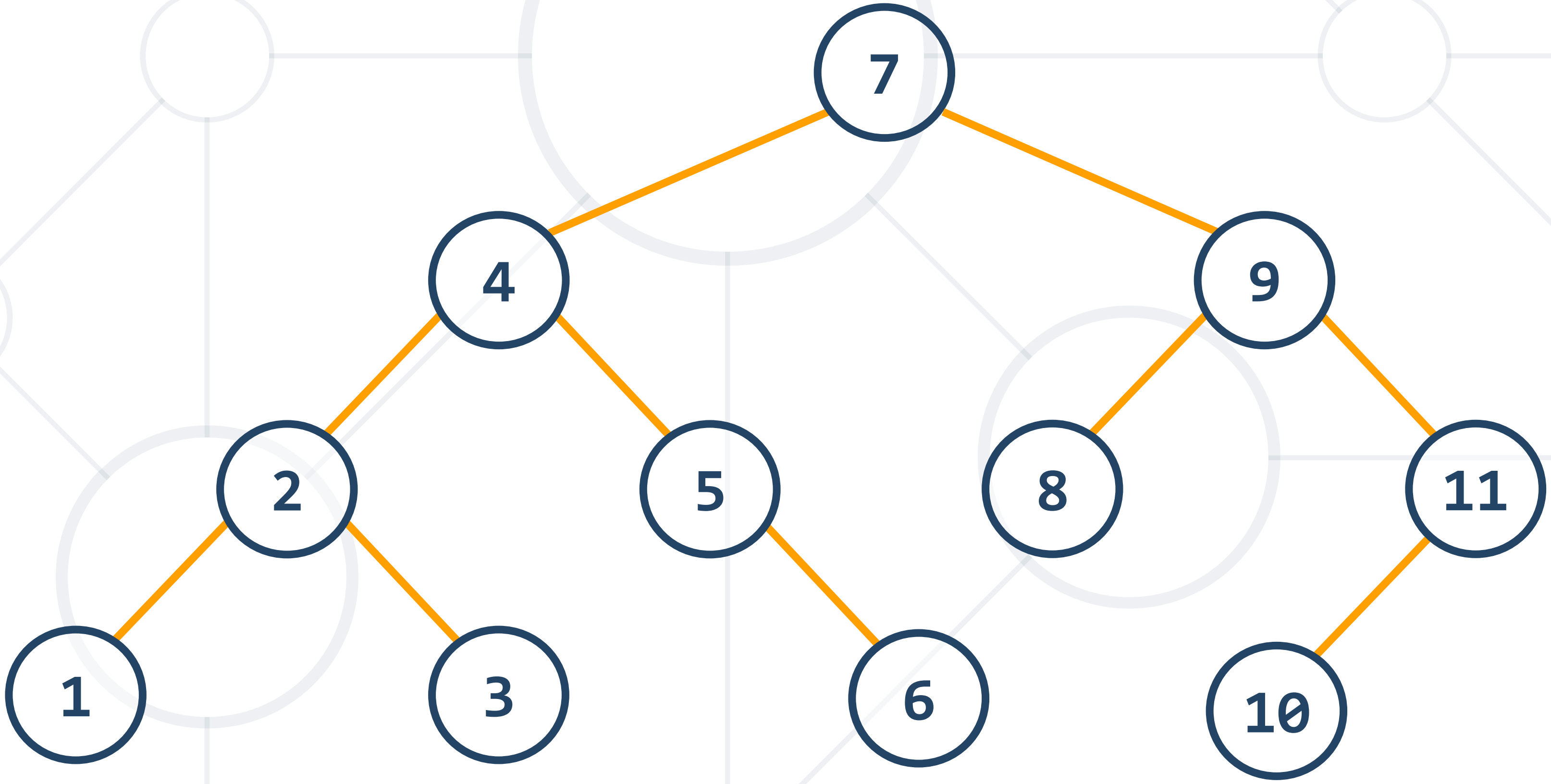
Binary Trees Traversal: Pre-order

Output: 7 4 2 1 3 5 6 9 8 11 10



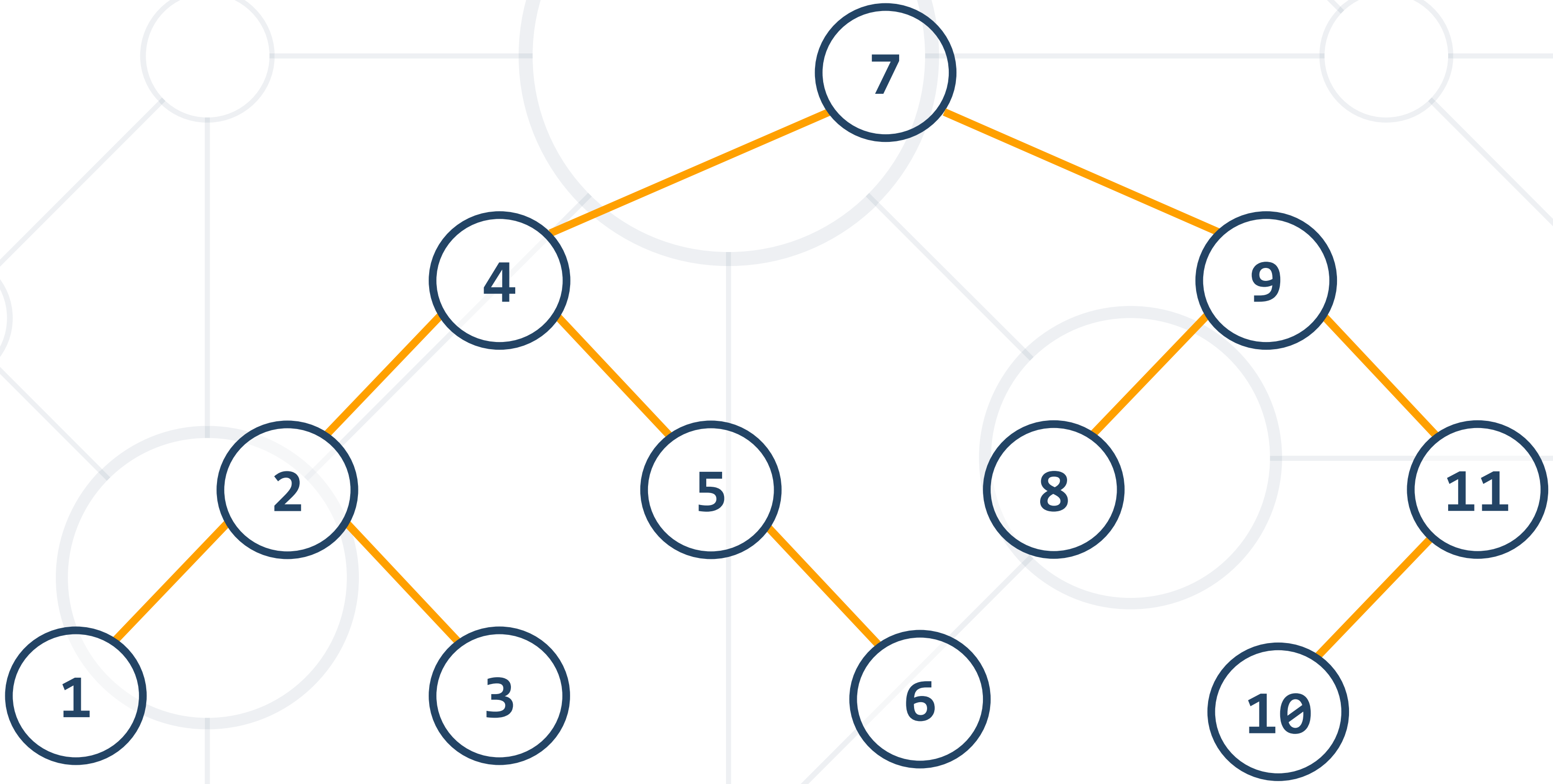
Binary Trees Traversal: In-order

Output: 1 2 3 4 5 6 7 8 9 10 11



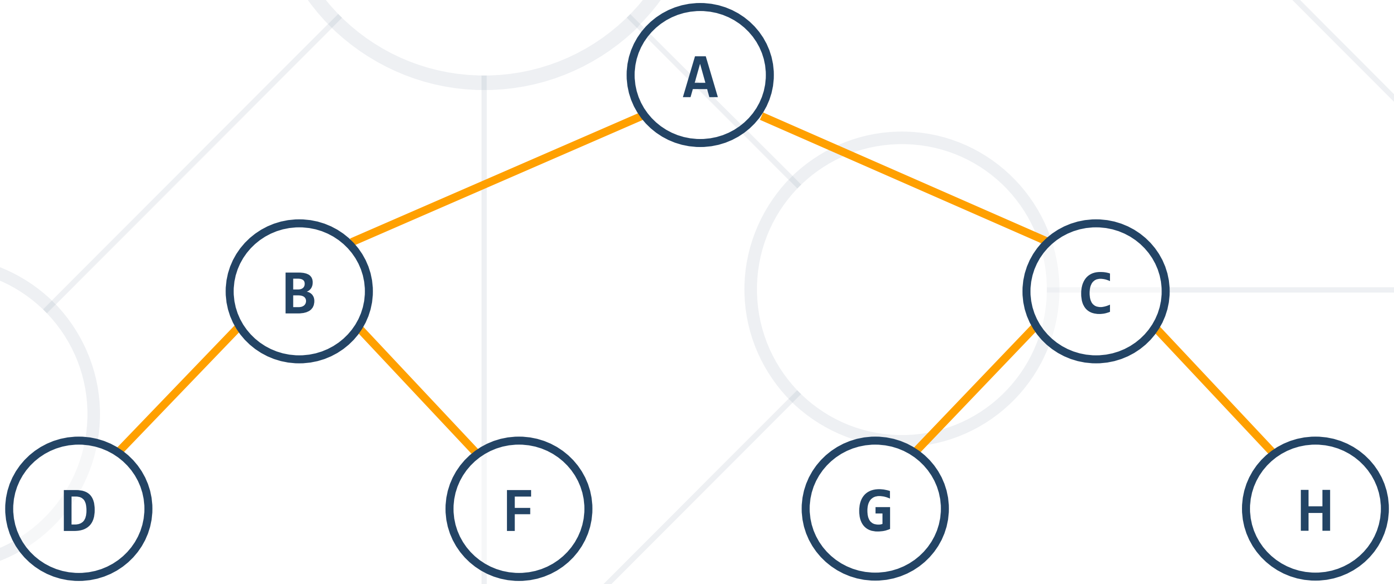
Binary Trees Traversal: Post-order

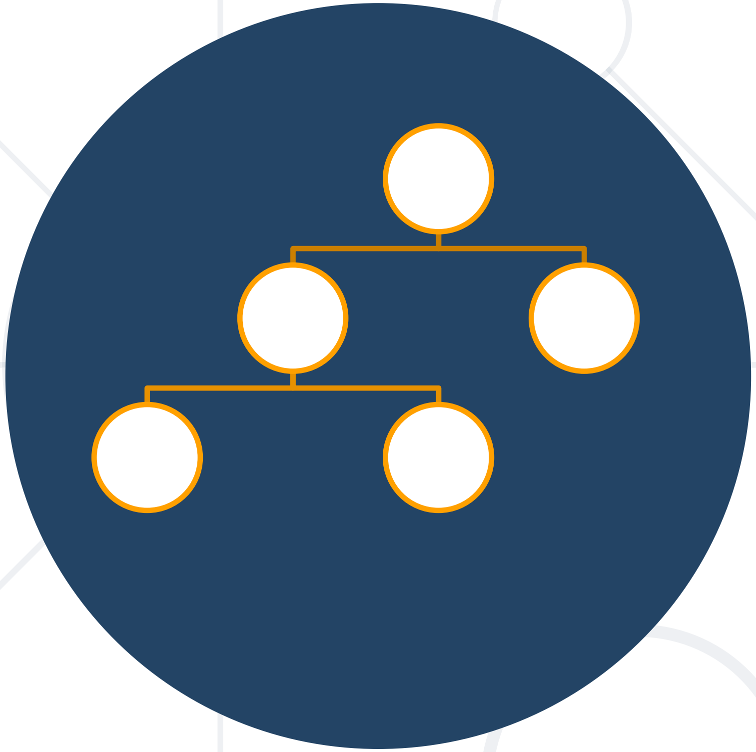
Output: 1 3 2 6 5 4 8 10 11 9 7



Problem: Binary Tree Traversals

- Inside the given skeleton
  - Implement **IAbstractBinaryTree<T>**
  - For more details you can inspect the lab document provided





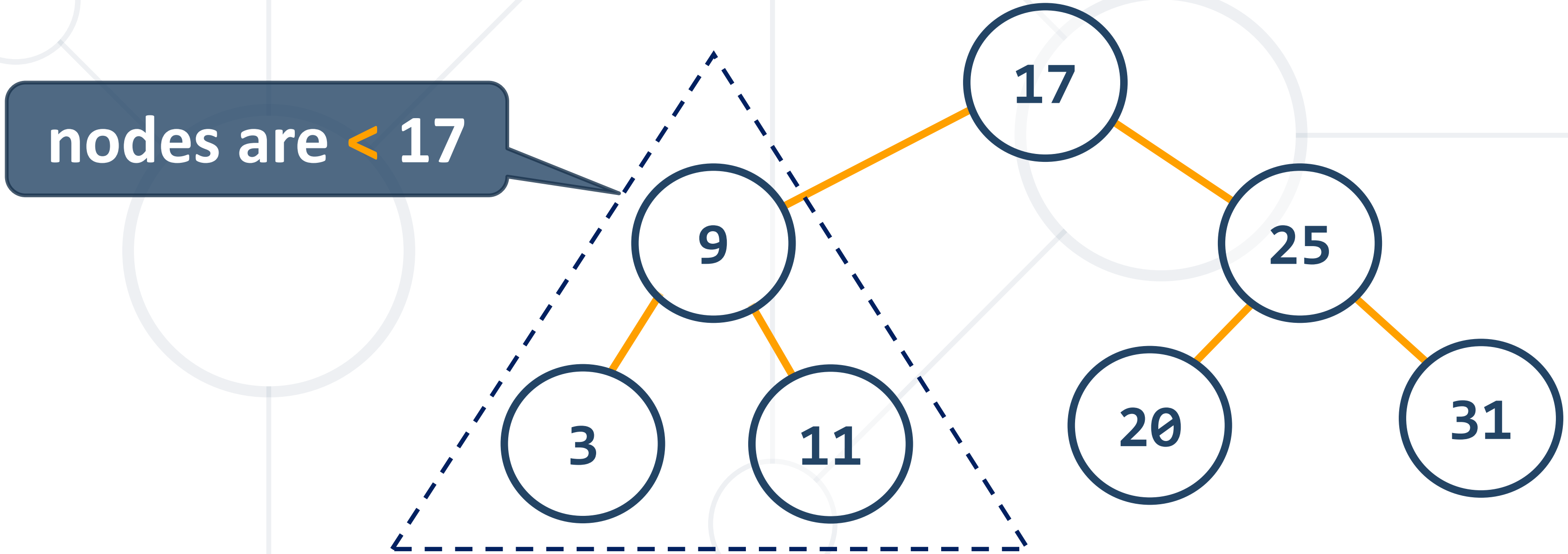
Binary Search Trees

Two Children at Most

Binary Search Trees

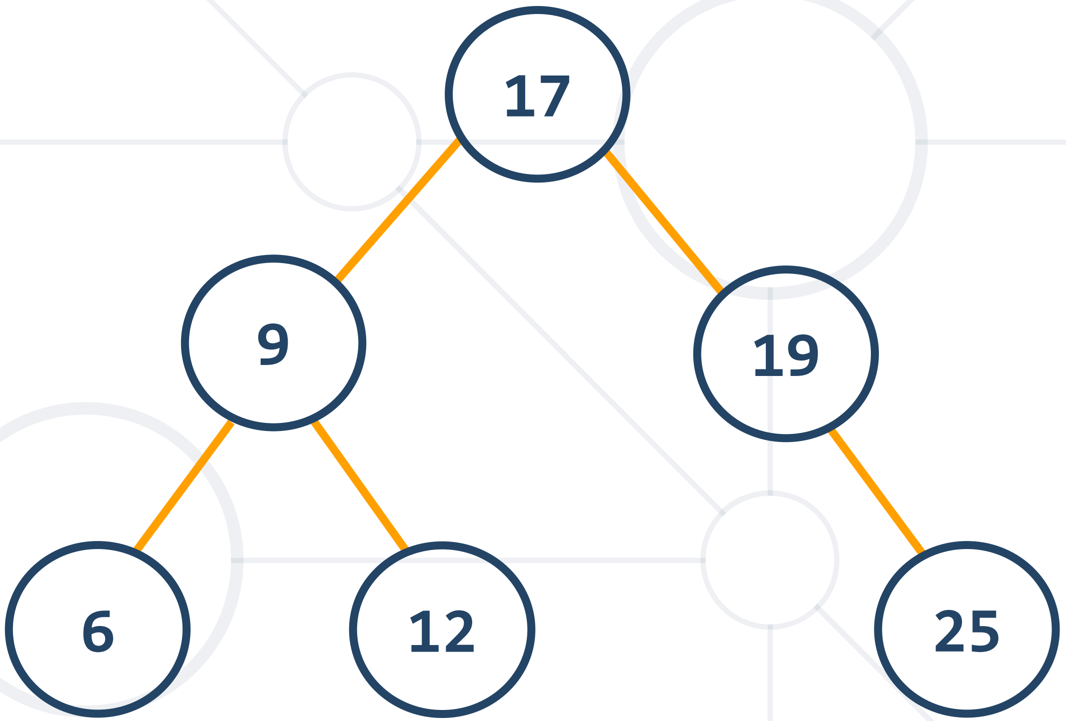
- **Binary search trees** are **ordered**
  - For each node ***x***
    - Elements in left subtree of ***x*** are **<** ***x***
    - Elements in right subtree of ***x*** are **>** ***x***

what about ==



BST - Search

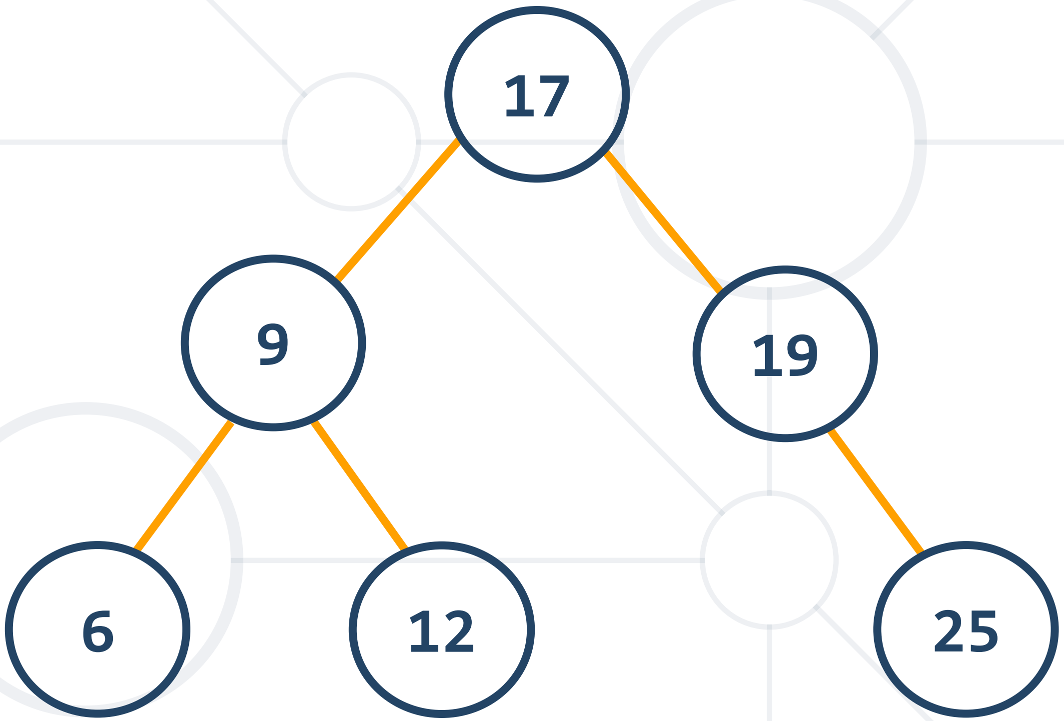
- Search for **x** in BST
  - if the node is not null
    - if x **<** node.value → **go left**
    - else if x **>** node.value → **go right**
    - else if x **==** node.value → **return**




```
Search 12 → 17 9 12
Search 27 → 17 19 25 null
```

BST - Insert

- Insert **x** in BST
  - if node is **null** → insert x
  - else if x **<** node.value → **go left**
  - else if x **>** node.value → **go right**
  - else → node **exists**

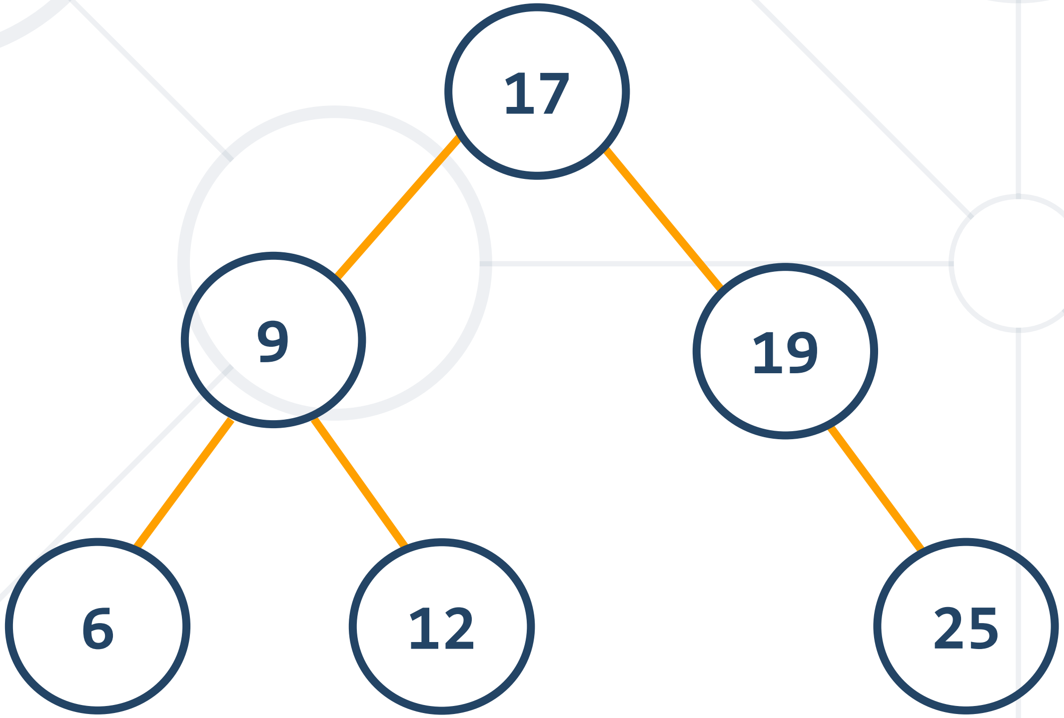


`Insert` **12** → `17 9` **12 return**

`Insert` **27** → `17 19 25` **null(insert)**

Problem: BST

- You are given a skeleton
  - Implement **IAbstractBinarySearchTree<T>**
    - **IAbstractBinarySearchTree <T> Search(T value)**
    - **bool Contains(T element)**
    - **void Insert(T element)**



Solution: BST Contains

```
public bool Contains(T element) {
    Node<T> current = this.Root;
    while (current != null) {
        // TODO: Implement on your own
    }
    return current != null;
}
```

Solution: BST Insert

```
public void Insert(T element) {
    if (this.Root == null) {
        this.Root = new Node<T>(element);
    } else {
        // TODO: Find the place to insert
    }
}
```

Solution: BST Search

```
public IAbstractBinarySearchTree<T> Search(T element) {
    Node<T> current = this.Root;
    // TODO: Find the node with the element
    return new BinarySearchTree<T>(current);
}
```

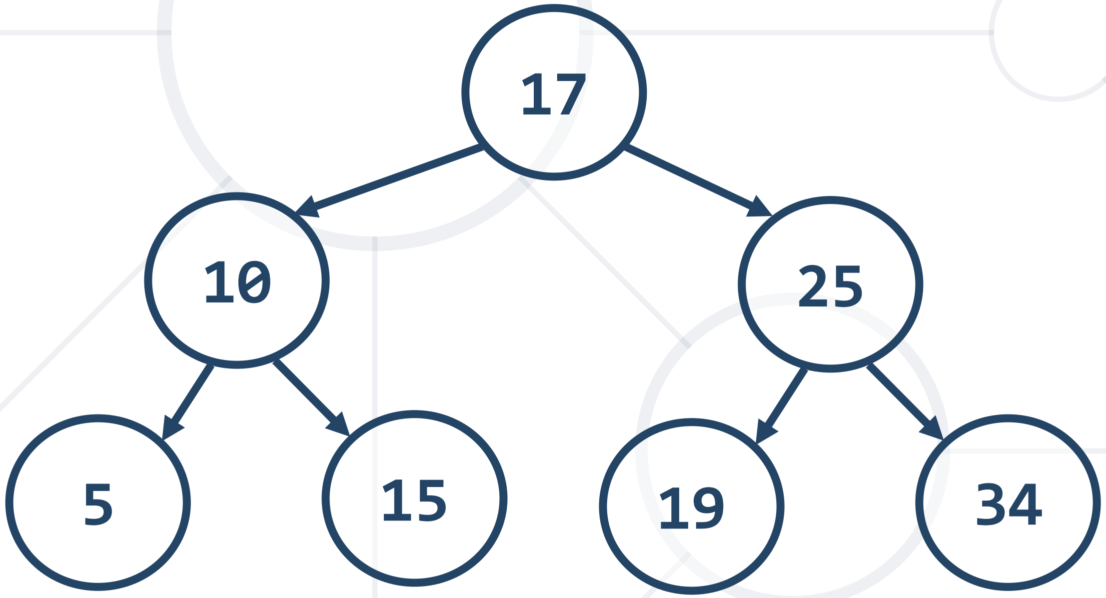
Solution: BST Search (2)

```
public BinarySearchTree(Node<T> root) {
  this.Copy(root);
}

private void Copy(Node<T> node) {
  // TODO: Perform a full copy of the tree
}
```

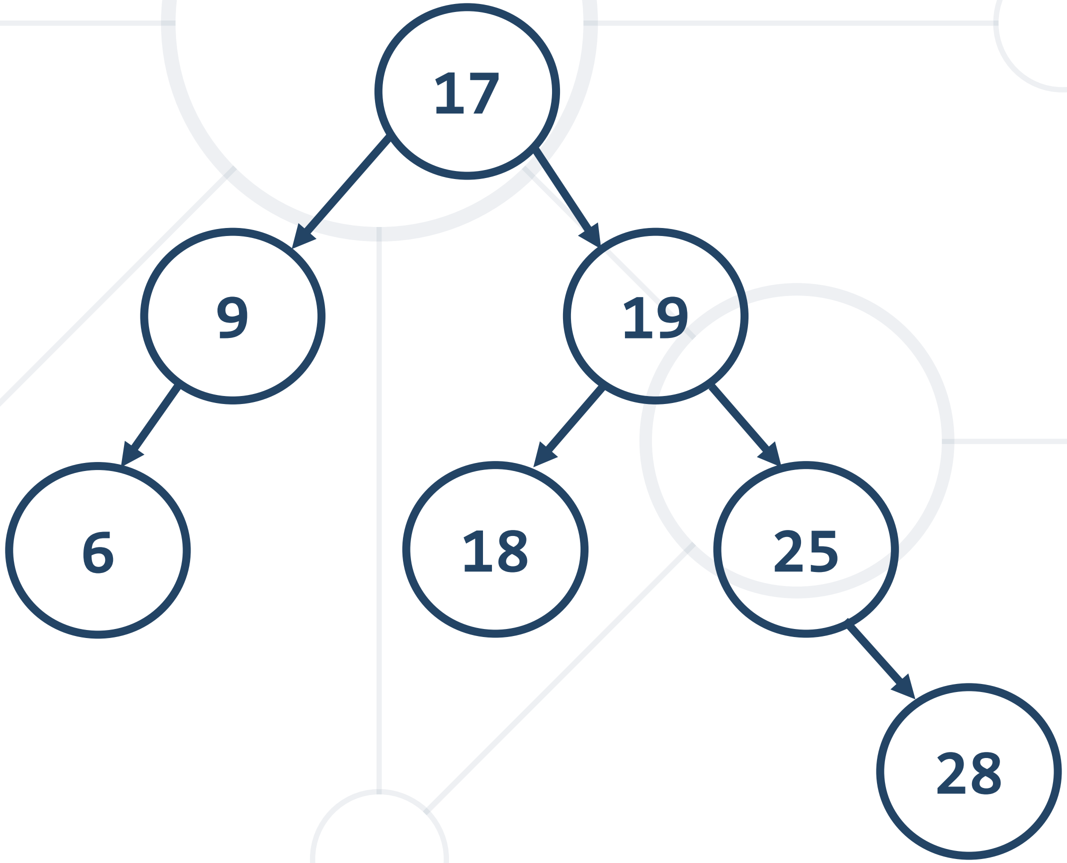
Binary Search Trees – Best Case

- Example: Insert 17, 10, 25, 5, 15, 19, 34



Binary Search Trees – Average Case

- You can insert values in ever **random** order
- Example: Insert 17, 19, 9, 6, 25, 28, 18

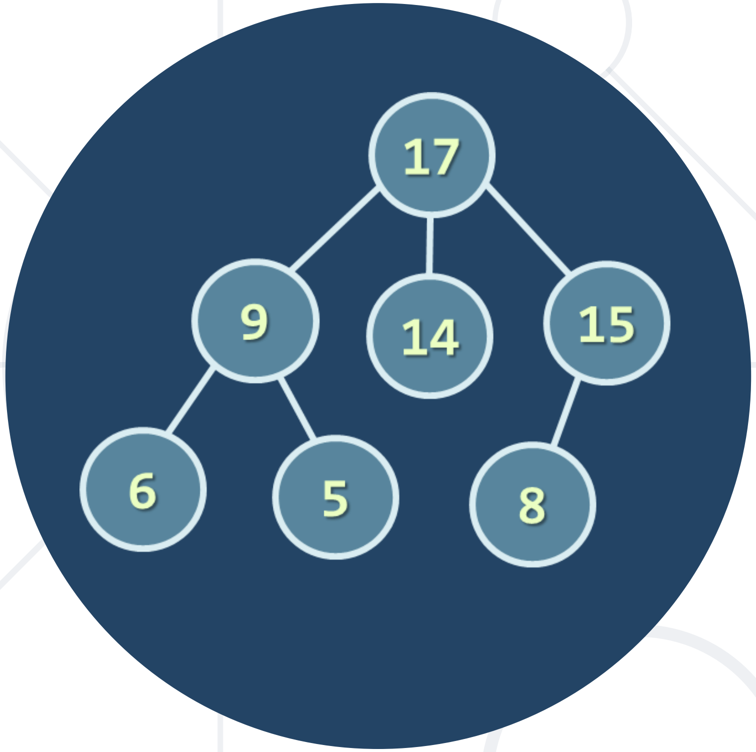


Binary Search Trees – Worst Case

- You can insert values in ever **increasing/decreasing** order
- Example: Insert 17, 19, 25, 34

Balanced Binary Search Trees

- Binary search trees can be **balanced**
  - For each node in BST, there are nearly equal number of nodes in its subtrees
  - **Balanced trees** have **height of ~ log(n)**



Heaps

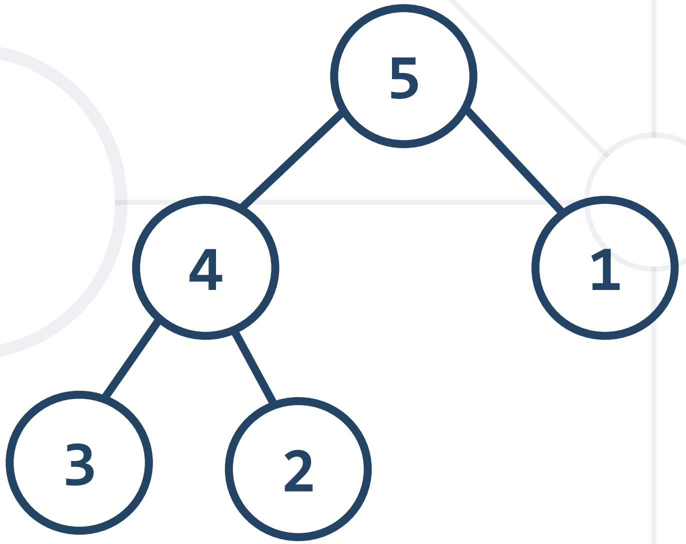
Heap, Binary Heap

What is Heap?

- **Heap**
  - Tree-based data structure
  - Stored in an array
- Heaps hold the **heap property** for each node:
  - **Min Heap**
    - parent ≤ children
  - **Max Heap**
    - parent ≥ children

Binary Heap

- **Binary heap**
  - Represents a Binary Tree
- **Shape property** - Binary heap is a **complete binary tree**:
  - Every level, except the last, is **completely filled**
  - Last is filled **from left to right**



Binary Heap – Complexity Goal

- Unsorted Resizing Array
  - ex.

2	4	1	3	5

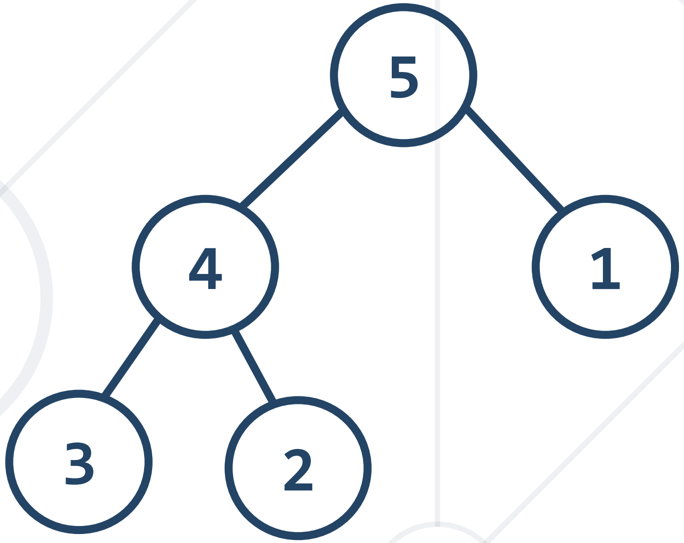
- Sorted Resizing Array
  - ex.

1	2	3	4	5

Operation	Insert	Extract	Peek
Unsorted Array	O(1)	O(N)	O(N)
Sorted Array	O(N)	O(1)	O(1)
Goal	O(logN)	O(logN)	O(1)

Binary Heap – Array Implementation

- Binary heap can be efficiently stored in an array
- **Parent(i)** = (i - 1) / 2
- **Left(i)** = 2 * i + 1;
- **Right(i)** = 2 * i + 2



heap and **shape** properties are satisfied

5	4	1	3	2
0	1	2	3	4

Heap Insertion

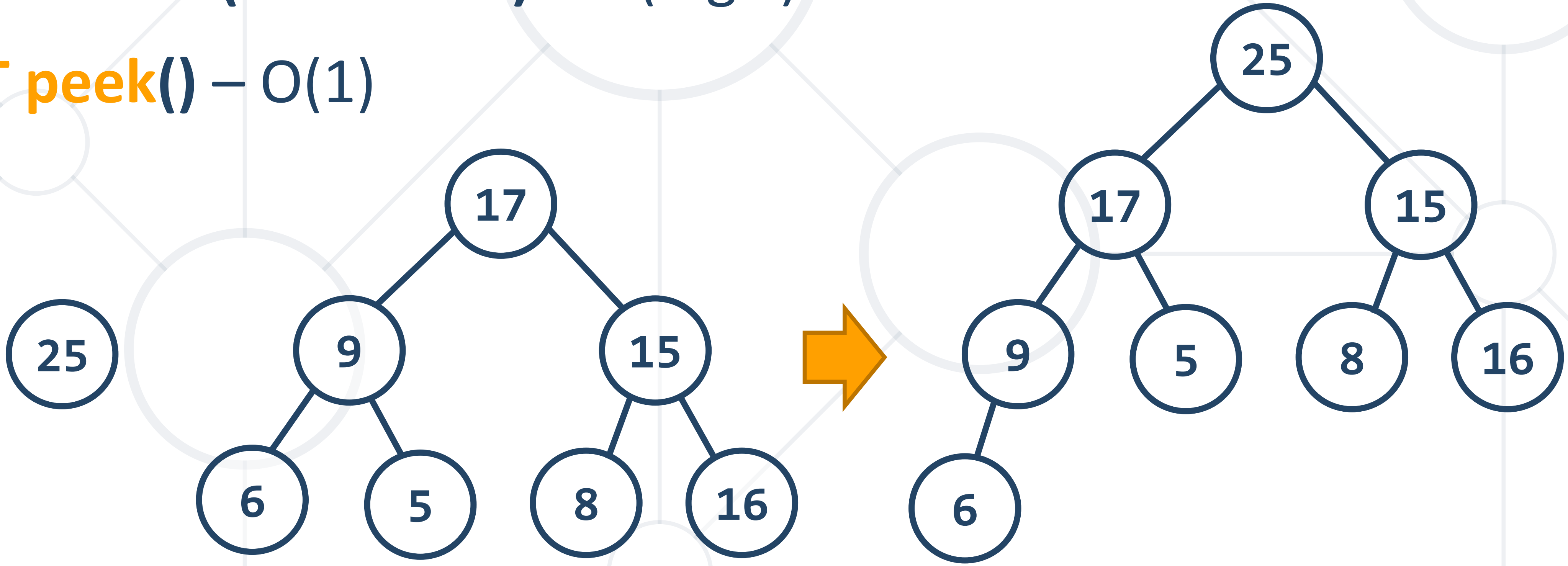
- To preserve **heap properties**:
  - **Insert** at the end
  - **Heapify** element up

Promote while element > parent

- Right: Max Heap
  - Insert 16
  - Insert 25

Problem: Heap Add and Peek

- Implement a max **MaxHeap<T>** with:
  - **int Size**
  - **void Add(T element)** – O(logN)
  - **T peek()** – O(1)



Solution: Heap Add and Peek (1)

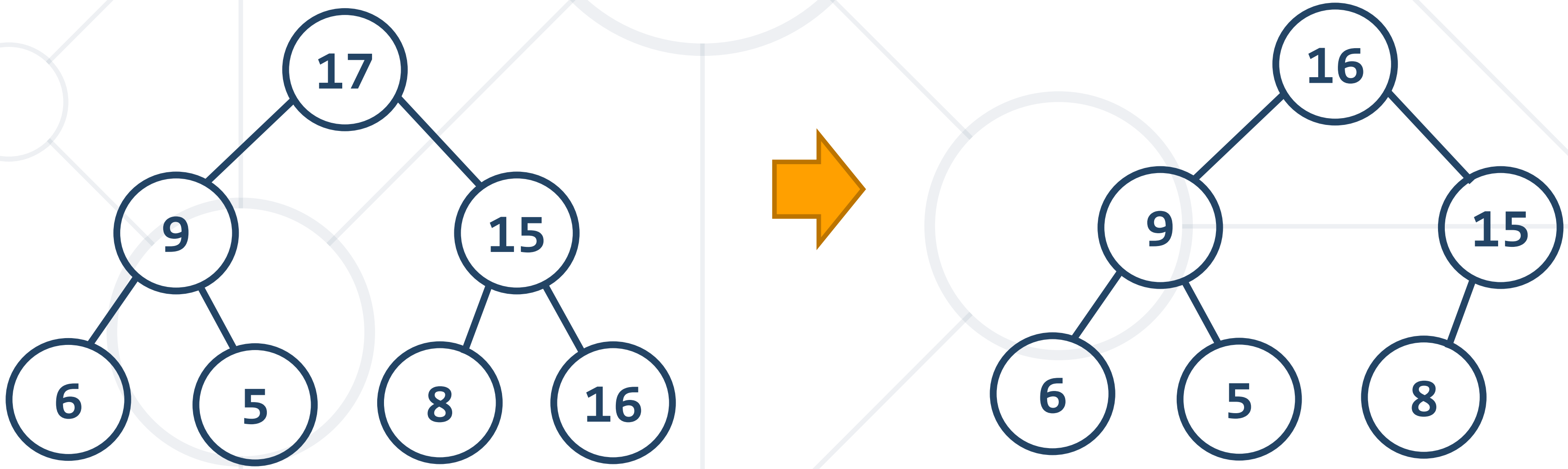
```
public class MaxHeap<T> : IAbstractHeap<T>
  where T : IComparable<T>
{
    // TODO: store the elements
    public void Add(T element)
    {
        this.elements.Add(element);
        this.HeapifyUp(this.Size - 1);
    }
}
```

Solution: Heap Add and Peek (2)

```
private void HeapifyUp(int index)
{
    int parentIndex = this.GetParentIndex(index);
    while (index > 0 && IsGreater(index, parentIndex)) {
        this.Swap(index, parentIndex);
        index = parentIndex;
        parentIndex = this.GetParentIndex(index);
    }
}
//TODO: Implement GetParentIndex(), IsGreater() and Swap()
```

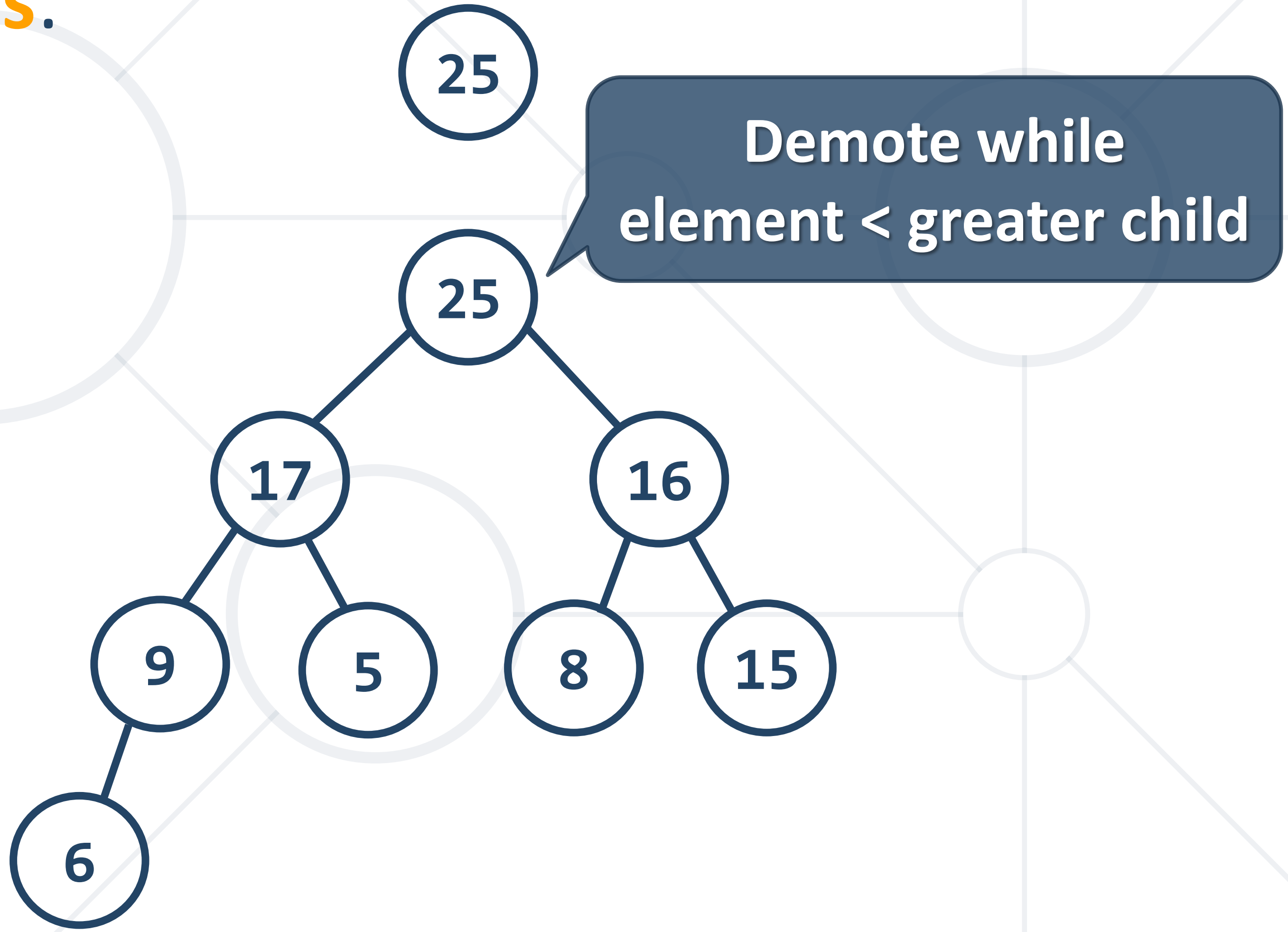
Problem: Heap ExtractMax

- Using your **MaxHeap<T>** implement:
  - **T ExtractMax()** – O(log(N))



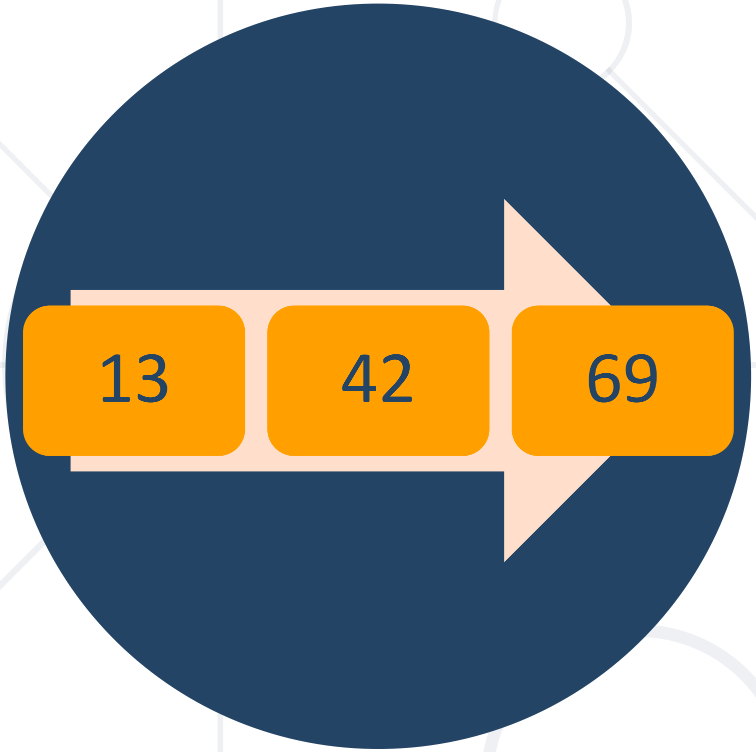
Problem: Heap ExtractMax

- To preserve **heap properties**:
  - **Save** first element
  - **Swap** first with last
  - **Remove** last
  - **Heapify** first down
  - **Return** element
- Right: Max Heap
  - **ExtractMap** – returns 25



Solution: Heap ExtractMax

```
public T ExtractMax()
{
    this.ValidateIfNotEmpty();
    T element = this.elements[0];
    this.Swap(0, this.Size - 1)
    this.elements.RemoveAt(this.Size - 1);
    this.HeapifyDown(0);
    return element;
}
```

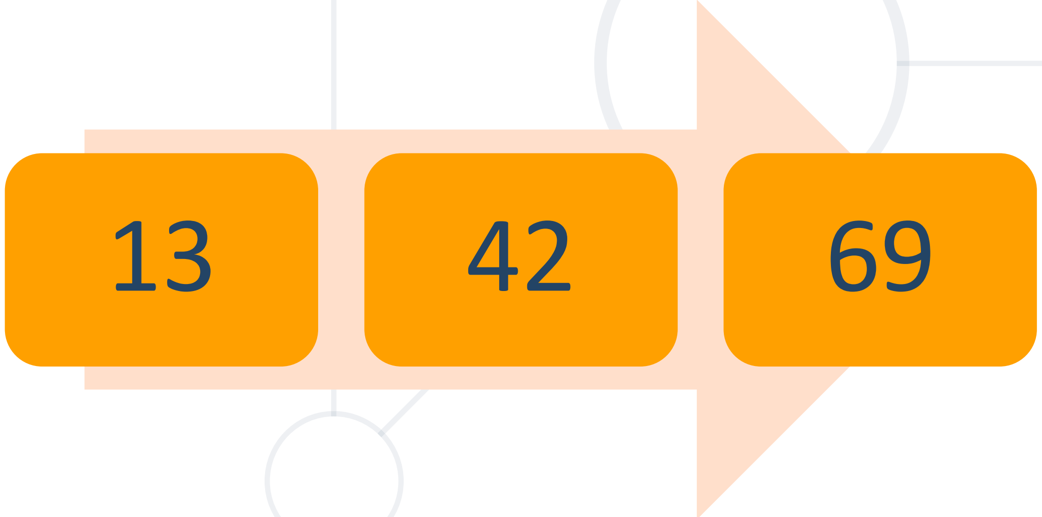


Priority Queue

Dequeue Most Significant Element

Priority Queue

- **PriorityQueue** ADS
  - Each element is served in **priority**
  - High priority is served **before** low priority
  - Elements with **equal** priority are served in **order** of **input**



Priority Queue Order

- Retains a **specific order** to the elements
- **Higher priority** elements are **pushed to the beginning** of the queue
- **Lower priority** elements are **pushed to the end** of the queue

Priority Queue C#

- In C# usually the priority is passed as comparator
  - E.g., **IComparable<T>**

```
public class PriorityQueue<T> : IAbstractHeap<T>
  where T : IComparable<T> {
      ...
}
```

Summary

- **Binary** trees have **up to 2** children
- **Heaps** are used to **implement priority** queues
- Binary Heaps have tree-like structure
- Priority Queues have **wide application**

Questions?

Software University

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

License

Software University

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg