TRAVERSAL AND SEARCHING TECHNIQUES

Understanding the Fundamentals with Practical Applications

BATCH-(2024-2026)

THE ICFAI University,H.P

Submitted to:

Ms.Vandana

(Assistant Professor)

Submitted by:

Nikita Kumari

MCA,II sem

OVERVIEW

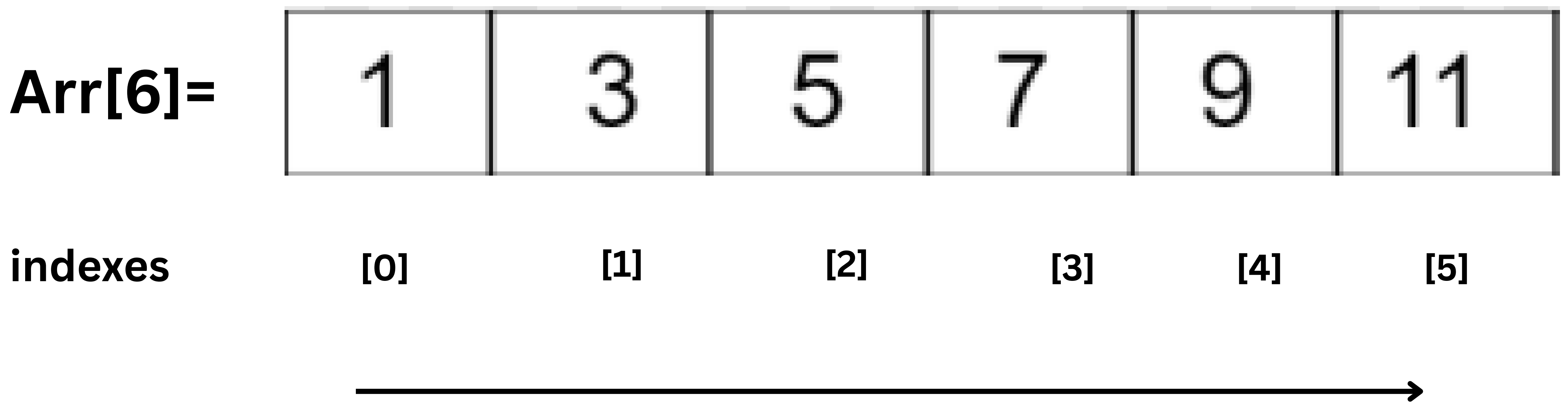
- Introduction
- Types of Traversals
- Types of searching
- Binary Search
- Tree Traversal
- Graph Traversal
- Conclusion

INTRODUCTION

WHAT IS TRAVERSAL?

- Visiting each item/element/node in data structure one by one.
- .   Essential for searching, sorting, and modifying data.
- Traversal is the action.
- Example: Reading a book page by page.

EXAMPLE



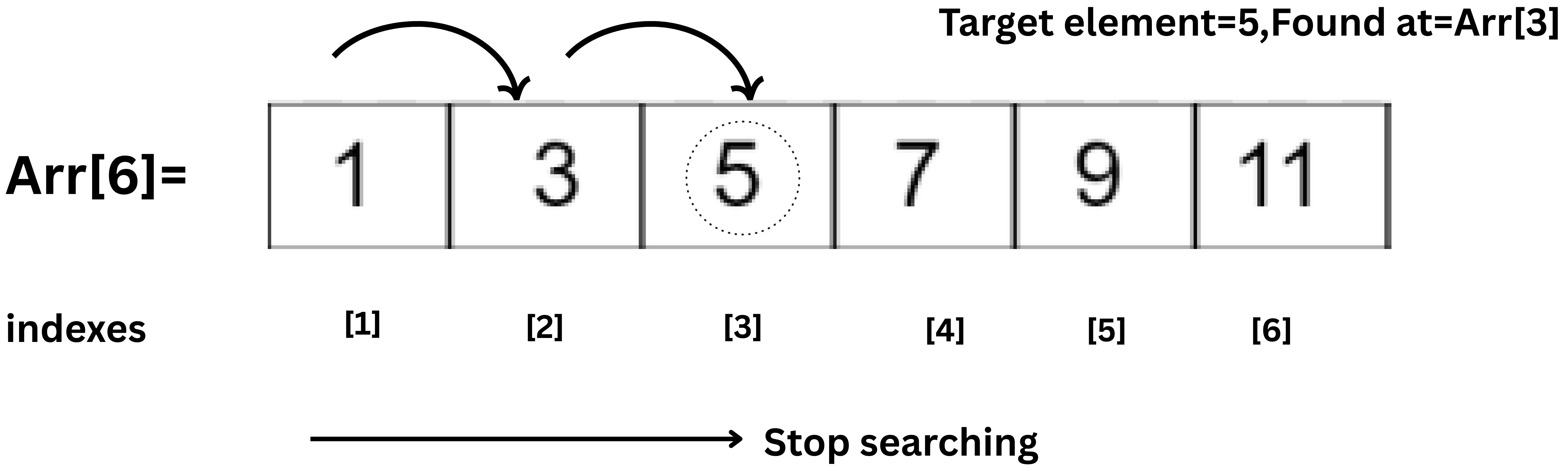
"If indexing starts from 0, it goes from 0 to (n - 1);

if indexing starts from 1, it goes from 1 to n."

WHAT IS SEARCHING?

- Finding a specific item in a data structure.
- Searching depends on data organization and how efficient it is.
- Searching is a goal.
- . Example: Looking for a word in a dictionary.

EXAMPLE



"If indexing starts from 0, it goes from 0 to (n - 1);

if indexing starts from 1, it goes from 1 to n."

TYPES OF TRAVERSING

- **Linear Traversal** **check elements one by one.**
  - **Forward Traversal** (from start to end)
  - **Backward Traversal** (from end to start)

example: (like array,linked list)

TYPES OF TRAVERSING

- **Non-Linear Traversal**

  non-linear data structures (like trees and graphs) are not traversed in a single straight line.

  1.Tree traversal:

  In trees, each node can have multiple child nodes, so we use different orders to visit the nodes. The main types

  - **Inorder (Left → Root → Right)**
  - **Preorder (Root → Left → Right)**
  - **Postorder (Left → Right → Root)**

  2.Graphs Traversals:

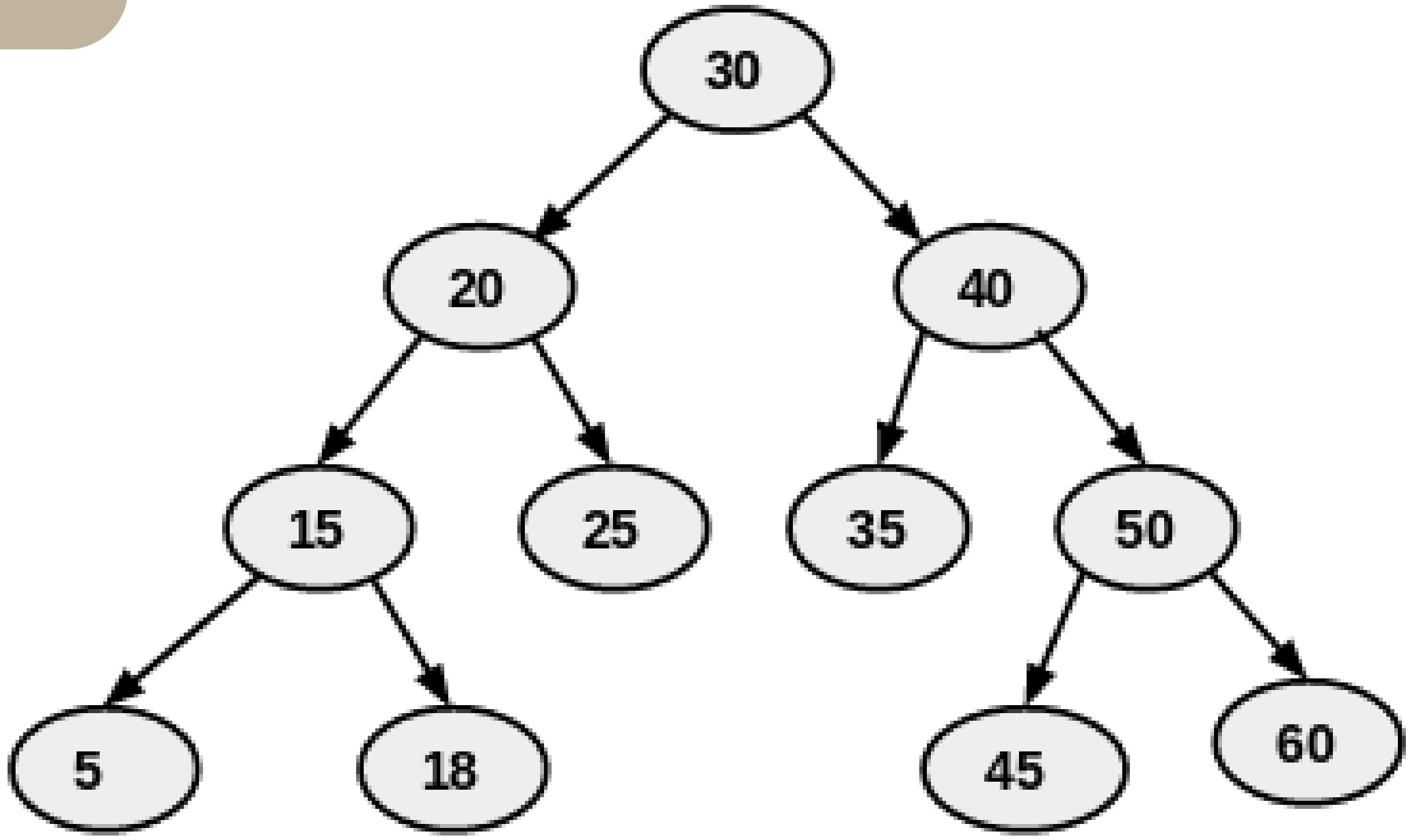
  Graphs can be more complex than trees—they may have cycles and nodes may be connected in any way. Two common ways to explore a graph are:

  - **BFS-(Breadth First search) also called level order.**
  - **DFS-(Depth First search)**

EXAMPLE OF INORDER

Inorder Traversal: Left → Root → Right

- Visit the left subtree
- Visit the root
- Visit the right subtree
- Example Tree:



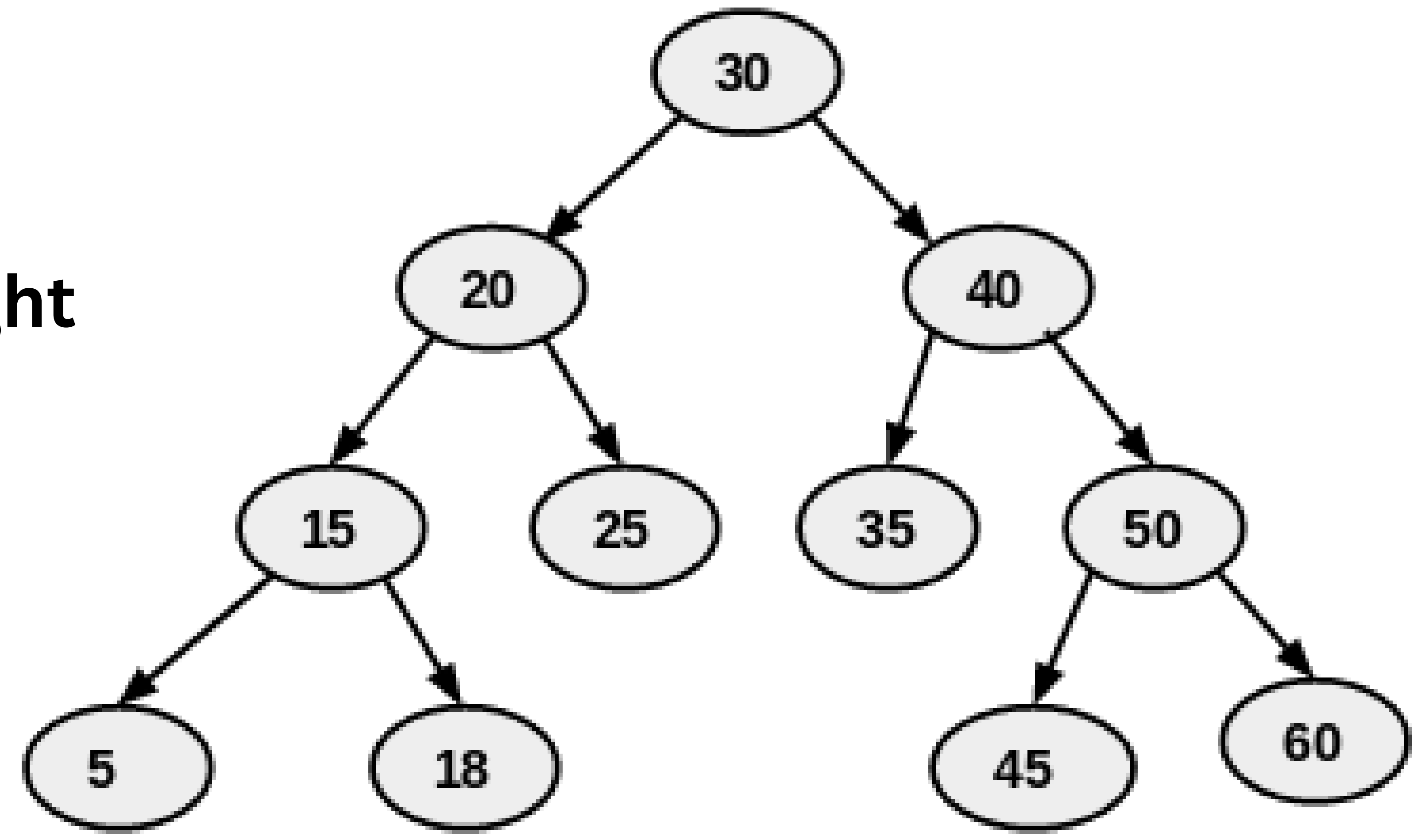
Time Complexity:O(n) for all cases.

Output: [5,15,18,20,25,30,35,40,45,50,60]

EXAMPLE OF PREORDER

Preorder Traversal: Root → Left → Right

- Visit the root.
- Visit the left subtree.
- Visit the right subtree
- Example Tree:



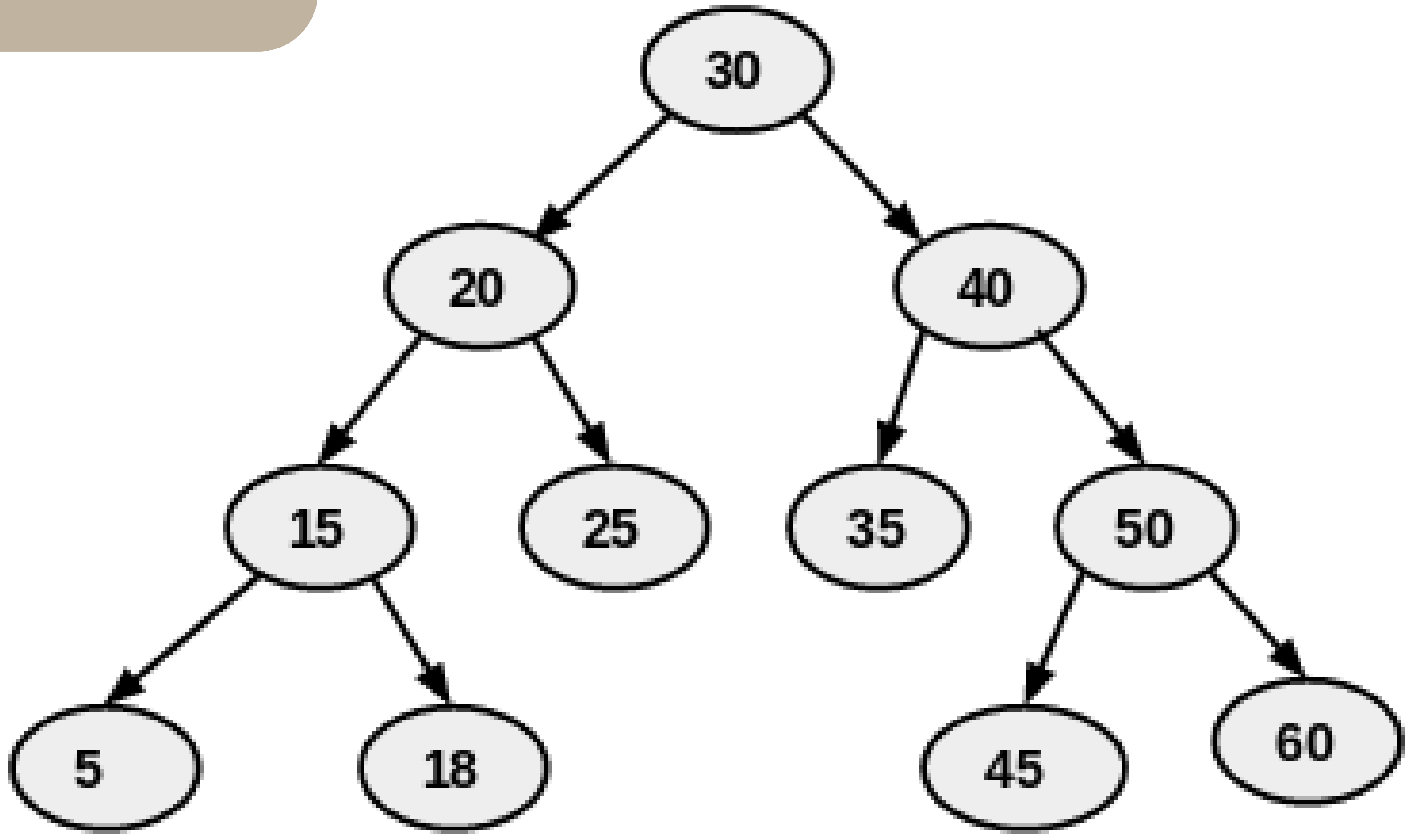
Time Complexity:O(n):

Output: [30,20,15,5,18,25,40,35,50,45,60]

EXAMPLE OF POSTORDER

Postorder Traversal: Left→ Right → Root

- Visit the left subtree.
- Visit the right subtree.
- Visit the root.
- Example Tree:



Time Complexity:O(n)

Output: [5,18,15,25,20,35,45,60,50,40,30]

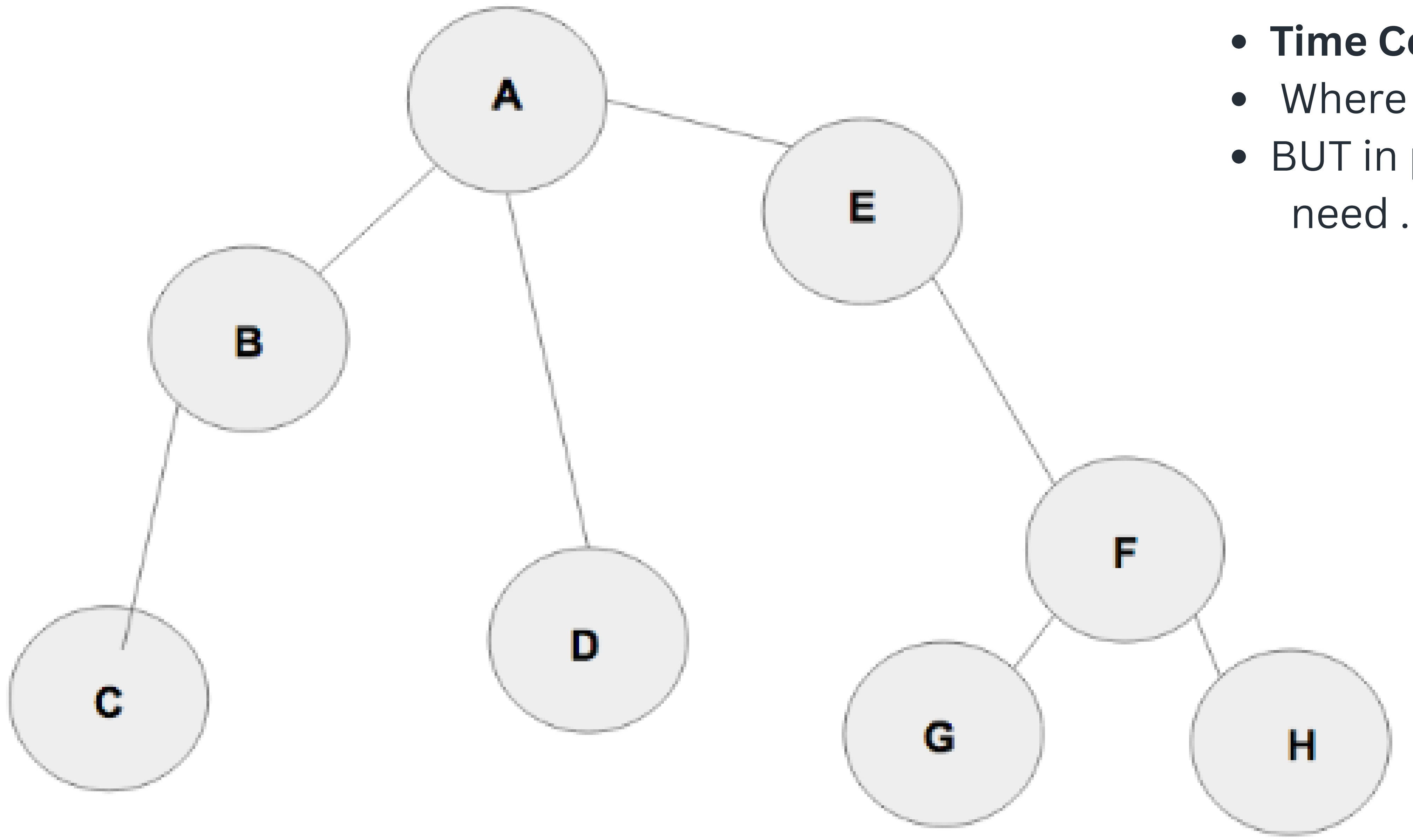
GRAPH TRAVERSAL-BREADTH FIRST SEARCH

- **Defination:**Breadth-First Search (BFS) is a graph or tree traversal algorithm that **explores all the nodes at the present level before moving to the next level.**
- It uses a **queue** to keep track of nodes to visit next.
- **Advantages**:
- level by level exploring.
- It follows FIFO.
- give one or more path solution ,we need to find the optimal solution or shortest path.
- **Disadvantages**:
- Memory usage high(stores all nodes at a level.
- can not work on infinite loop.
- **Real-life example**:
- You check every room on the current floor, then move to the next floor.

Steps to Perform the BFS:

- Start from the root (or starting) node.
- Visit the starting node and mark it as visited.
- Add the starting node to a queue.
- While the queue is not empty:
- Remove the node from the front of the queue.
- Visit all its unvisited neighbors.
- Mark them as visited and add them to the queue.

EXAMPLE OF BFS AND DFS



- Theoretically, both BFS and DFS have the same time complexity:
- **Time Complexity = O(V + E)**
- Where **V = number of vertices (nodes),** and **E = number of edges**.
- BUT in practice, one might be faster or slower depending on need .

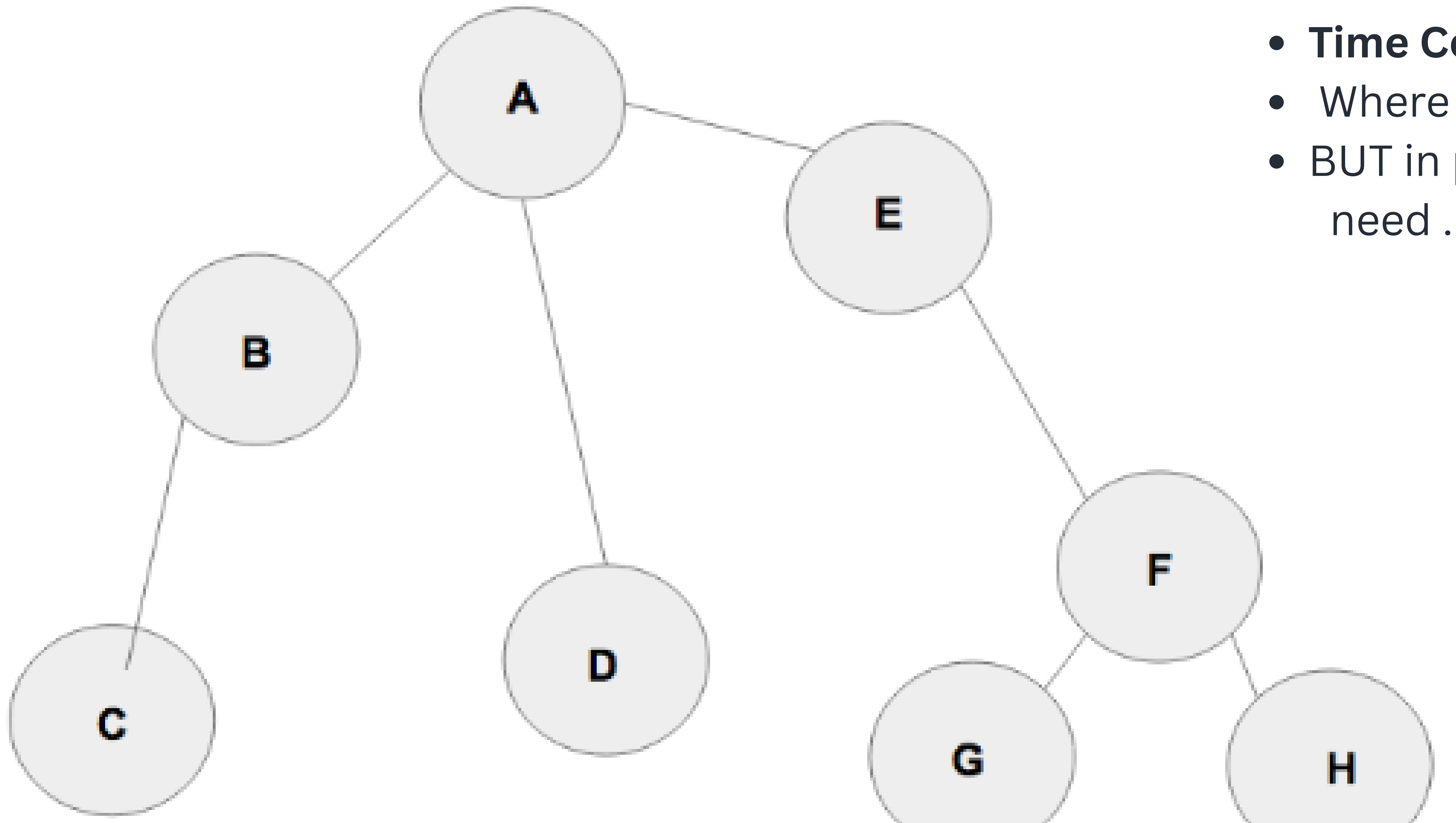
GRAPH-DEPTH FIRST SEARCH

- **Defination:**Depth-First Search (DFS) is a graph or tree traversal algorithm that goes as deep as possible along each branch before backtracking.
- It uses a stack to keep track of the path.
- **Advantages:**
- .Explores deep before backtracking .
- It follows LIFO.
- Memory usage low.(stores only current node address.
- **Disadvantages:**
- shortest path->not guranted.
- can not work on infinite loop.
- **Real Life example:**
- Solving a maze: go down a tunnel until you hit a wall. Then backtrack and try a different tunnel.

Steps to Perform the DFS:

- Start from the root (or starting) node.
- Visit the node and mark it as visited.
- Push the node onto a stack (or use recursive function calls).
- Repeat:
- Look at the node on top of the stack.
- If it has unvisited neighbors:
- Visit the neighbor.
- Mark it as visited and push it to the stack.
- If no unvisited neighbors:
- Backtrack (pop from the stack).
- Stop when the stack is empty (or when the goal is found).

EXAMPLE OF BFS AND DFS



- Theoretically, both BFS and DFS have the same time complexity:
- **Time Complexity = O(V + E)**
- Where **V = number of vertices (nodes),** and **E = number of edges**.
- BUT in practice, one might be faster or slower depending on need .

TYPES OF SEARCHING

- **Searching Techniques**

1.Binary Search

Used on sorted linear data (like arrays).

Divides data into halves to search efficiently.

2.Binary Search tree

A non-linear tree structure.

Allows fast search, insertion, and deletion

- **Inorder (Left → Root → Right)**
- **Preorder (Root → Left → Right)**
- **Postorder (Left → Right → Root)**

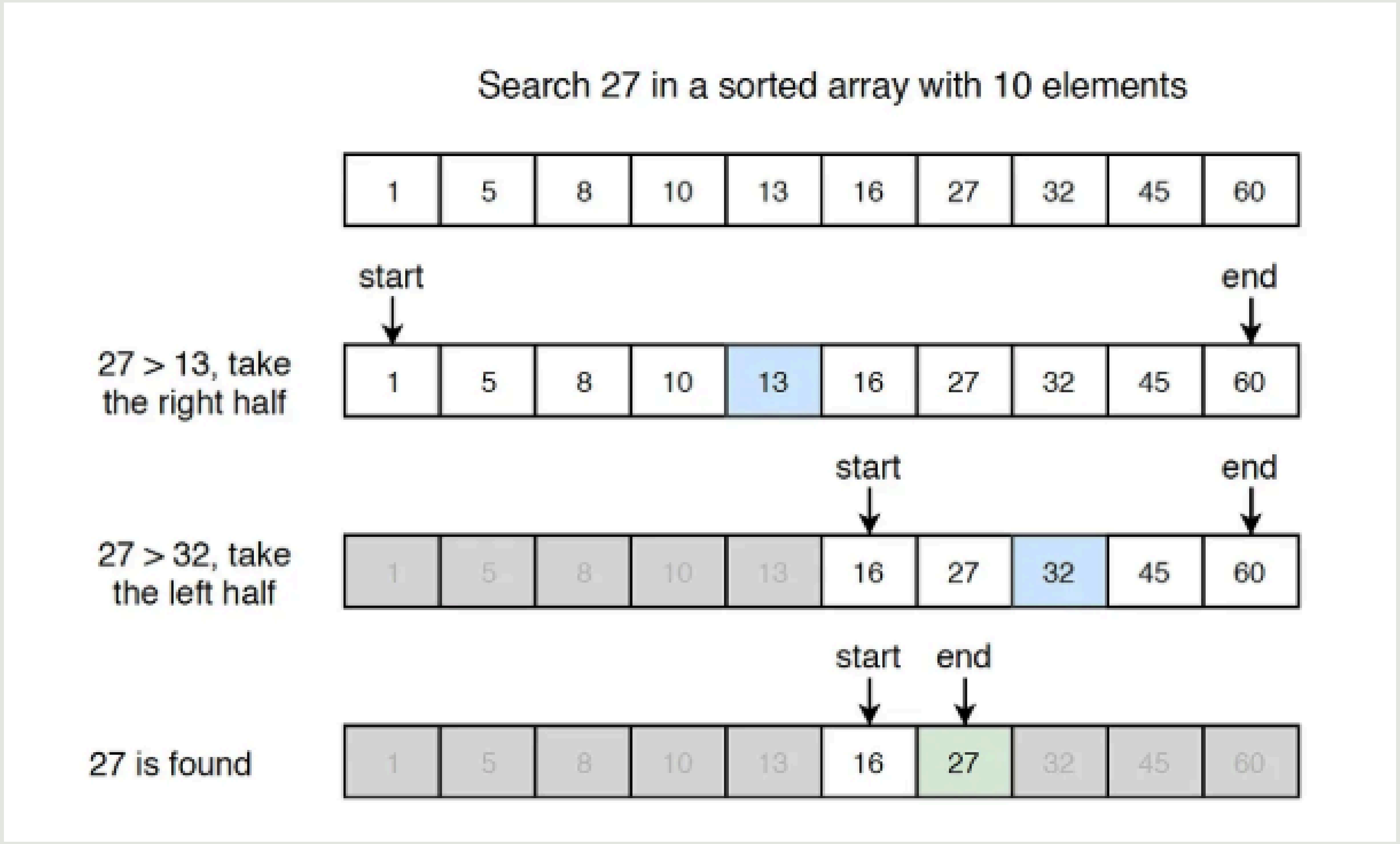
BINARY SEARCH

How it works:

1. It works only on sorted arrays.
2. It repeatedly divides the search interval in half.
3. If the target is equal to the middle element, it returns the index.
4. If the target is smaller than the middle element, it searches the left half.
5. Otherwise, it searches the right half.
6. **Time Complexity:**

- **Best case: O(1)** **When the target element is found at the middle index on the first try.**
- **Average case: O(log n)** **The array is divided in half each time, so it takes $\log_2(n)$ comparisons in the average case.**
- **Worst case: O(log n)** **Even in the worst case (when the element is not present), the array is still halved each time.**

Binary Search Example



Search 27 in a sorted array with 10 elements
1 5 8 10 13 16 27 32 45 60
start
end
27 > 13, take the right half
start
end
27 > 32, take the left half
start
end
27 is found

BINARY SEARCH TREE

- A Binary Search Tree (BST) is a type of binary tree that maintains a special property:
- For every node:
- The left child (and its descendants) contain values less than the node's value.
- The right child (and its descendants) contain values greater than the node's value.

Time complexity:

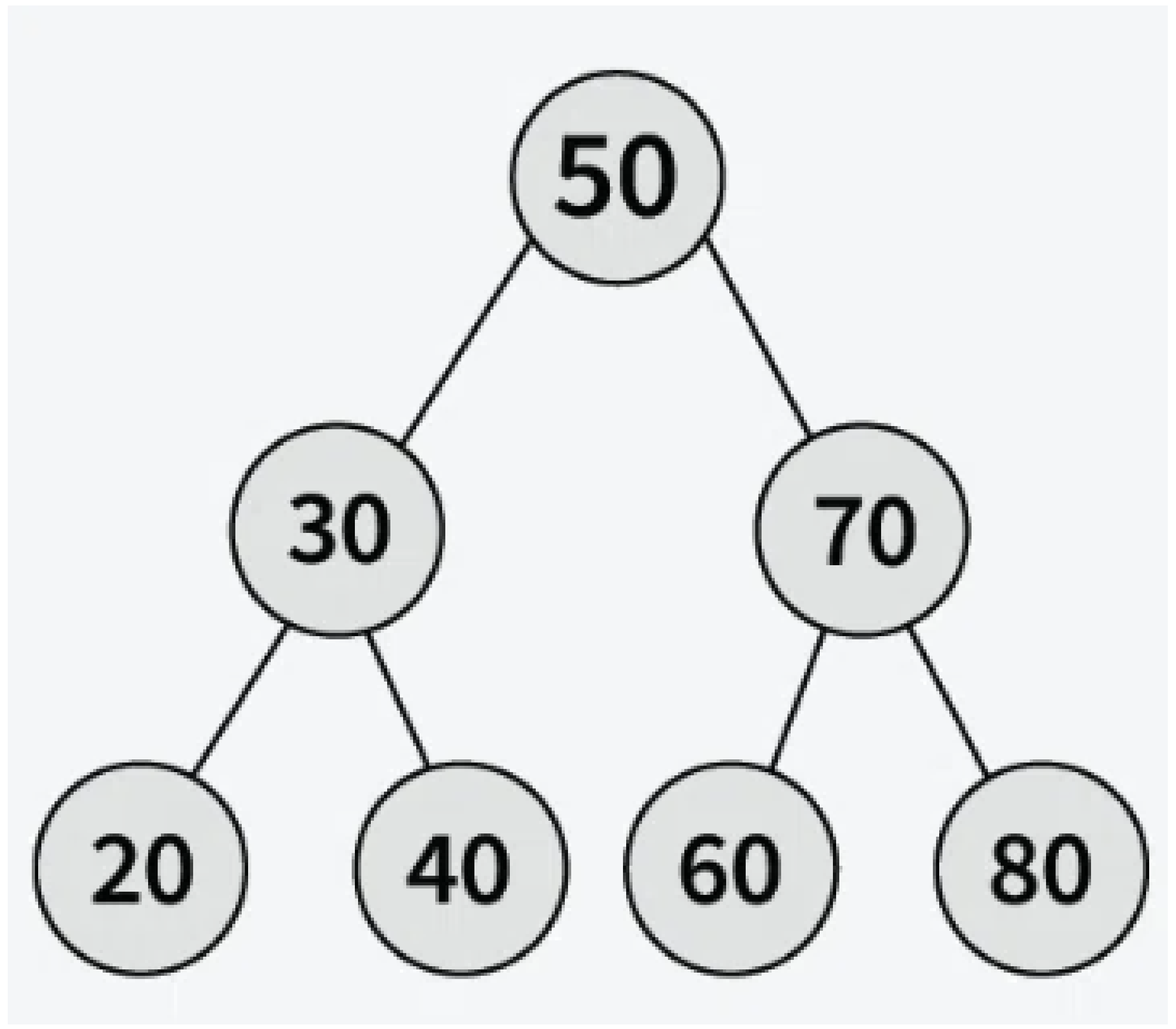
- **Best case:** **O(1)**
- **Average case:** **O(logn)**
- **Worst case: Time complexity:** **O(logn)**

EXAMPLE:

Why is this a BST?

- Left of 50 → all nodes (30, 20, 40) are less than 50
- Right of 50 → all nodes (70, 60, 80) are greater than 50
- Same rule applies to all subtrees.

[50,30,70,20,40,60,80]



EXAMPLE:

EXAMPLE OF SEARCHING

- Let’s say we want to search for 60.
- Steps:
- Start at root → 50
- 60 > 50 → go right
- Now at 70
- 60 < 70 → go left
- Now at 60
- 🎯 found it!

CONCLUSION

This presentation explained traversal and so searching in a simple and practical way. Instead of using hard coding terms, I used real-life examples and easy steps to show how these techniques work. By connecting technical ideas to everyday situations, I made it easier to understand how data is organized, explored, and searched. It helps simplify complex ideas so that it can easy to grasp and apply them.

Thank You