Name: Nouran Sameh Mohamed

Search

1. Complexity

What is Complexity?

"Complexity refers to how the time or space (memory) required by an algorithm grows as the input size increases. It helps us compare different solutions and choose the most efficient one."

Types of Complexity:

1. *Time Complexity: How runtime increases with input size (e.g., O(n), O(log n)).*
2. *Space Complexity: How memory usage increases with input size.*

Why Does Complexity Matter?

"Efficient algorithms save:

- Time (better user experience).
- Resources (cost-effective).
- Scalability (handles large data smoothly)."

Conclusion:

"Understanding complexity helps us write smarter, faster code. By analyzing algorithms, we optimize performance—making technology more powerful and efficient!"

===

2.Big-O

What is Big-O?

"Big-O describes how an algorithm's runtime grows relative to input size (n). It focuses on the worst-case scenario by default (upper bound)."

Best-Case Scenario (Ω):

"Omega (Ω): The fastest possible runtime.

- Example: Linear search finds the target in the first try → Ω(1).
- Rarely used in practice (too optimistic!).

Worst-Case Scenario (O):

"Big-O (O): The slowest possible runtime.

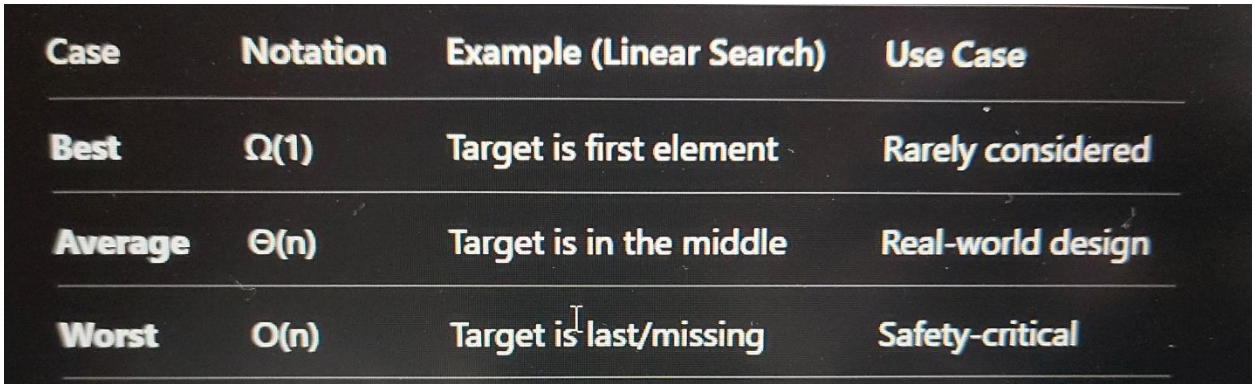
- Example: Linear search checks all elements → O(n).
- Crucial for reliability (e.g., medical systems, aviation).

Average-Case Scenario (Θ):

"Theta (Θ): Expected runtime for random inputs.

- Example: Linear search finds the target halfway → Θ(n/2) ≈ Θ(n).
- Most practical for real-world predictions.

Comparison Table



Case	Notation	Example (Linear Search)	Use Case
Best	Ω(1)	Target is first element	Rarely considered
Average	Θ(n)	Target is in the middle	Real-world design
Worst	O(n)	Target is last/missing	Safety-critical

Real-World Analogy:

"Imagine waiting in line:

- Best-case: You're first! (Ω(1)).
- Average-case: You're in the middle (Θ(n/2)).
- Worst-case: You're last (O(n))."

Why Does It Matter?

"Choosing algorithms based on worst-case (O) ensures reliability, while average-case (Θ) optimizes for everyday use. Best-case (Ω) is like winning the lottery—nice, but don't rely on it!"

Conclusion

"Big-O, Omega, and Theta give us a complete picture of algorithm performance. Always analyze all three to make informed decisions!"

==

3.Search

Types of search:-

1. Linear Search
2. Binary Search
3. Breadth-First Search
4. Depth-First Search
5. Dictionary Search
6. Randomized Search
7. Value-based Search
8. Binary Search Tree Search

Breadth-First Search(BFS)

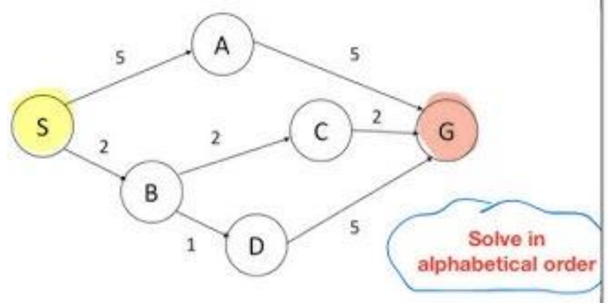
What is BFS?

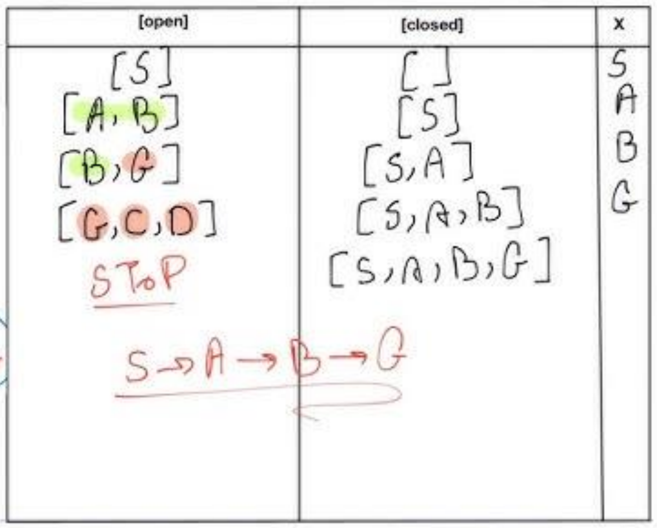
"BFS is a graph traversal algorithm that explores all neighbors at the current depth before moving deeper. It uses a queue to track nodes."

How BFS Works (Step-by-Step):

1. Start at a node (e.g., node A).
2. Visit all direct neighbors (B, C).
3. Then their neighbors (D, E, F), and so on.

Example:





[open]	[closed]	X
[S]	[ ]	S
[A, B]	[S]	A
[B, G]	[S, A]	B
[G, C, D]	[S, A, B]	G
STOP	[S, A, B, G]	

S → A → B → G

BFS Pseudocode :-

```
BFS(start_node):
  queue = [start_node]
  visited = {start_node}
  while queue:
    current = queue.pop(0)
    for neighbor in current.neighbors:
      if neighbor not in visited:
        visited.add(neighbor)
        queue.append(neighbor)
```

Key Features of BFS:

- Uses a queue (FIFO: First-In-First-Out).
- Finds shortest paths in unweighted graphs.
- Time Complexity: O(V + E) (V = vertices, E = edges).

Why Use BFS?

"BFS guarantees the shortest path in unweighted graphs and explores uniformly—perfect for networks, puzzles (e.g., Rubik’s cube), and more!"

Conclusion:

"BFS is a fundamental algorithm for systematic, level-wise exploration. It’s a must-know for coding interviews and real-world problem-solving!"

===

Depth-First Search

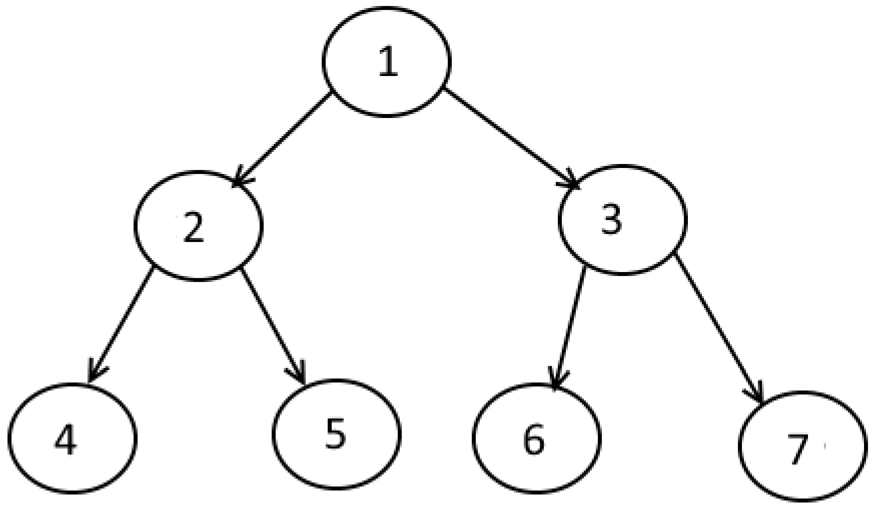
What is DFS?

"DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It uses a stack (or recursion) to track nodes."

How DFS Works (Step-by-Step):

1. Start at a node (e.g., node A).
2. Go as deep as possible (e.g., A → B → D → F).
3. Backtrack when no new nodes are found (e.g., F → D → B → E).

Example



DFS Traversal - 1 2 4 5 3 6 7

DFS Pseudocode:

```
DFS(start_node):
   stack = [start_node]
   visited = {start_node}
   while stack:
      current = stack.pop()
      for neighbor in current.neighbors:
         if neighbor not in visited:
```

```
        visited.add(neighbor)
        stack.append(neighbor)
```

Key Features of DFS:

- Uses a stack (LIFO: Last-In-First-Out) or recursion.
- Memory efficient (only stores current path).
- Time Complexity: O(V + E) (V = vertices, E = edges).

DFS Variants:

1. Preorder: Visit node before children (A → B → D → E → C → F).
2. Inorder (for trees): Left → Root → Right.
3. Postorder: Visit node after children (D → E → B → F → C → A).

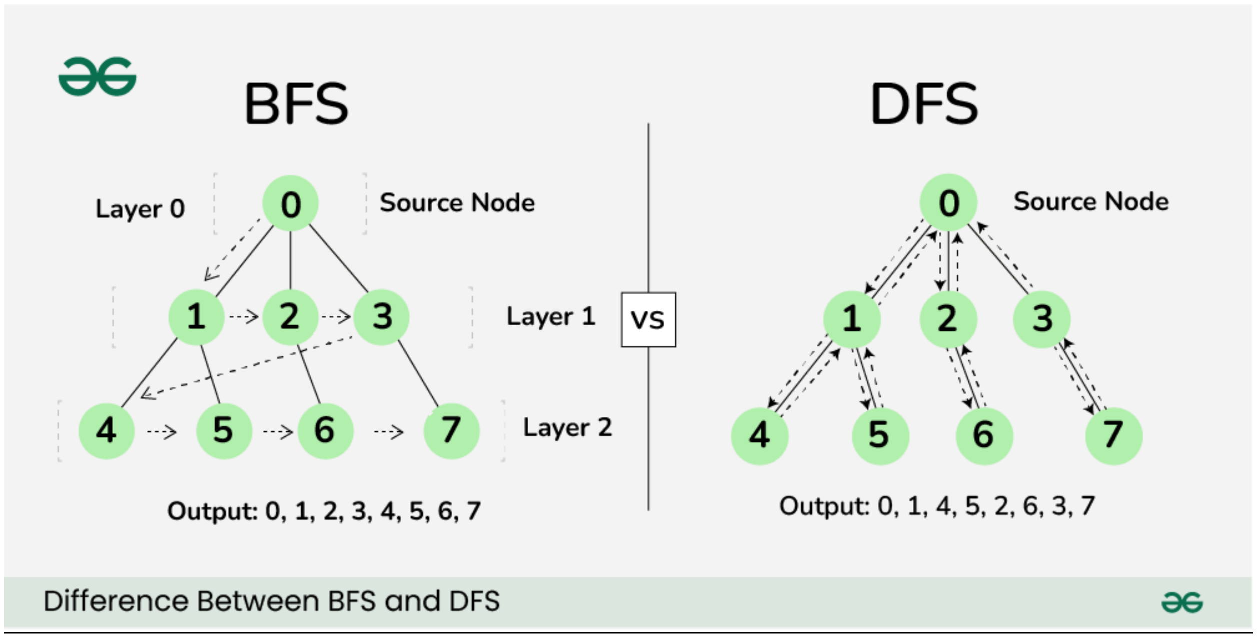
Why Use DFS?

"DFS is ideal for deep graphs, pathfinding (e.g., puzzles), and scenarios where memory is limited!"

Conclusion:

"DFS is a powerful, memory-efficient algorithm for deep exploration. Mastering it unlocks solutions to complex problems!"

BFS vs. DFS


BFS
DFS
Layer 0
Source Node
Layer 1
Layer 2
vs
0
1
2
3
4
5
6
7
Output: 0, 1, 2, 3, 4, 5, 6, 7
Source Node
Output: 0, 1, 4, 5, 2, 6, 3, 7
Difference Between BFS and DFS

Thank you