BASIC CONCEPTS IN DATA STRUCTURES AND ALGORITHMS

CONTENT

I DSA, ADT, STACK VS QUEUE

II TWO SORTING ALGORITHMS

III TWO NETWORK SHORTEST PATH ALGORITHMS

I

DSA, ADT, STACK VS QUEUE

A INTRODUCTION TO DSA

Definition: Data Structures and Algorithms (DSA) is a fundamental area of computer science that focuses on organizing and manipulating data efficiently. It forms the backbone of problem-solving in software development, optimizing code, and making applications run faster and smoother.

Data Structure

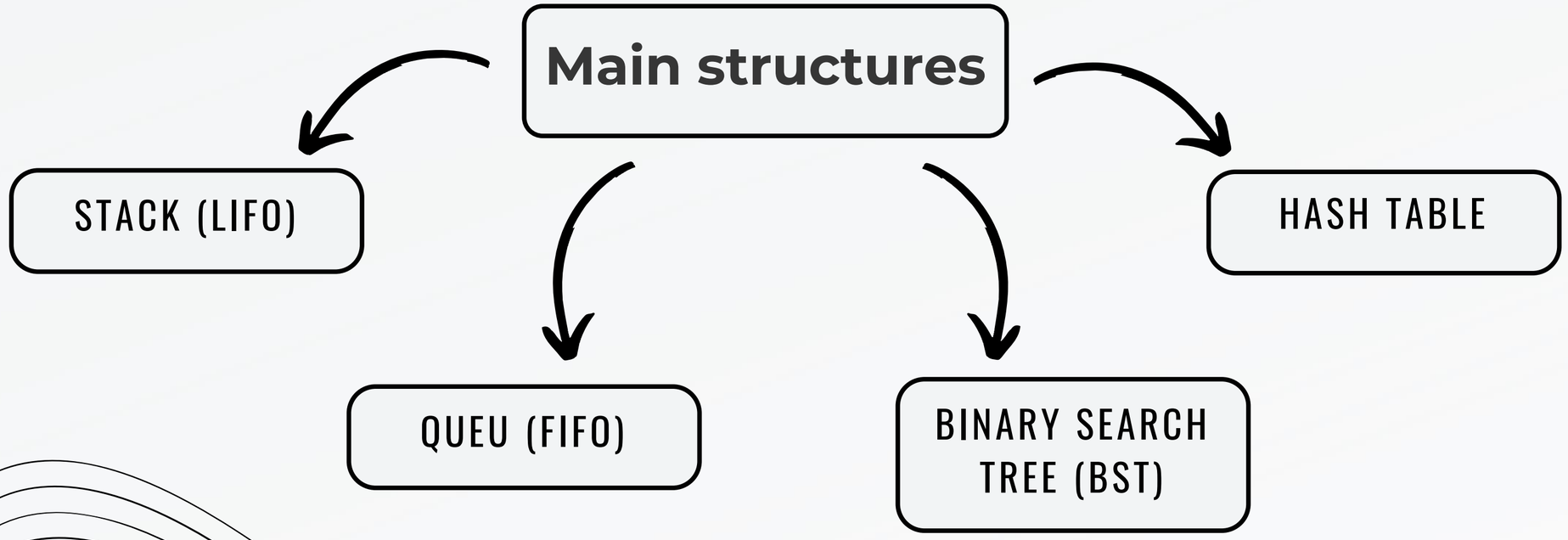
Data Structures are ways to organize and store data for easy access and management.

Algorithm

Algorithms are steps or formulas to solve a problem. Choosing the optimal algorithm helps process data faster and more efficiently

1 DATA STRUCTURES

Data structures: Are ways of organizing and storing data for efficient processing. They play an important role in optimizing the performance of algorithms and programs.



2 ALGORITHMS

Algorithms: are step-by-step procedures or formulas for solving problems. They can be categorized based on their purpose and the types of problems they solve

Main categories

- Sorting algorithms (e.g. Bubble Sort, Quick Sort).
- Search algorithms (e.g. Linear Search, Binary Search).
- Dynamic programming (e.g. Fibonacci, Knapsack problem).
- Graph algorithms (e.g. Dijkstra, DFS, BFS).
- Greedy algorithms (e.g. Prim, Huffman Coding).

B INTRODUCTION TO ADT

Definition: An ADT is a data structure defined by its behavior (operations) rather than its implementation. It focuses on what operations are available without revealing how they work.

Key Features

- Encapsulation: Hides internal information from users.
- Deterministic Operations: Provides operations like insert, delete, get.
- Behavior over implementation: Allows multiple implementations for the same ADT.

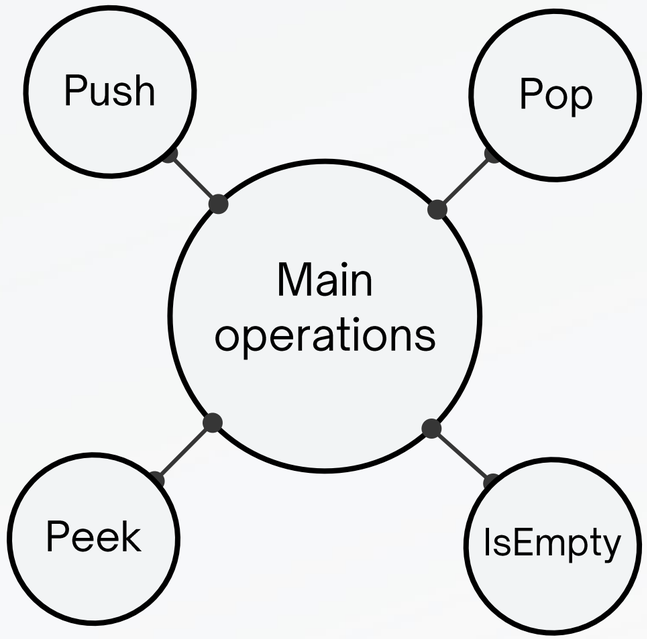
Benefits

- Modularity: Separates interface and implementation for easier management.
- Reusability: ADTs can be reused in different applications.
- Ease of maintenance: Changing the implementation does not affect the code that uses the ADT.

STACK VS QUEUE

1 STACK

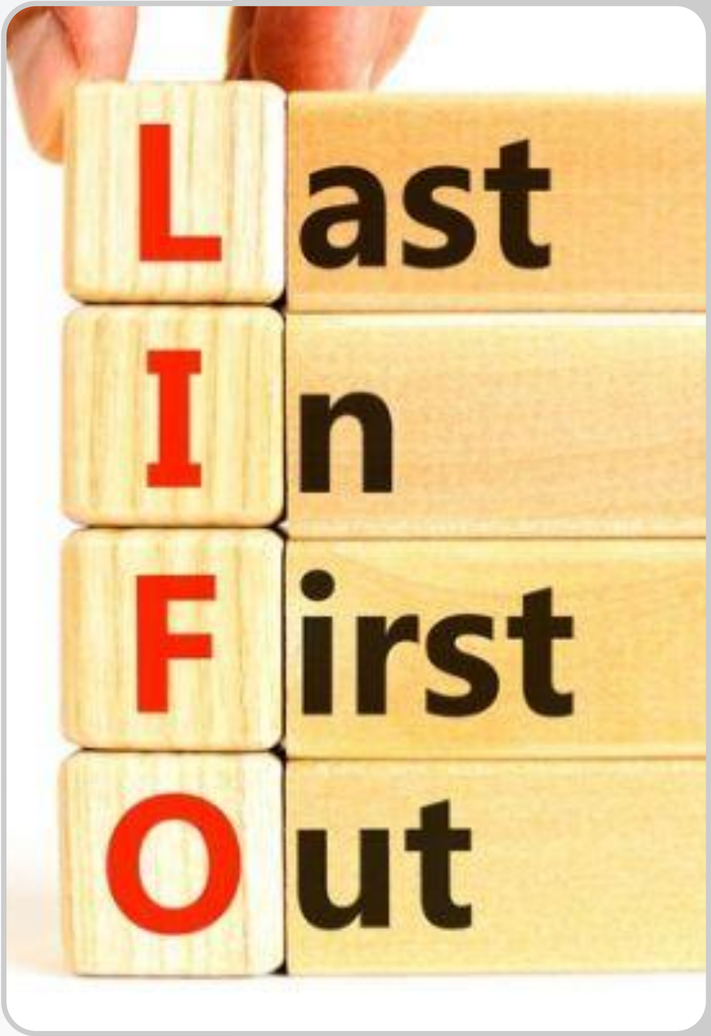
Definition: A stack is a data structure that allows adding and removing elements at only one end, called the top. It is a part of temporary memory in a computer and plays an important role in making function calls and managing local variables.



2 LIFO STACK

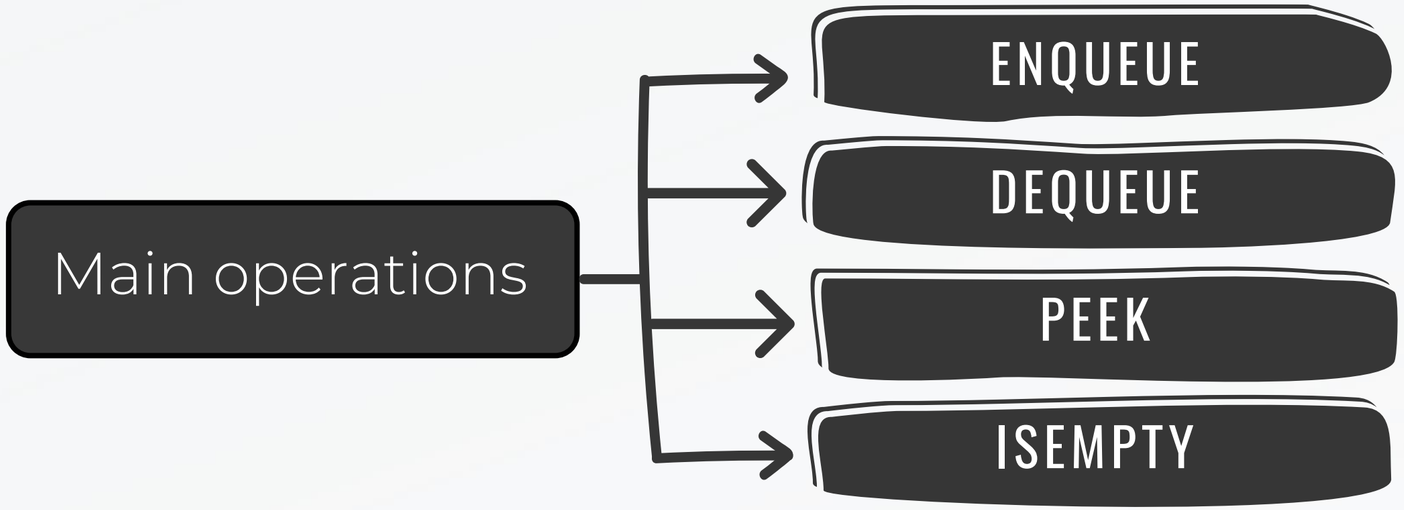
A LIFO (Last In, First Out) stack is a data structure that stores and manages elements according to the principle that the last element added is the first element removed.

LIFO stacks are often used in situations where the most recent item needs to be processed first, such as in undo functionality in applications and in managing function calls in programming (call stacks).



3 QUEUE

Definition: A queue is a data structure that allows elements to be added at one end and removed from the other end, operating on the FIFO principle. This means that the first element added will be the first element removed, which is useful in applications that need to manage the order of processing.



4 FIFO Queues

A FIFO (First In, First Out) queue is a data structure that stores and manages elements according to the principle that the first element added is the first element removed.

FIFO queues are often used in situations where the order of processing is important, such as in network programming and task management.

5 DIFFERENCE BETWEEN STACK AND QUEUE

Stack	Queue
Definition: LIFO (Last In, First Out) data structure.	Definition: FIFO (First In, First Out) data structure.
How it works: The last element added is the first element taken out.	How it works: The first element added is the first element taken out.
Main operations: • push(): add element to the stack. • pop(): take element out of the stack.	Main operations: • enqueue(): add element to queue. • dequeue(): take element out of queue.
Applications: History management, undo command in software, expression analysis.	Applications: Task management, data processing in order, queue creation.
Examples: Call stack, command in browser.	Examples: Printing queue, customer queue.

6 HOW MANY WAYS ARE THERE TO IMPLEMENT STACK AND QUEUE?

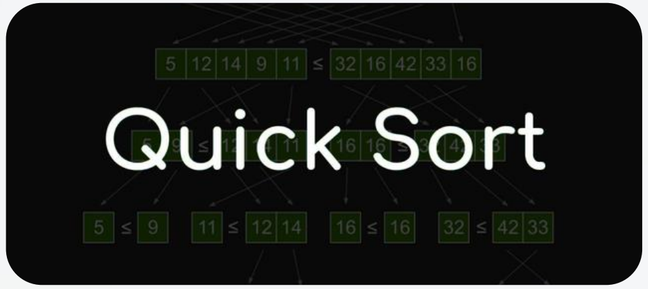
Stacks and Queues can be implemented in various ways based on requirements:

- **Array**: Fixed or dynamic array structures are common implementations.
- **Linked List**: Nodes linked together, ideal for flexible-sized structures.
- **Doubly Linked List**: Used for queues needing operations from both ends.
- **Built-in Libraries**: Many languages offer efficient, pre-built stack and queue libraries.

II

TWO SORTING ALGORITHMS

1 INTRODUCTION TO TWO SORTING ALGORITHMS

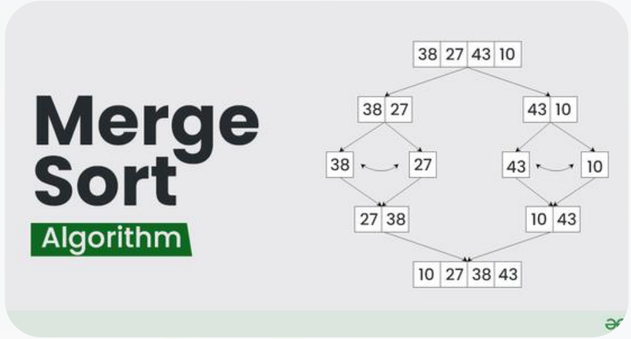


Quick Sort:

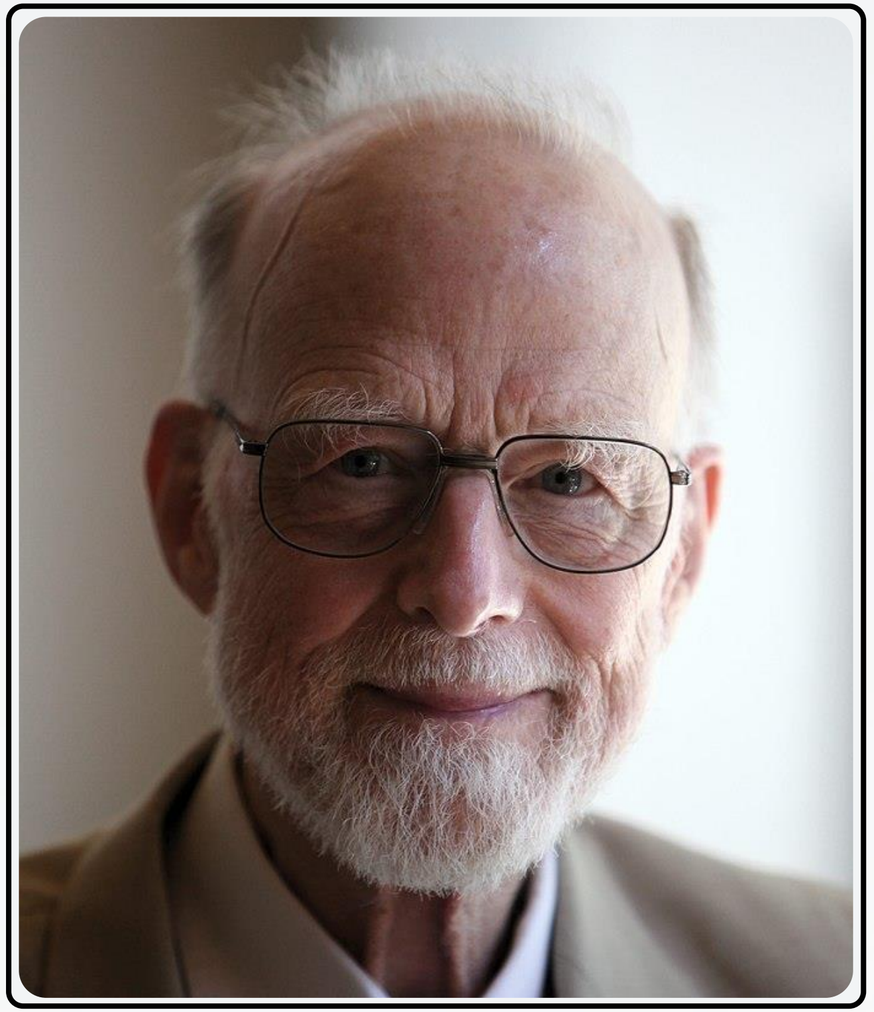
- Efficient sorting algorithm using divide and conquer.
- Average time O(n log n), but can be slower in the worst case.

Merge Sort:

- Divide the array into smaller pieces, sort, and merge them.
- Guaranteed time O(n log n) in all cases, but requires more memory.



QUICK SORT'S CREATOR

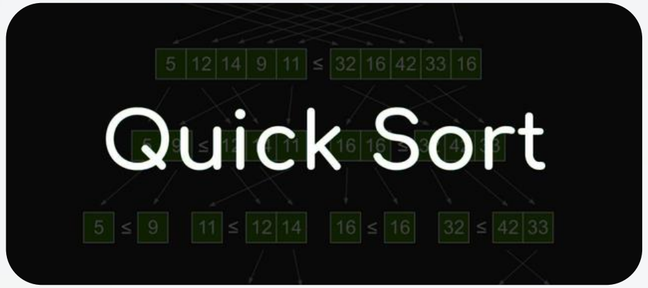


Tony Hoare

MERGE SORT'S CREATOR

John Von Neumann

2 TIME COMPLEXITY ANALYSIS

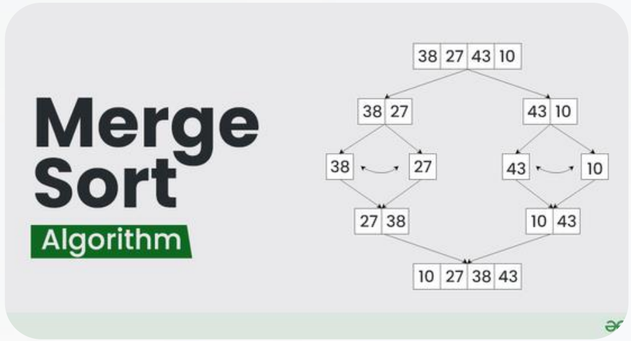


Quick Sort:

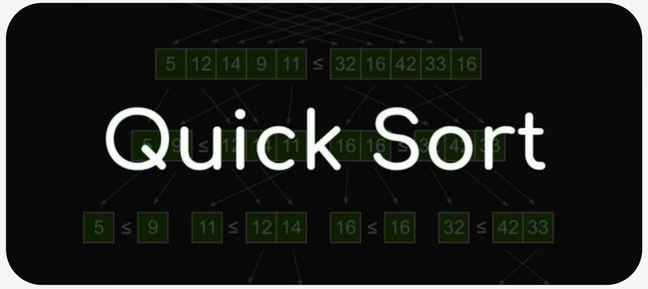
- Best and average time: O(n log n).
- Worst case: O(n²) if pivot is not chosen properly.

Merge Sort:

- Complexity: Always O(n log n) for all cases.
- Consistency: Ensures stable performance in all situations.



3 SPACE COMPLEXITY ANALYSIS

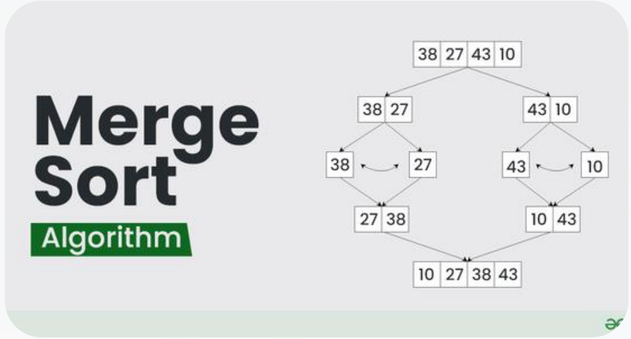


Quick Sort:

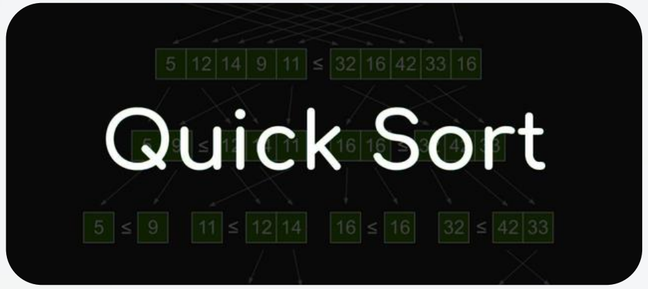
- Best and average time: O(log n) due to recursion.
- Worst case: Can be up to O(n).

Merge Sort:

- Complexity: O(n) due to additional space required to store sub-arrays during merging.
- Disadvantage: Can be a disadvantage in case of large data.



4 | ALGORITHM STABILITY

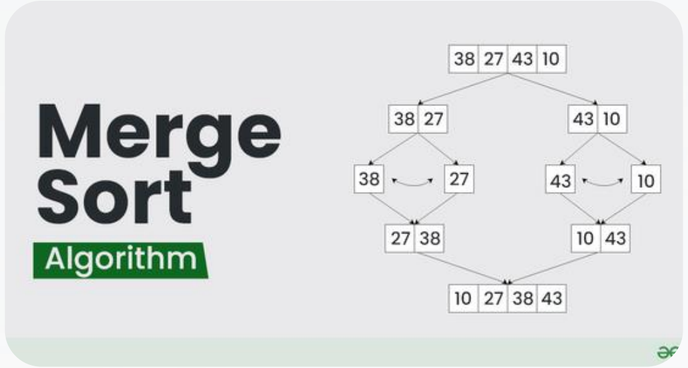


Quick Sort:

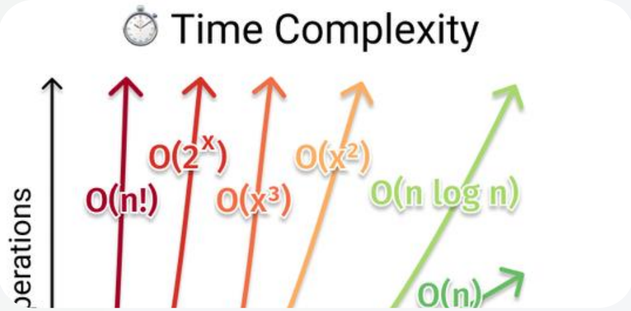
- Stability: Unstable; the order of identical elements may change during sorting.

Merge Sort:

- Stability: A stable algorithm; preserves the order of elements with equal values.
- Applications: Important in many applications where the original order needs to be maintained.



5 COMPARING THE PERFORMANCE OF TWO ALGORITHMS

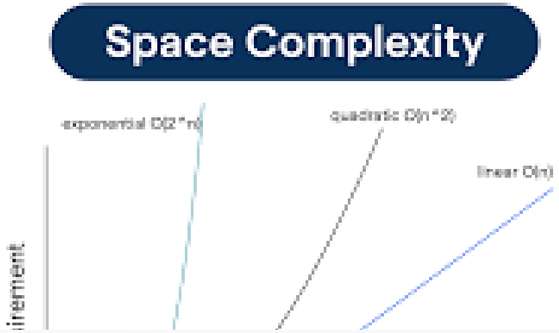


Time complexity:

- Quick Sort: Average performance O(n log n); can go up to O(n²) in the worst case.
- Merge Sort: Always maintains O(n log n) complexity in all situations.

Space complexity:

- Quick Sort: O(log n) in the best case; can go up to O(n) in the worst case.
- Merge Sort: Requires O(n) to store sub-arrays during merge.



6 | CONCRETE EXAMPLE

Example:
Array to be sorted: [38, 27, 43, 3, 9, 82, 10].

Quick Sort:

- Choose pivot: For example, 38.
- Split the array into elements smaller than (3, 9, 27) and larger than (43, 82).
- Call each element recursively until the array is sorted.

Merge Sort:

- Split the array into two halves: [38, 27, 43] and [3, 9, 82, 10].
- Continue dividing until each halve has only one element left.
- Merge the halves in sorted order.

III

TWO NETWORK SHORTEST PATH ALGORITHMS

1 INTRODUCTION TO SHORTEST PATH ALGORITHMS IN NETWORKS

Shortest path algorithms are an important aspect of graph theory and its applications in networks. The main goal of these algorithms is to find the shortest path from a source node to all other nodes in a weighted graph

Dijkstra's Algorithm

Finds the shortest path from a source node to all other nodes in the graph.

Prim–Jarnik Algorithm

Focuses on finding the minimum spanning tree for the graph.

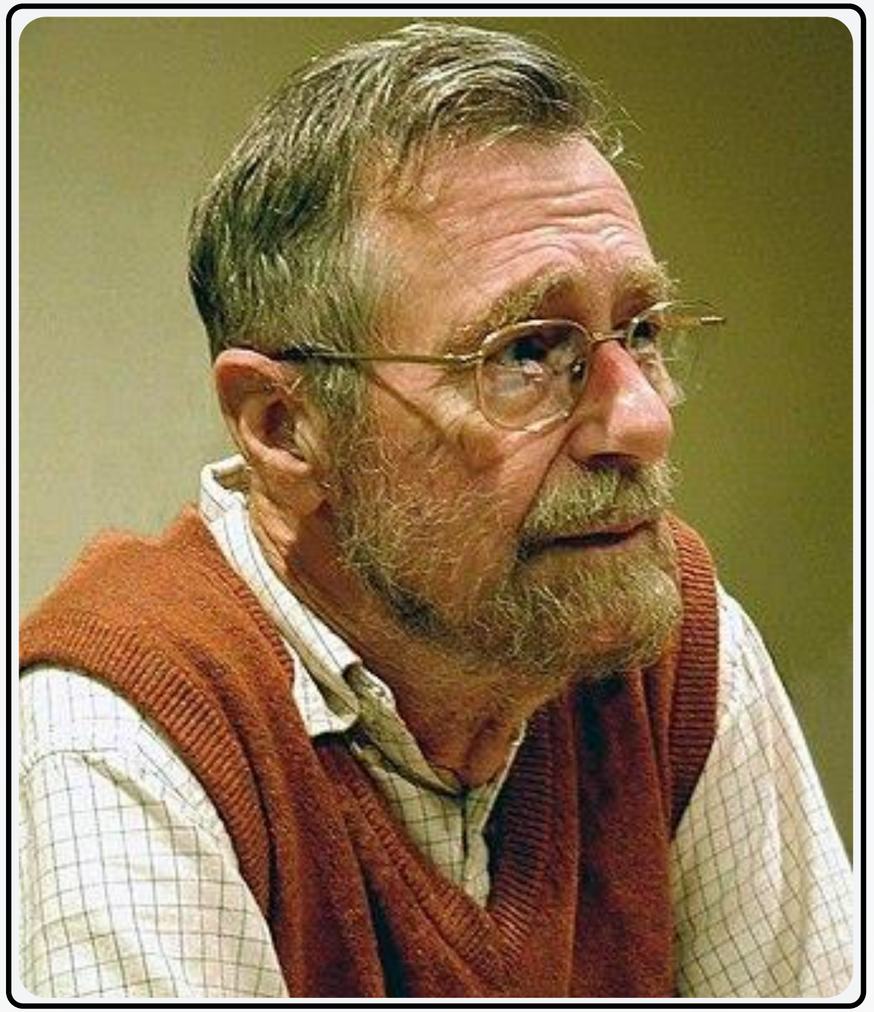
2 DIJKSTRA'S ALGORITHM

Dijkstra's Algorithm is one of the famous algorithms for finding the shortest path in a non-negative weighted graph. The algorithm starts from a source node, explores its neighbors one by one, and updates the shortest path lengths to those nodes.

Specifically, the algorithm performs as follows:

1. Initialize the path length from the source to all nodes to infinity, except for the source node, which is set to 0.
2. Use a priority queue to select the node with the shortest path length that has not been explored yet.
3. Update the path lengths for all neighbors of the selected node.
4. Repeat until all nodes have been explored.

2 DIJKSTRA'S ALGORITHM



Edsger W.
Dijkstra

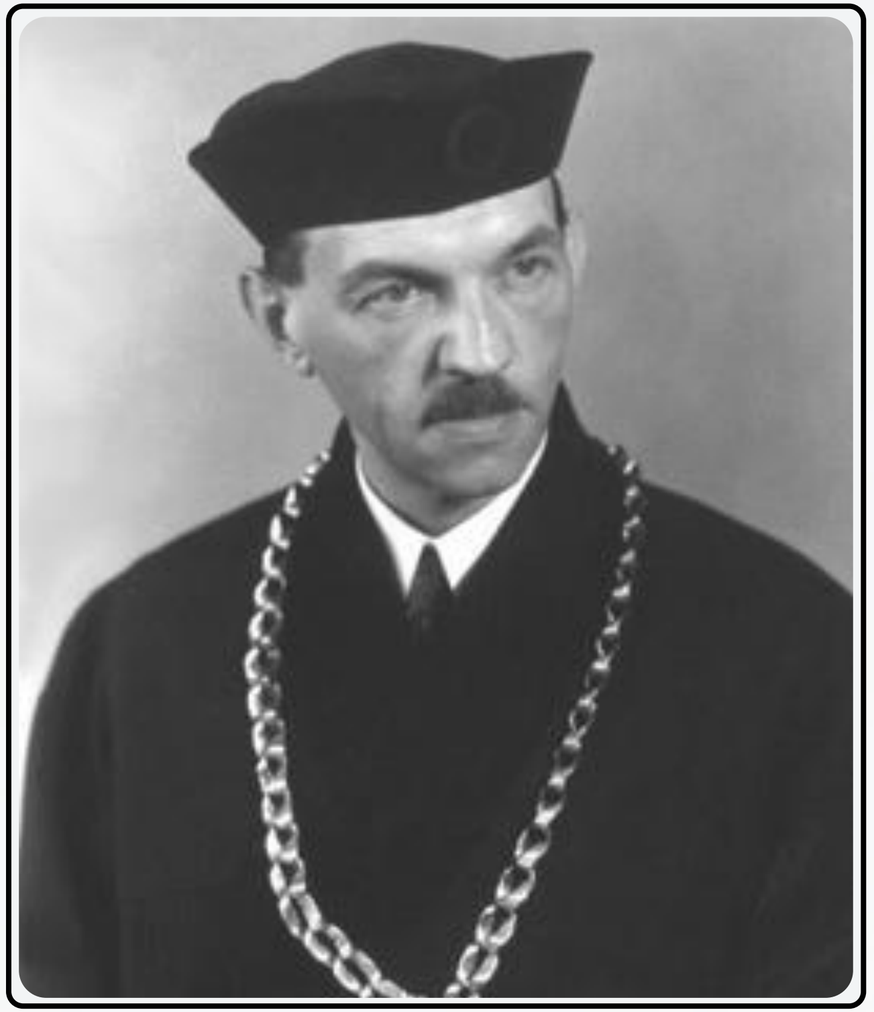
3 PRIM-JARNIK ALGORITHM

The Prim-Jarnik algorithm is an algorithm used to find the minimum spanning tree for a weighted graph. The algorithm works by starting at a node and adding edges with the least weight to the spanning tree, until all the nodes in the graph are connected.

The procedure is as follows:

1. Pick any starting node and mark it.
2. Find the least weighted edge connecting the marked nodes to the unmarked nodes.
3. Add that edge to the spanning tree and mark the new node.
4. Repeat until all the nodes are marked.

3 PRIM-JARNIK ALGORITHM



Vojtěch Jarník

4 PERFORMANCE ANALYSIS

01

TIME COMPLEXITY

Dijkstra is $O(V^2)$ for adjacency matrix, $O((V + E) \log V)$ for priority queue.

Prim-Jarnik is $O(E \log V)$ for priority queue.

02

USE CASES

Dijkstra is good for finding shortest path from a source node.

Prim-Jarnik is used to build minimum spanning tree in network.

03

ACCURACY

Both algorithms are accurate; **Dijkstra** is used for graphs with no negative weights, while **Prim-Jarnik** is used for all weighted graphs.

THANK YOU!