**DATA STRUCTURES**

**Linked List**

- Definition: A linear collection of nodes where each node contains data and a reference to the next node.

- Operations:

- Insert at Beginning: O(1)

- Insert at End: O(n)

 - Delete: O(n)

- Search: O(n)

- Importance:

- Dynamic size

- Ease of insertion/deletion

- Usage:

- Implementing other data structures (stacks, queues)

- Memory management in operating systems

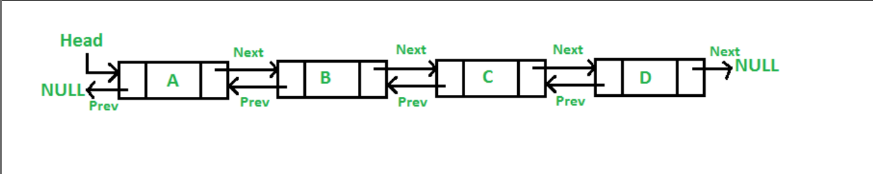
**Doubly Linked List**

- Definition: A linked list where each node contains a reference to both the next and the previous node.

- Operations:

- Insert at Beginning: O(1)

- Insert at End: O(1)

 - Delete: O(1) (if node is given)

- Search: O(n)

- Importance:

- Bidirectional traversal

- Easier deletion

- Usage:

- Navigation systems (e.g., browsers' forward and back functionality)

- Undo functionality in software

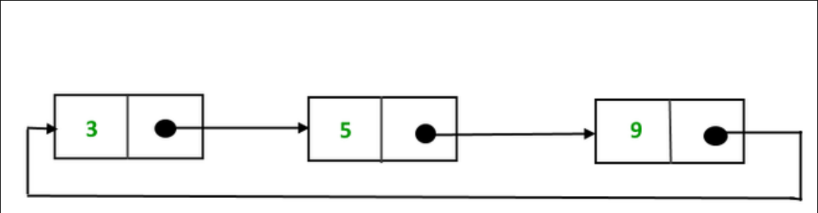
**Circular Linked List**

- Definition: A linked list where the last node returns to the first node.

- Operations:

- Insert at Beginning: O(1)

- Insert at End: O(1)

 - Delete: O(n)

- Search: O(n)

- Importance:

- Cyclic nature

- Continuous looping

- Usage:

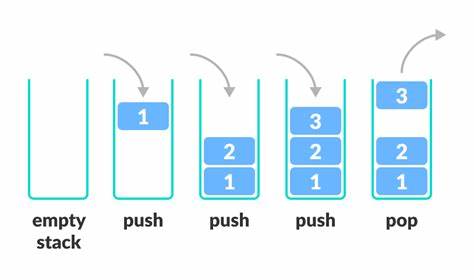
- Round-robin scheduling

- Buffer management (e.g., streaming data)

**Stack**

- Definition: A linear data structure that follows LIFO (Last In First Out).

- Operations:

 - Push: O(1)

- Pop: O(1)

- Peek: O(1)

- IsEmpty: O(1)

- Importance:

- Supports recursive algorithms

- Easy to implement

- Usage:

- Expression evaluation

- Backtracking algorithms (e.g., maze solving)

- Function call management

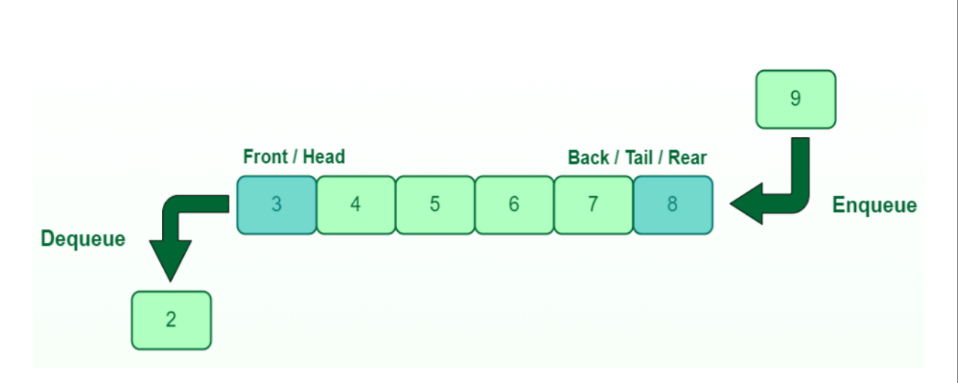
**Queue**

- Definition: A linear data structure that follows FIFO (First In First Out).

- Operations:

- Enqueue: O(1)

- Dequeue: O(1)

 - Peek: O(1)

- IsEmpty: O(1)

- Importance:

- Ordered processing

- Fair scheduling

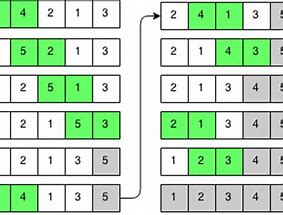
- Usage:

- Task scheduling

- Breadth-First Search (BFS)

- Print spooling

**Sorting Algorithms**



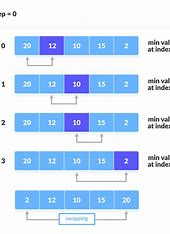
**Bubble Sort**

- Time Complexity: O(n^2)

- Stable: Yes

- Importance: Simple, easy to implement

- Usage: Small datasets, educational purposes



**Selection Sort**

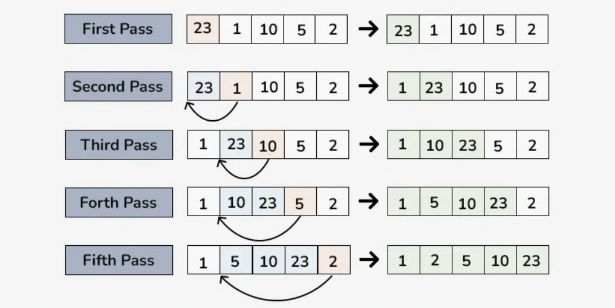
- Time Complexity: O(n^2)

- Stable: No

- Importance: Simple, less swapping

- Usage: Small datasets, where memory writing is costly

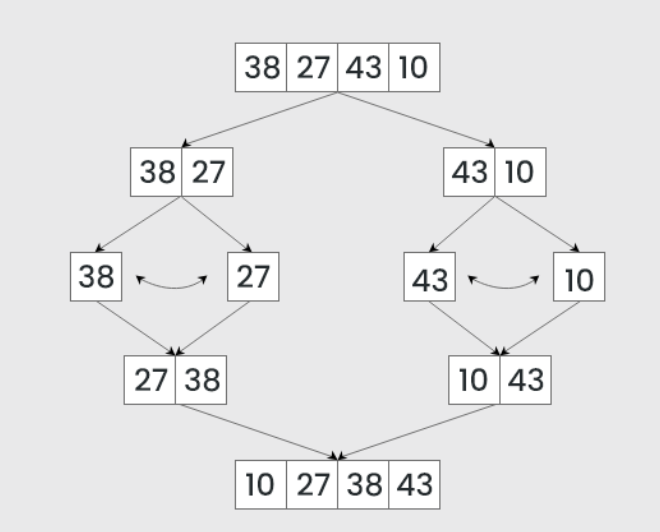
**Insertion Sort**

****- Time Complexity: O(n^2)

- Stable: Yes

- Importance: Efficient for small datasets, adaptive

- Usage: Partially sorted arrays, small datasets

****

**Merge Sort**

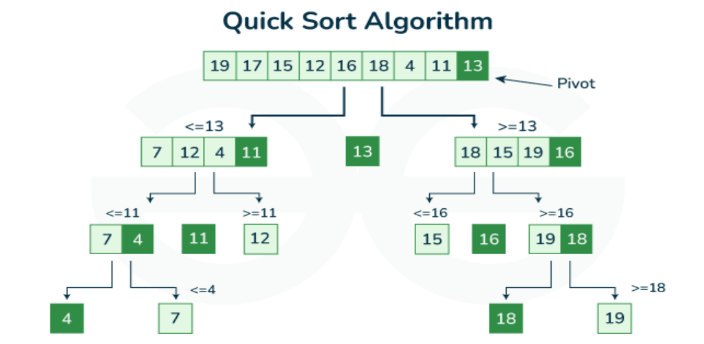
- Time Complexity: O(n log n)

- Stable: Yes

- Importance: Divide and conquer, stable

- Usage: Large datasets, linked lists

**Quick Sort**

- Time Complexity: O(n log n) average, O(n^2) worst-case

- Stable: No

- Importance: Divide and conquer, efficient

- Usage: Large datasets, general-purpose sorting

**Heap Sort**

- Time Complexity: O(n log n)

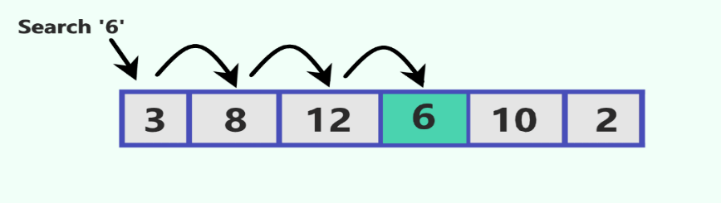
- Stable: No

- Importance: In-place, efficient

- Usage: Priority queues, large datasets

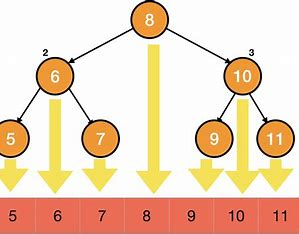
**Search Algorithms**

**Linear Search**

- Time Complexity: O(n)

- Importance: Simple, no need for sorted data

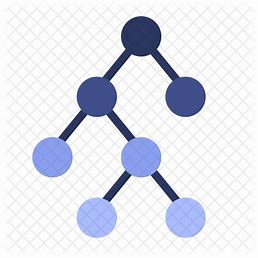
- Usage: Small datasets, unsorted collections

**Binary Search**

- Time Complexity: O(log n)

- Importance: Fast, requires sorted data

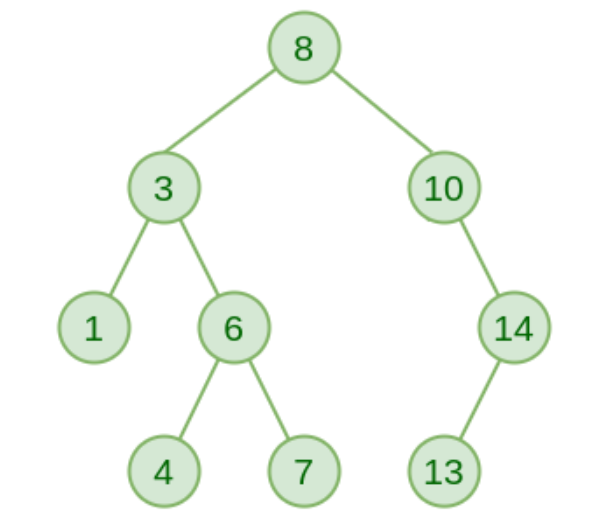
- Usage: Large, sorted datasets, search operations

**Trees**

**Binary Tree**

- Definition: Each node has at most two children.

- Usage: Hierarchical data, expression parsing

**Binary Search Tree (BST)**

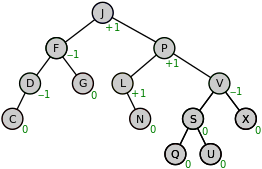
- Definition: Left child < parent < right child.

- Operations: Search (O(log n) average), Insert (O(log n) average),

Delete (O(log n) average)

- Importance: Efficient search, insertion, and deletion

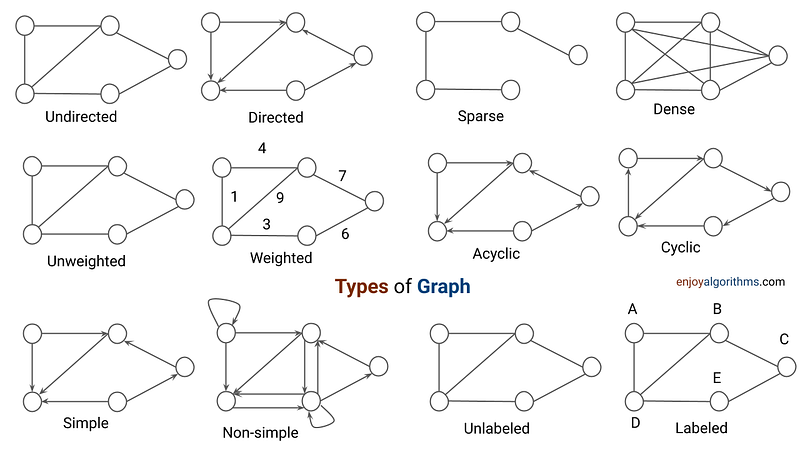
- Usage: Databases, file systems

**Balanced Trees** (e.g., AVL, Red-Black)

- Importance: Maintain height balance to ensure O(log n) operations

- Usage: High-performance applications, databases

**Graphs**

Definitions:

- Vertex (Node): Fundamental unit.

- Edge: Connection between vertices.

Representations:

- Adjacency Matrix: O(V^2) space header

- Adjacency List: O(V + E) space

**Traversal:**

- **Depth-First Search (DFS)**: O(V + E)

- Importance: Pathfinding, topological sorting

- Usage: Solving puzzles, analyzing networks

- **Breadth-First Search (BFS)**: O(V + E)

- Importance: Shortest path in unweighted graphs

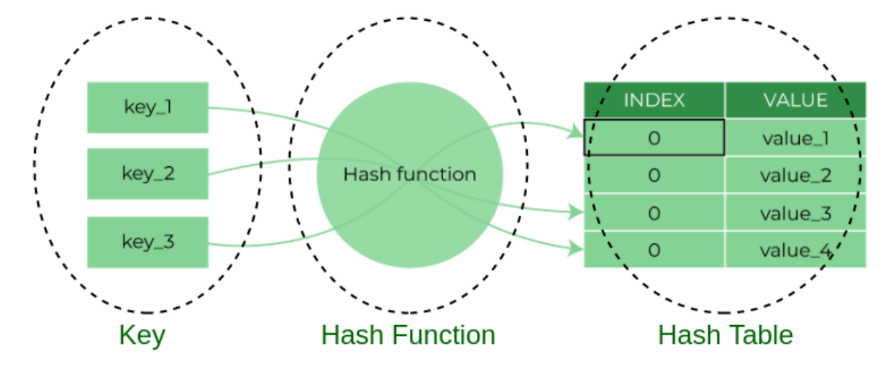
- Usage: Social networking sites, shortest path algorithms

**Hash Table**

- Definition: Stores key-value pairs for efficient lookups.

- Operations: Insert (O(1) average), Search (O(1) average), Delete (O(1) average)

- Importance: Efficient lookups, insertions, and deletions

- Usage: Caching, databases, associative arrays