Time Complexity and Space Complexity in Data Structures and Algorithms (DSA)

1. Introduction

In DSA, understanding how efficiently an algorithm runs and how much memory it uses is crucial. These efficiencies are measured by:

- **Time Complexity**: How the execution time of an algorithm changes with input size.
- **Space Complexity**: How the memory consumption of an algorithm changes with input size.

2. Time Complexity

- Describes the amount of time an algorithm takes to complete relative to input size `n`.
- Usually expressed using Big O notation, which provides an upper bound on the running time.

Common Time Complexities:

Complexity	Description	Example
O(1)	Constant time	Accessing an array element
O(log n)	Logarithmic time	Binary Search
O(n)	Linear time	Linear Search
O(n log n)	Linearithmic time	Efficient sorting (Merge Sort)
$O(n^2)$	Quadratic time	Bubble Sort
O(2^n)	Exponential time	Recursive Fibonacci

3. Space Complexity

- Describes the amount of memory an algorithm uses relative to input size `n`.
- Also expressed in Big O notation.

Space Considerations:

- Variables, data structures, stack space (in recursion).
- Example:
  - Iterative algorithms usually use O(1) extra space.
  - Recursive algorithms may use O(n) space due to call stack.

4. Examples

- **Linear Search**:
  - Time Complexity: O(n) — must check each element.
  - Space Complexity: O(1) — no extra space apart from input.
- **Merge Sort**:
  - Time Complexity: O(n log n) — divide and conquer.
  - Space Complexity: O(n) — temporary arrays for merging.