1. Introduction to Algorithms

- An algorithm is a step-by-step method for solving problems or completing tasks. They are at the heart of software development and computational problem-solving.
- Time Complexity: Refers to how long an algorithm takes to execute as the input size changes. Examples of common complexities are:
    - $O(1)$: Constant execution time
    - $O(n)$: Execution time grows linearly with input size
    - $O(n^2)$: Execution time grows quadratically
- Space Complexity: This measures the memory usage of an algorithm relative to the input size.

2. Common Data Structures

- Data structures are frameworks used to store and manage data, optimizing access and modifications.
- Arrays: Ordered collections of elements stored in contiguous memory. Arrays offer constant-time element access but require resizing to accommodate larger data.
- Linked Lists: Composed of nodes, where each node holds data and a link to the next node. Linked lists allow dynamic sizing and are ideal for scenarios where data is frequently added or removed.

3. Popular Sorting Algorithms

- Sorting algorithms reorder data in a specified way, typically in ascending or descending order. Common sorting algorithms include:
    - Bubble Sort: This algorithm compares and swaps adjacent elements repeatedly until the entire list is sorted. Time complexity: $O(n^2)$.
    - Merge Sort: It uses a divide-and-conquer approach to split data into smaller arrays, sort them, and then merge them back together. Time complexity: $O(n \log n)$.

4. Efficient Search Techniques

- Search algorithms are used to locate elements in data structures. Two widely used search algorithms are:
    - Linear Search: Sequentially examines each element in a list until the target value is found. Time complexity: $O(n)$.
    - Binary Search: This algorithm works on sorted arrays, repeatedly dividing the search range in half until the target value is found. Time complexity: $O(\log n)$.