MsDS - Data Structures and Algorithms

Module No: 3

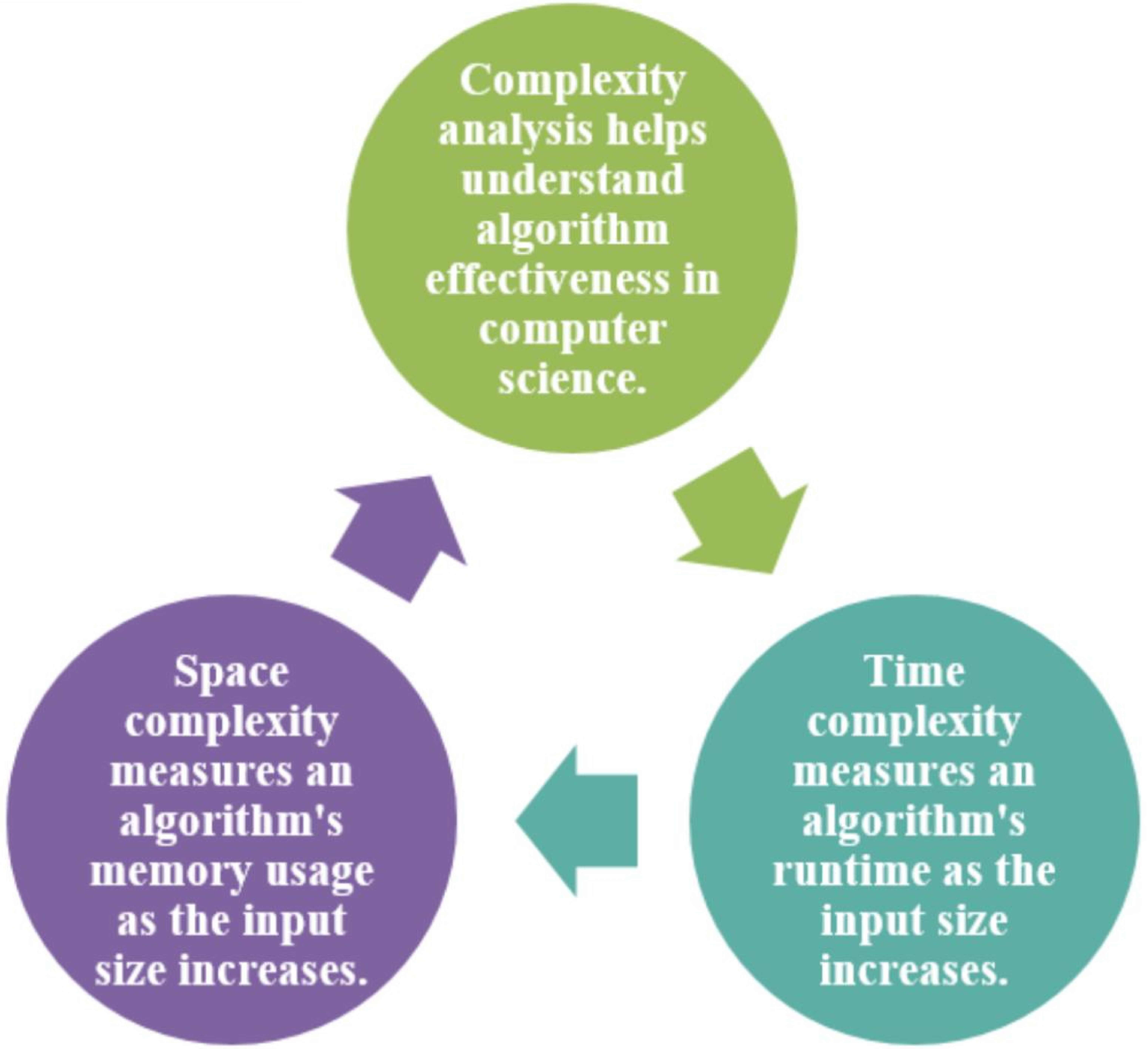
Revision from last Lecture

1. Module 2 compares static allocation (arrays) with dynamic allocation (linked lists) in terms of memory allocation and access efficiency.
2. Access-restricted lists are explored, emphasising the importance of restricting data access for operational efficiency and optimisation.
3. Stacks and queues are fundamental data structures that follow LIFO and FIFO principles, respectively, with specific use cases in function calls, expression evaluation, work scheduling, and breadth-first search algorithms.

Expected Learning Outcomes

1. To assess the effectiveness of algorithms, be familiar with the foundations of time complexity and space complexity.
2. Develop the Big-O Notation application skills for evaluating the scalability and performance of algorithms.
3. Comprehend the idea of recurrence relations and create simple recursive algorithmic solutions.
4. Effectively design and optimise data structure operations using the divide-and-conquer method.

Time Complexity and Space Complexity


Complexity analysis helps understand algorithm effectiveness in computer science.
Time complexity measures an algorithm's runtime as the input size increases.
Space complexity measures an algorithm's memory usage as the input size increases.

Review and use of Big-O Notation

- Big-O notation provides a high-level understanding of an algorithm's efficiency and scalability, disregarding low-level details and hardware specifics.
- It allows developers to compare the effectiveness of different algorithms and predict their performance based on input size.
- However, it is important to consider that Big-O notation only provides an upper limit and may not accurately reflect the actual performance of an algorithm in all scenarios.

Recurrence Relations and Simple Solutions

Recurrence relations are important in computer science for analysing the computational complexity of recursive algorithms.

Recurrence relations can be used to understand the sequential decomposition of a problem and the running time of an algorithm.

Recurrence relations can be solved using techniques such as iteration, substitution, the Master theorem, and generating functions, but not all recurrences have closed-form solutions.

Use of Divide-and-Conquer in Designing Operation on Data Structure

- The divide-and-conquer approach decomposes a problem into smaller sub-problems, which are then addressed individually until they can be directly resolved.
- The three stages of the divide-and-conquer methodology are division, conquest, and consolidation.
- Examples of divide-and-conquer algorithms include binary search, quick sort and merge sort for sorting, tree and graph traversals, and matrix multiplication using the Strassen method.

Important Terminologies

- **Time Complexity:** The runtime of an algorithm is evaluated in terms of its scalability with respect to the size of the input.
- **Space Complexity:** The term "space complexity" refers to the amount of memory that an algorithm requires in relation to the size of its input.
- **Big-O Notation:** The use of standardised notation is employed to articulate the upper bound growth rate of an algorithm's time or space complexity in the worst-case scenario.

Important Terminologies

- **Recurrence Relations:** Equations that establish a correlation between distinct components of a sequence, often used to comprehend the intricacy of recursive functions.
- **Divide-and-Conquer:** The algorithmic methodology used involves the decomposition of issues into smaller sub-problems, the resolution of each sub-problem, and the subsequent integration of their respective answers.
- **Recursive Algorithms:** Recursive techniques are algorithms that use fewer inputs to solve bigger issues by repeatedly using the same logical process.

Summary

- ❖ Understanding time complexity and space complexity helps predict algorithm behavior and optimise performance and resource usage.
- ❖ Big-O notation provides a concise measure for comparing and evaluating the efficiency of different algorithms.
- ❖ Recursive algorithms use recurrence relations to analyse computational efficiency and problem resolution. The divide-and-conquer technique simplifies complex problems and is essential in efficient data structure operations.

THANK YOU