Data Structures

Dilip Kumar Maripuri
Computer Applications

Data Structures

Session : Performance analysis: Time complexity, Asymptotic Notation

Dilip Kumar Maripuri
Computer Applications

- **Definition and Significance**
  - The rate of growth in the context of algorithms refers to how the resource consumption (time, memory, etc.) of an algorithm increases as the size of the input data increases.
    - It provides a high-level understanding of the algorithm's efficiency.
    - It helps in predicting the scalability of an algorithm.
    - It allows for comparing different algorithms in a standardized way, independent of hardware or implementation details.

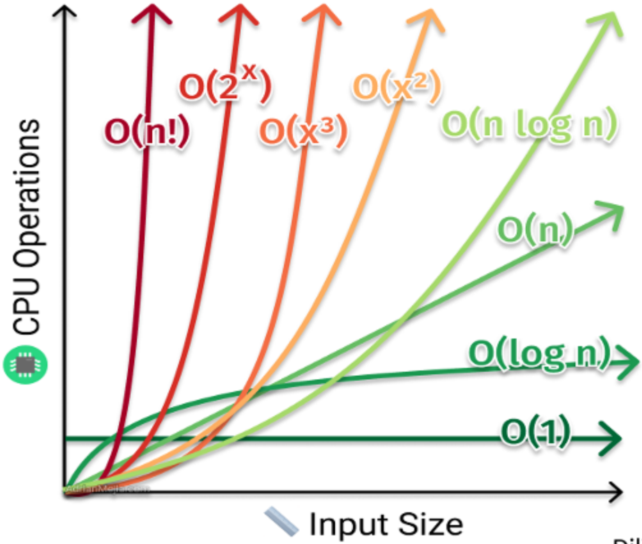
- **Simplifying Assumptions in Growth Rate Comparison**
  - When comparing the growth rates of algorithms, certain simplifying assumptions are made to focus on the most significant factors:
  - **Dominant Term**
    - Only the highest order term is considered because it has the most significant impact on growth rate as the input size becomes large.
    - For instance, in $O(n^2 + n)$, the n term is negligible compared to n^2 for large n.

- **Simplifying Assumptions in Growth Rate Comparison**
- **Constant Factors**
  - Constants are ignored in Big O notation since they do not change the growth rate.
  - For example, 2n and 100n are both O(n).
- **Lower Order Terms**
  - Lower-order terms are discarded in Big O notation for large input sizes because their impact on the growth rate is minimal compared to higher-order terms.

- **Linear Growth (O(n))**
  - Example: A check for duplicates in a list by scanning each element exactly once.
  - Significance: The time taken grows directly in proportion to the number of elements.
- **Quadratic Growth ($O(n^2)$)**
  - Example: A bubble sort algorithm where each element is compared to every other element.
  - Significance: The time taken grows exponentially relative to the number of elements, making it impractical for large datasets.

- **Logarithmic Growth (O(log n))**
  - Example: Binary search in a sorted array.
  - Significance: The time taken increases only slightly with the size of the input, representing an efficient search for large datasets.
- **Exponential Growth ($O(2^n)$)**
  - Example: The number of possible different sequences in which to visit n nodes in a traveling salesman problem without repeating any.
  - Significance: The time taken grows very quickly with the number of nodes, becoming unfeasible for even modestly sized n.

Algorithms - Rate of Growth



- To understand how to analyze the performance of an algorithm, particularly in terms of time complexity, it is essential to grasp the concepts of best, worst, and average case scenarios.
- These are the fundamental aspects of asymptotic analysis, which helps in understanding the behavior of algorithms in terms of time taken for different inputs.

Types of Analysis

- What is it?
  - Asymptotic analysis is a method of describing limiting behavior and provides an upper or lower limit for the performance of an algorithm.
- Why is it used?
  - It helps in simplifying the expression of an algorithm's time complexity by focusing on the most significant factors, especially for large input sizes.
- **Types**
  - Worst Case Analysis
  - Best Case Analysis
  - Average Case Analysis

- **Worst Case Analysis**
  - **Definition**
    - The worst case scenario defines the input for which the algorithm takes the longest time to complete.
  - **Importance**
    - It provides a guarantee on the algorithm's performance, ensuring that it will not take longer than this time under any circumstances.
  - **How to Determine**
    - Analyze the algorithm's behavior with the most challenging possible input data set.
    - This could mean the largest size, most complex structure, or a specific arrangement that makes the algorithm work hardest.

- **Best Case Analysis**
  - **Definition**
    - The best case scenario is where the algorithm takes the least time to complete.
  - **Importance**
    - While not as critical as the worst case, it gives an idea of how efficient the algorithm can be under ideal conditions.
  - **How to Determine**
    - Analyze the algorithm with the simplest or smallest possible input data set that requires the least processing.

- Average Case Analysis
  - Definition
    - This provides an average running time of the algorithm across all possible inputs.
  - Importance
    - It offers a more realistic measure of the algorithm's performance in practical scenarios.
  - How to Determine
    - It requires statistical knowledge, assuming that all inputs are equally probable.
    - We might need to calculate the average of all possible inputs' running times.

Practical Steps for Analysis

- **Identify the Operation to Count** : Determine the basic operation of the algorithm (like comparisons in a sorting algorithm).
- **Establish the Best, Worst, and Average Cases** : Based on the algorithm's logic, identify the conditions that lead to these cases.
- **Use Asymptotic Notations** : Express the time complexity using Big O (worst case), Big Omega (best case), and Big Theta (average case) notations.
- **Calculate Time Complexity** : Derive the expressions for time complexity based on the input size (n) for different cases.

Considerations

- **Input Size Matters :** The performance often varies significantly with the size of the input data.
- **Algorithm's Structure :** Recursive, iterative, and other structural elements of the algorithm influence its performance.
- **Data Structures Used :** The choice of data structures (arrays, lists, trees, etc.) also impacts performance.

RAM Model

- The RAM model is a theoretical model of computation used to simplify the analysis of algorithms.
- It assumes that each simple operation (like addition, multiplication, assignment, if-statement, etc.) takes a constant amount of time.
- Memory is accessed randomly (i.e., accessing any memory location takes constant time).

RAM Model - Why

- **Simplification :** It simplifies the complexity analysis by ignoring specific hardware and system details.
- **Focus on Algorithm :** The RAM model allows us to focus on the algorithm's logic rather than its implementation on a specific machine.

RAM Model - Characteristics

- **Uniform Memory Access :** Memory access in the RAM model is uniform, meaning each memory access takes a constant amount of time, regardless of the memory location.
- **Basic Operations :** Fundamental operations like addition, subtraction, assignment, and comparison are performed in a single time step.
- **No Concurrency :** The model assumes a single processor executing instructions sequentially, without parallel processing.
- **Direct Access to Inputs :** The inputs to the algorithm are directly accessible in the memory.

- **Analyzing a Simple Algorithm Using the RAM Model**
  - **Goal**: To find a specific element in an array.
  - **Algorithm Steps**:
    1. Start from the first element of the array.
    2. Compare the current element with the target element.
    3. If they match, return the index.
    4. If not, move to the next element.
    5. Repeat steps 2-4 until the end of the array.
    6. If the element is not found, return -1.

- **Analyzing a Simple Algorithm Using the RAM Model**
  - **Basic Operation**:
    - The primary operation in this algorithm is the comparison between the current element and the target element.

- **Time Complexity Calculation**
  - **Best Case O(1)**
    - This occurs when the element is found at the beginning of the array, requiring only one comparison.
  - **Worst Case O(n)**
    - In this scenario, the element is either not found or found at the very end, necessitating n comparisons, where n is the size of the array.
  - **Average Case O((n+1)/2)**
    - Assuming each element is equally likely to be the target, the average number of comparisons is approximately, which simplifies to O(n) in asymptotic notation.

- **Considerations in the RAM Model**
  - Each comparison and the action of moving to the next array element are considered to take a constant amount of time.
  - The algorithm does not involve complex operations or use sophisticated data structures, making the analysis straightforward under the RAM model.

- **Key Points in RAM Model Analysis**
  - **Constant Time Operations**
    - Assume each basic operation takes constant time.
  - **Ignore Non-Dominant Terms**
    - In Big O notation, we ignore less significant terms for large n.
  - **Consider Worst Case**
    - Often, we focus on the worst case for guarantees on performance.

- **Fibonacci Sum Algorithm**
  - **Goal:** To calculate the sum of the first n Fibonacci numbers.
  - **Algorithm Steps:**
    - Initialize three variables: **a = 0**, **b = 1** (the first two Fibonacci numbers), and **sum = 0**.
    - For each number from 1 to n:
    - Add **a** to **sum**.
    - Calculate the next Fibonacci number: **next = a + b**.
    - Update **a** and **b**: set **a = b** and **b = next**.

- **Basic Operation**
  - The primary operations are the addition of numbers and updating the variables for the next Fibonacci number.
- **Time Complexity Calculation**
  - Each Iteration: In each iteration of the loop, a constant number of operations are performed: two additions and two assignments.
  - Total Iterations: The loop runs n times.

- **Algebraic Expression for Time Complexity**
  - **Additions:** Each iteration performs two additions (**a + b** and **a** added to **sum**), resulting in **2n** additions for **n** iterations.
  - **Assignments:** Each iteration involves three assignments (**next**, **a**, and **b**), resulting in **3n** assignments for **n** iterations.
  - **Total Operations:** The total number of operations is **2n** (additions) + **3n** (assignments) = **5n**.

- **Final Time Complexity**
  - **Time Complexity:** The time complexity, based on the number of basic operations, is **O(5n)**.
  - **Simplification:** In asymptotic analysis, constants are ignored, so the time complexity simplifies to **O(n)**.

Thank You

Dilip Kumar Maripuri
Associate Professor
Department of Computer Applications
dilip.maripuri@pes.edu
8073212026