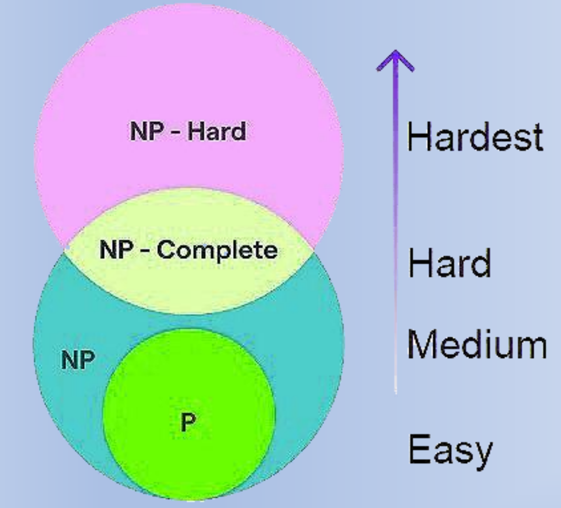
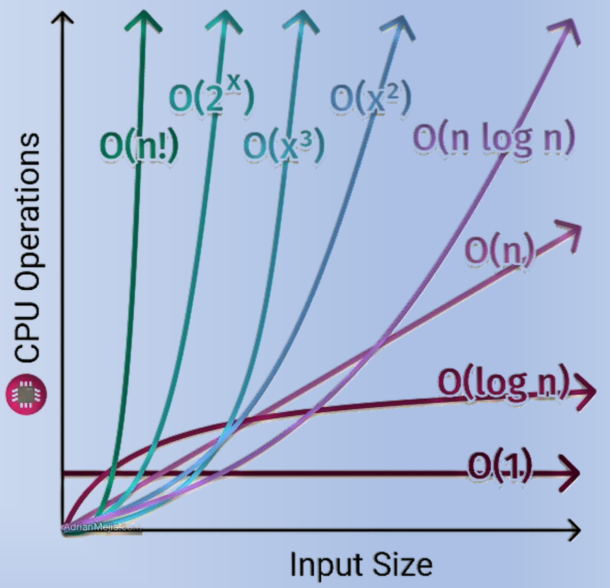
Complexity

S. M. Hossein Mousavi
Lugano, Switzerland
December 2024

Computational Complexity Theory

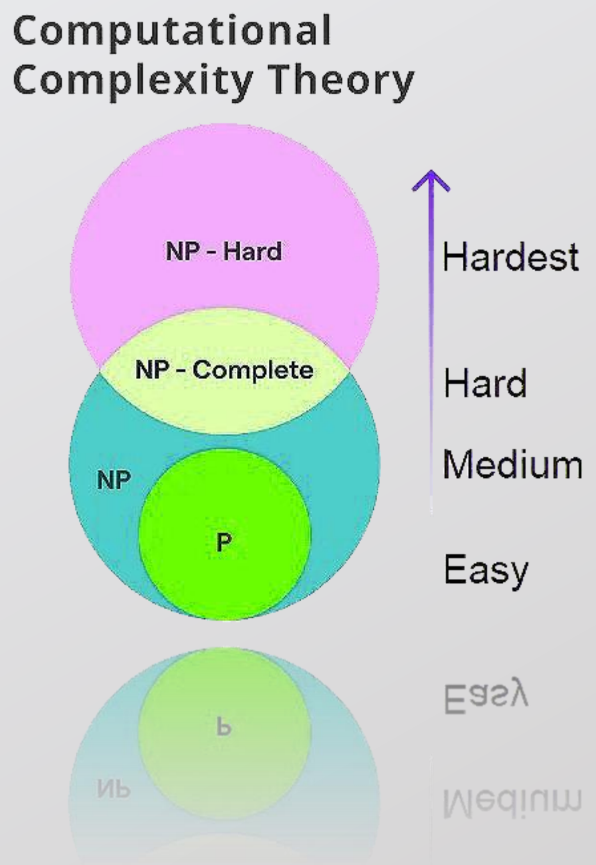


Time Complexity



Outline:

- **Complexity**
  - Definition
  - Importance
  - Applications
  - Big O – Notation Classes



❖ Definition

- The complexity of an algorithm is the **amount of resources required to run** the algorithm.
- Particular focus is given to **computation time** (generally measured by the **number of needed operations**) **and memory storage** requirements.
- The complexity of a problem is the **complexity of the best algorithms** that allow solving the problem.
- The complexity refers to the study of how computational resources (like **time, memory, or energy**) are consumed by **algorithms or systems**.
- Complexity helps us evaluate and compare the efficiency and feasibility of solutions for computational problems.

❖ Importance

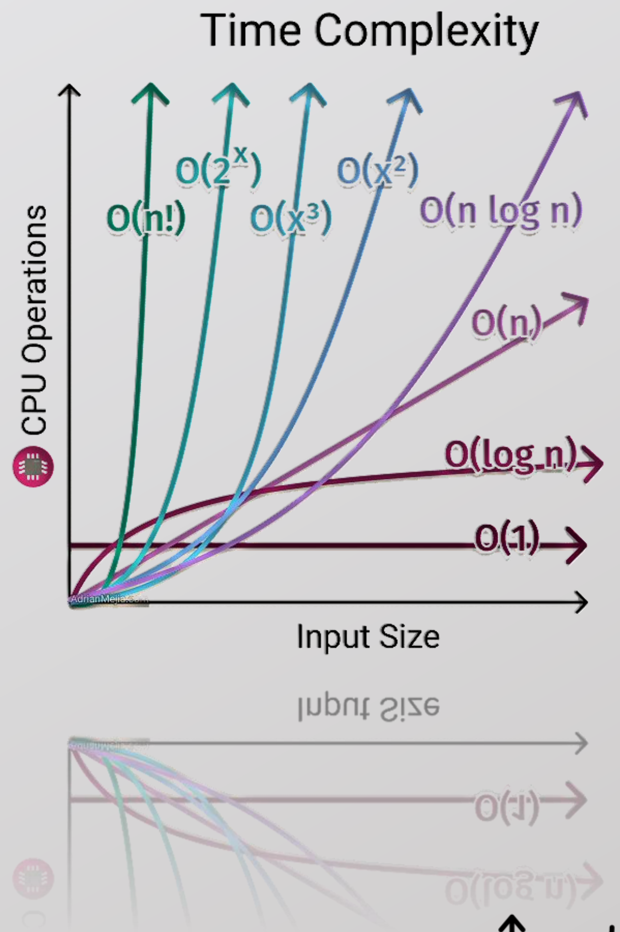
- The calculation of algorithmic complexity is crucial as it helps evaluate the **efficiency and feasibility of solutions to computational problems.**
- By analyzing complexity, we can determine **how an algorithm scales with input size, predict its performance, and identify resource requirements such as time, memory, or energy**.
- This allows for informed decisions when **choosing or designing algorithms, ensuring optimal performance and cost-effectiveness, especially for large-scale or resource-constrained applications**.
- Understanding complexity is **essential for balancing trade-offs and developing robust, scalable systems**.

❖ Applications

- **Algorithm Optimization**: **Identifying inefficiencies in algorithms to improve runtime** and memory usage.
- **Scalability Assessment**: Ensuring that **solutions remain effective as input sizes grow**, critical in fields like **big data, machine learning, and cloud computing.**
- **System Design**: **Choosing the most suitable algorithms for hardware or software systems based on available resources.**
- **Real-Time Applications**: **Ensuring algorithms meet strict time constraints** in systems like robotics, autonomous vehicles, or real-time monitoring.
- **Gaming and Graphics**: **Optimizing algorithms for rendering**, physics simulation, and AI behavior in real-time environments.
- **Network Design**: **Optimizing routing protocols and resource allocation for efficient communication**.
- **Energy Efficiency**: **Designing algorithms with lower energy consumption for battery-operated** or energy-constrained devices.
- **Scientific Simulations**: **Ensuring computational models in physics, biology**, or climate science are efficient and scalable.
- **Artificial Intelligence**: **Optimizing machine learning algorithms for faster training** and inference in AI applications.

❖ Big O – Notation Classes

- It describes how algorithms grow in terms of resource use (like time or memory) as input size increases.
- It goes from the **fastest/less complex** of **O(1)** to the **slowest/most complex** of **O(n!)**.
- **The number of samples, iterations, epochs, data size, and layers affects the complexity.**
  - ➢ **Constant Time** (***O*(1)**) class:
    - ✓ **Definition**: Takes the same time regardless of input size.
    - ✓ **Example**: Accessing an element in an array by index.
    - ✓ **Graph Shape**: Flatline.
  - ➢ **Logarithmic Time (O(log n))**class:
    - ✓ **Definition**: Time grows slowly as the input size increases.
    - ✓ **Example**: Binary search on a sorted list.
    - ✓ **Graph Shape**: Slowly increasing curve.
  - ➢ **Linear Time (O(n))**class:
    - ✓ **Definition**: Time grows directly proportional to the input size.
    - ✓ **Example**: Searching for an element in an unsorted list.
    - ✓ **Graph Shape**: Straight diagonal line.
  - ➢ **Linearithmic Time (O(n log n))**class:
    - ✓ **Definition**: Combination of linear and logarithmic growth.
    - ✓ **Example**: Merge sort or heap sort.
    - ✓ **Graph Shape**: Curves faster than linear but slower than quadratic.



❖ Big O – Notation Classes

- **Quadratic Time** (**$O(n^2)$**) class:
  - ✓ **Definition**: Takes the same time regardless of input size.
  - ✓ **Example**: Accessing an element in an array by index.
  - ✓ **Graph Shape**: Flatline.
- **Quadratic Time** (**$O(n^2)$**) class:
  - ✓ **Definition**: Time grows as the square of input size.
  - ✓ **Example**: Comparing all pairs in a list (e.g., bubble sort).
  - ✓ **Graph Shape**: Parabolic curve.
- **Cubic Time** (**$O(n^3)$**) class:
  - ✓ **Definition**: Time grows as the cube of input size.
  - ✓ **Example**: Matrix multiplication with brute force.
  - ✓ **Graph Shape**: Steep parabolic curve.
- **Exponential Time** (**$O(2^X)$**) class:
  - ✓ **Definition**: Time doubles with each additional input.
  - ✓ **Example**: Solving the traveling salesman problem (brute force).
  - ✓ **Graph Shape**: Exponentially steep curve.
- **Factorial Time** (**$O(n!)$**) class:
  - ✓ **Definition**: Time grows faster than exponential.
  - ✓ **Example**: Generating all permutations of n elements.
  - ✓ **Graph Shape**: Extremely steep curve.