What is Computational Complexity?

Complexity theory addresses the study of the fundamental complexity of computational tasks. The "ultimate goal" is to judge for every well-defined task, how complex it is to solve it, i.e., how many resources and how much time is needed – at most and at least – to perform the given task.

How do different tasks relate to each other in their complexity? Are some always more difficult than others, no matter how we model them? How do time and resources relate to each other? Are there "most difficult" problems? To study these questions, the first step is to have a precise understanding of what it means to perform a task, to solve a problem.

Polynomial time and its justification

Polynomial time is a concept in computer science and computational complexity theory that refers to the efficiency of an algorithm in terms of its running time as a function of the size of its input.

Definition: An algorithm runs in polynomial time if its worst-case running time can be bounded by a polynomial function of the input size.

In other words, if the worst-case running time of an algorithm can be expressed as a polynomial in the size of the input, then it is considered to be a polynomial-time algorithm.

Example: O(n^2), O(n^3), and O(n) are examples of polynomial time; O(2^n) is not.

Mathematically: Mathematically, an algorithm's running time T(n) is said to be polynomial if there exists a polynomial function P(n) such that:

$T(n) <= P(n)$

Where:

T(n) is the worst-case running time of the algorithm for an input of size n.

P(n) is a polynomial function in n (typically expressed as a sum of terms involving powers of n).

Polynomial time is a desirable property for algorithms because it implies that the algorithm's running time grows reasonably with the size of the input. Polynomial-time algorithms are considered efficient and practical for many real-world problems, as their running times remain manageable even for large inputs.

Justification for Polynomial Time

Feasibility: Polynomial time algorithms are practically feasible for a wide range of problem sizes. They can handle input sizes that are relevant to many real-world applications. Algorithms with exponential or factorial time complexity, on the other hand, quickly become impractical as the input size increases.

Computational Resources: In practice, computers and computational resources are finite. Polynomial time algorithms are more likely to be solvable within reasonable time and resource constraints, making them suitable for solving large-scale problems.

P vs. NP: The concept of polynomial time is closely related to the famous P vs. NP problem in theoretical computer science. If a problem can be solved in polynomial time, it belongs to the class P (problems that are efficiently solvable). However, many important problems, known as NP problems, do not have known polynomial-time solutions. If it were proven that P = NP, it would have profound implications for computational complexity theory and cryptography.

Empirical Evidence: Many practical algorithms for various problem domains have been designed with polynomial time complexity, and they have been successfully used to solve real-world problems efficiently. This empirical evidence supports the importance of polynomial time as a practical measure of algorithm efficiency.