Computational complexity refers to the quantitative description of the resources required by an algorithm to solve a computational problem. Typical resources include execution time, memory usage, number of processor instructions, and any other measurable quantity needed to complete the task. This complexity is usually expressed as a function of the input size , where the output is the amount of the selected resource needed to process an input of size .

There are generally two ways to determine computational complexity: a **theoretical approach**, which derives expressions from the algorithm’s structure and mathematical properties, and an **empirical approach**, which measures the actual resource consumption during execution. For some algorithms, obtaining a theoretical expression is straightforward; however, for complex problems with multiple nested operations, varying data sizes, and nontrivial memory-access patterns, the theoretical derivation becomes cumbersome or impractical. In such cases, an empirical strategy offers a direct and reproducible means of characterizing performance.

In this work we present an automated framework for experimental evaluation of computational complexity. The program has as input a dataset composed of different input size folders; each folder could contain multiple samples of identical size. The program iterates through the folders, processes each sample using the algorithm under test, and records performance metrics using the Linux perf utility. These metrics include the number of executed CPU instructions, processor cycles, and wall-clock execution time.

Once all samples have been processed, the program applies statistical regression to model the relationship between input size and each measured resource. The resulting regression curves provide an **experimental profile of the algorithm’s computational complexity**, enabling quantitative comparisons and evidence-based performance assessment without requiring a theoretical derivation.