**What is time complexity?**

Time complexity refers to the amount of time an algorithm takes to run, as a function of the input size.

**What is space complexity?**

Space complexity refers to the amount of memory an algorithm requires, as a function of the input size.

**What is Big O notation?**

Big O notation is used to describe the upper bound or worst-case scenario of an algorithm's time or space complexity.

**What does O(n) represent in terms of time complexity?**

O(n) represents linear time complexity, indicating that the algorithm's runtime grows proportionally with the input size.

**What does O(1) represent in terms of time complexity?**

O(1) represents constant time complexity, indicating that the execution time does not depend on the size of the problem or the size of the input

**What does O(n^2) represent in terms of time complexity?**

O(n^2) represents quadratic time complexity, indicating that the algorithm's runtime grows quadratically with the input size.

**What does O(log n) represent in terms of time complexity?**

O(log n) represents logarithmic time complexity, indicating that the algorithm's runtime grows logarithmically with the input size.

**What are some techniques to analyze time and space complexity?**

* Counting the number of basic operations performed by the algorithm.
* Identifying loops, recursive calls, and nested structures that affect the runtime.
* Analyzing the size of data structures used by the algorithm.
* Considering any algorithmic optimizations or shortcuts implemented.

**Why do we drop constants in runtime?**

When analyzing the time complexity of algorithms, we tend to drop constants to focus on how the algorithm's efficiency scales with the input size. This approach helps us compare and classify algorithms based on their overall growth rate, allowing us to prioritize scalability and efficiency over specific constant factors.

A picture containing text, screenshot, line, diagram

Description automatically generated