COMP 210: Data Structures

Description of Time Complexity

What is Time Complexity?

Time complexity is a way to measure how an algorithm's runtime grows as the size of its input increases. (This is also known as its *growth function.*) It is particularly important when dealing with very large datasets.

Time complexity is estimated by calculating the number of instructions that will be executed within a program, rather than by measuring the actual time it takes to execute, since that can vary based on factors such as the computer you're using, how many other programs are running at the same time, or the network load.

The "Big-Oh" Notation

To describe time complexity, we use **Big-Oh notation**. For example, an algorithm might be classified as $O(n)$ or $O(logn)$. Think of Big-Oh as a simplified way to talk about the **worst-case scenario** of a given algorithm. It gives us a broad classification of how an algorithm's performance scales with the size of its input (n).

Common Big-Oh Classifications

Here are some of the most common Big-Oh complexities you'll encounter, from most efficient to least efficient.

$O(1)$ **- Constant Time** An algorithm runs in constant time if its runtime is the same no matter how big the input is. A perfect example is accessing an element in an array by its index. It takes the same amount of time to get the 5th element from an array of 10 items as it does from an array of 10 million items.

$O(\log n)$ **- Logarithmic Time** Algorithms with logarithmic time complexity are very efficient. They work by repeatedly cutting the problem size in half. A great example is a **binary search**. If you're looking for a name in a sorted phone book with 1,000 pages, you don't have to check every page. You'd open to the middle, see if the name is in the first or second half, and then repeat the process, quickly narrowing down your search.

$O(n)$ **- Linear Time** In a linear time algorithm, the runtime grows in direct proportion to the size of the input. If you have to double the number of items you process, the runtime will roughly double. A simple `for` loop that iterates through every item in a list to find a specific value is a classic example of an $O(n)$ algorithm.

$O(n \log n)$ **- Log-linear Time** This is a very common complexity for efficient sorting algorithms like **Merge Sort** and **Quick Sort**. It's considered a "sweet spot" – much better than quadratic time ($O(n^2)$) but not quite as fast as linear time ($O(n)$).

$O(n^2)$ **- Quadratic Time** An algorithm runs in quadratic time when its runtime is proportional to the square of the input size. This often happens with **nested loops**, where for every element in the outer loop, you iterate through the entire input again in the inner loop. A simple **Bubble Sort** is an example of an $O(n^2)$ algorithm. If your input size doubles, the runtime increases by a factor of four. More generally, an algorithms runs in **Polynomial Time** if its runtime is proportional to a polynomial function of its input size, such as $O(n^3)$ or $O(n^5)$.

$O(2^n)$ **- Exponential Time** An algorithm runs in exponential time when its runtime is proportional to a function with the input size in the exponent, such as $O(2^n)$ or $O(3^n)$. For example, an algorithm whose runtime doubles with each addition to the input size would be exponential. This type of complexity is generally found in "brute-force" algorithms that test every subset or combination of the n inputs.

$O(n!)$ **- Factorial Time** In a factorial algorithm, the runtime is proportional to $n!$. This is a common variation of exponential time, that is is generally found in "brute-force" algorithms that test all permutations of the n inputs where ordering matters.

Exponential and Factorial algorithms are considered **intractable** because their running times grow so quickly relative to the size of the input that it becomes practically impossible to solve the problem in a reasonable amount of time, even with powerful computers.

"Authored" by Gemini, with significant additions and editing by Alyce Brady, 25 September, 2025. Other sources include Wikipedia and GeeksForGeeks.