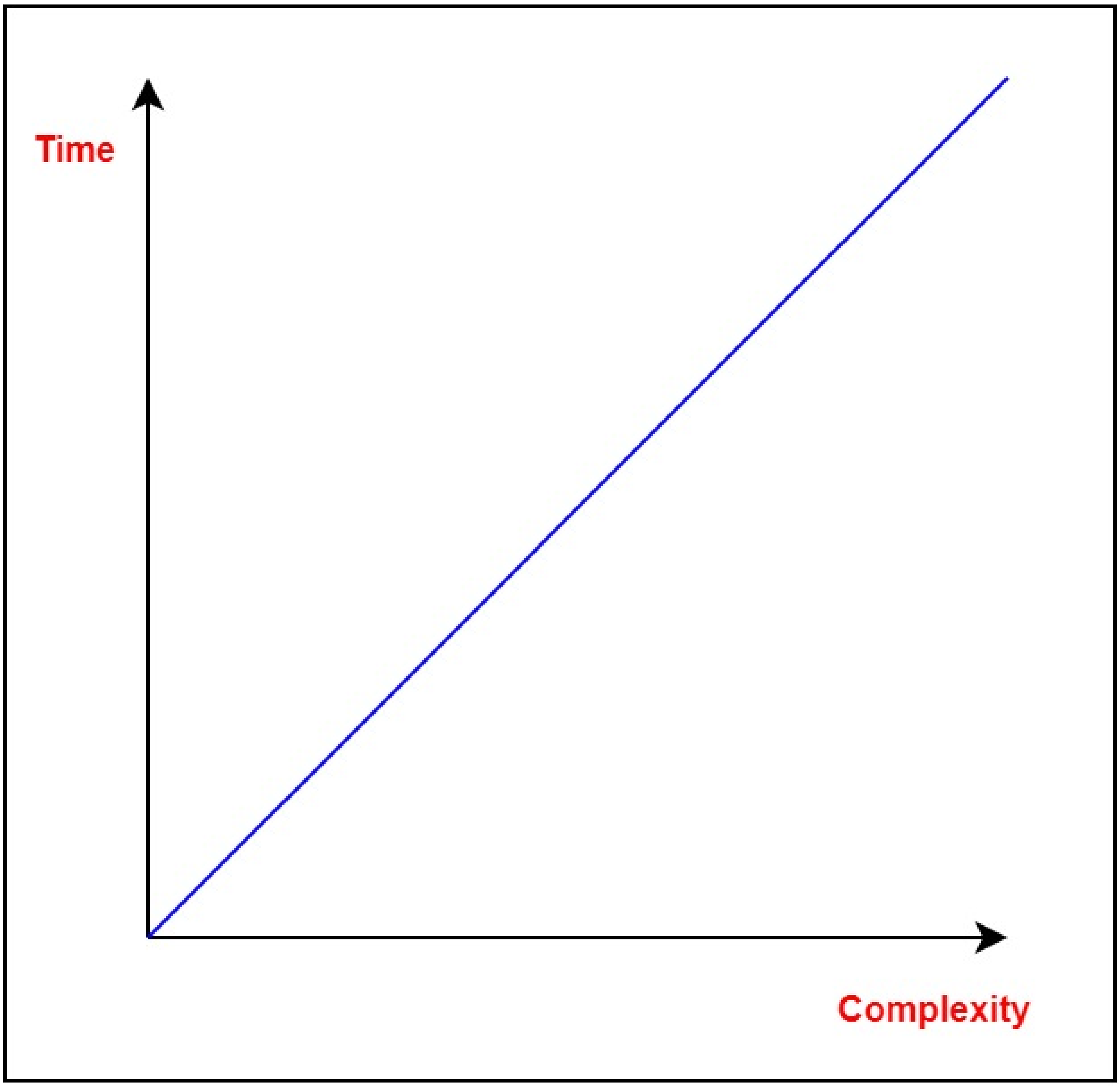
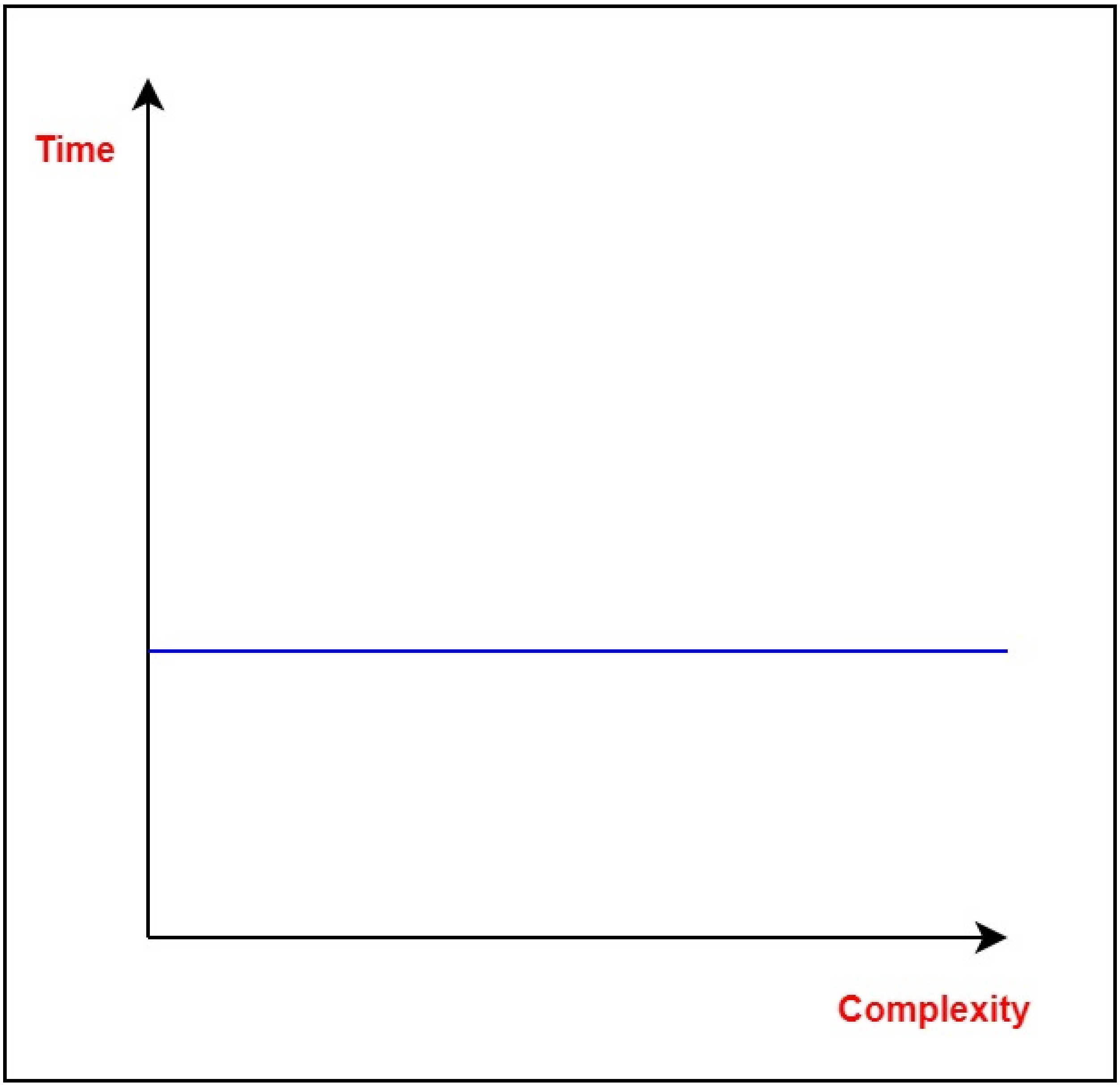
Asymptotic Notations

Hello everyone, and welcome to our new topic: **Asymptotic Analysis.** In this section, we will dive into the concept of asymptotic analysis and explore the powerful tool of asymptotic notations, which help us ***analyze the efficiency of algorithms***.

Efficiency of an algorithm is all about ***how well it utilizes resources*** like ***time*** and ***storage***. As you might have noticed, the performance of an algorithm can vary with different types of inputs. As the size of the input increases, the algorithm's performance might change significantly.



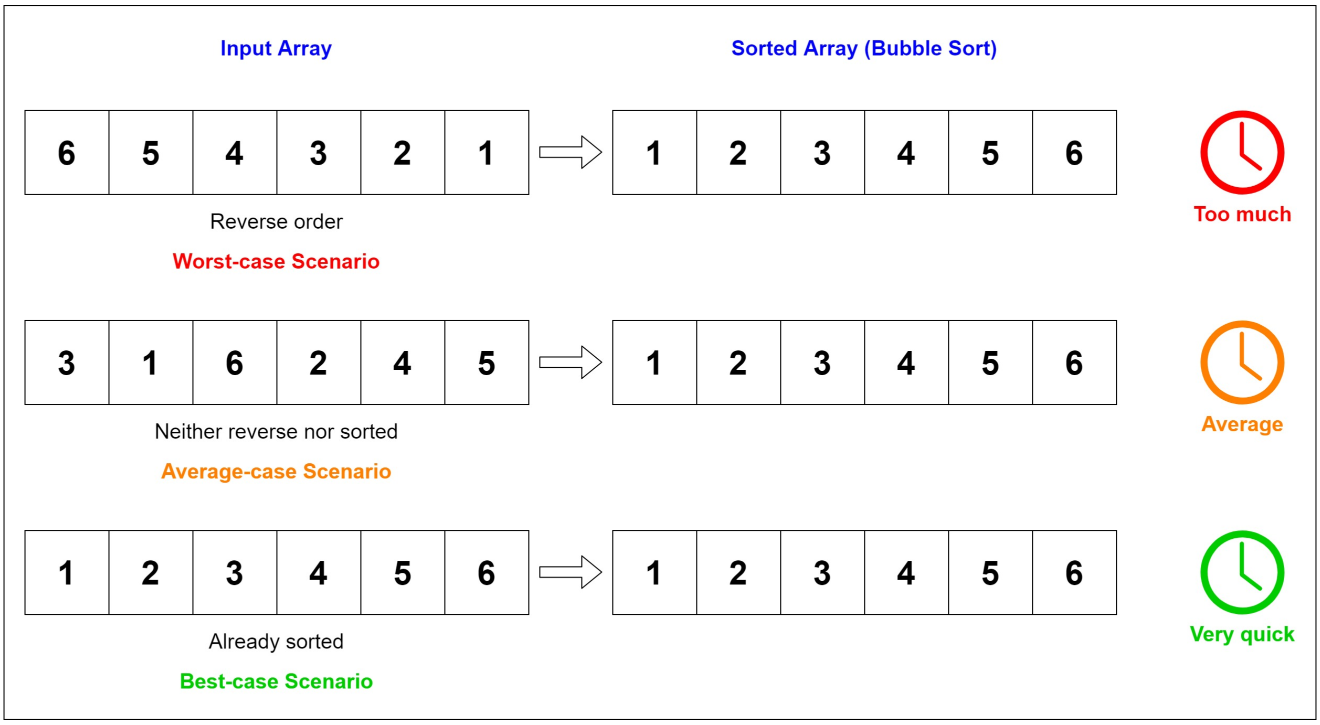


The ***study of how an algorithm's performance changes with the increase in the size of the input*** is what we call ***asymptotic analysis***. It allows us to understand how an algorithm behaves as the input becomes extremely large, and it is a fundamental concept in the analysis of algorithms.

Asymptotic Notations

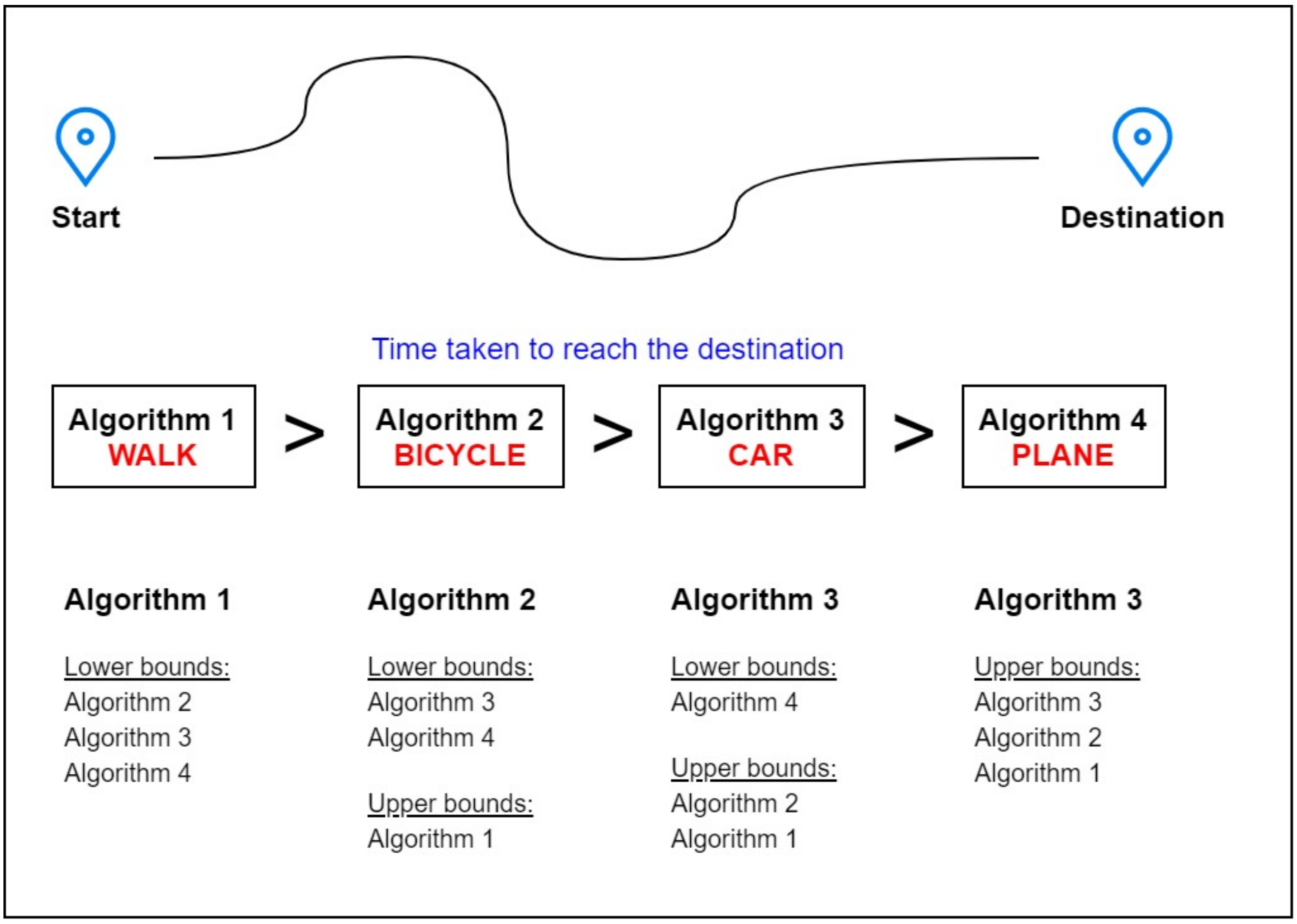
Asymptotic Notations are the mathematical tools we use to describe the running time of an algorithm as the input approaches a particular value or a limiting value. In simple terms, they help us understand how the running time of an algorithm changes with the size of the input.

Consider the example of **bubble sort**. When the input array is already sorted, the algorithm takes linear time, which is the best-case scenario. However, when the input array is in reverse order, it takes quadratic time, which is the worst-case scenario. And when the input array is neither sorted nor in reverse order, it takes an average amount of time. These different scenarios are captured and denoted using asymptotic notations.



There are ***three main asymptotic notations*** we will explore in this section:

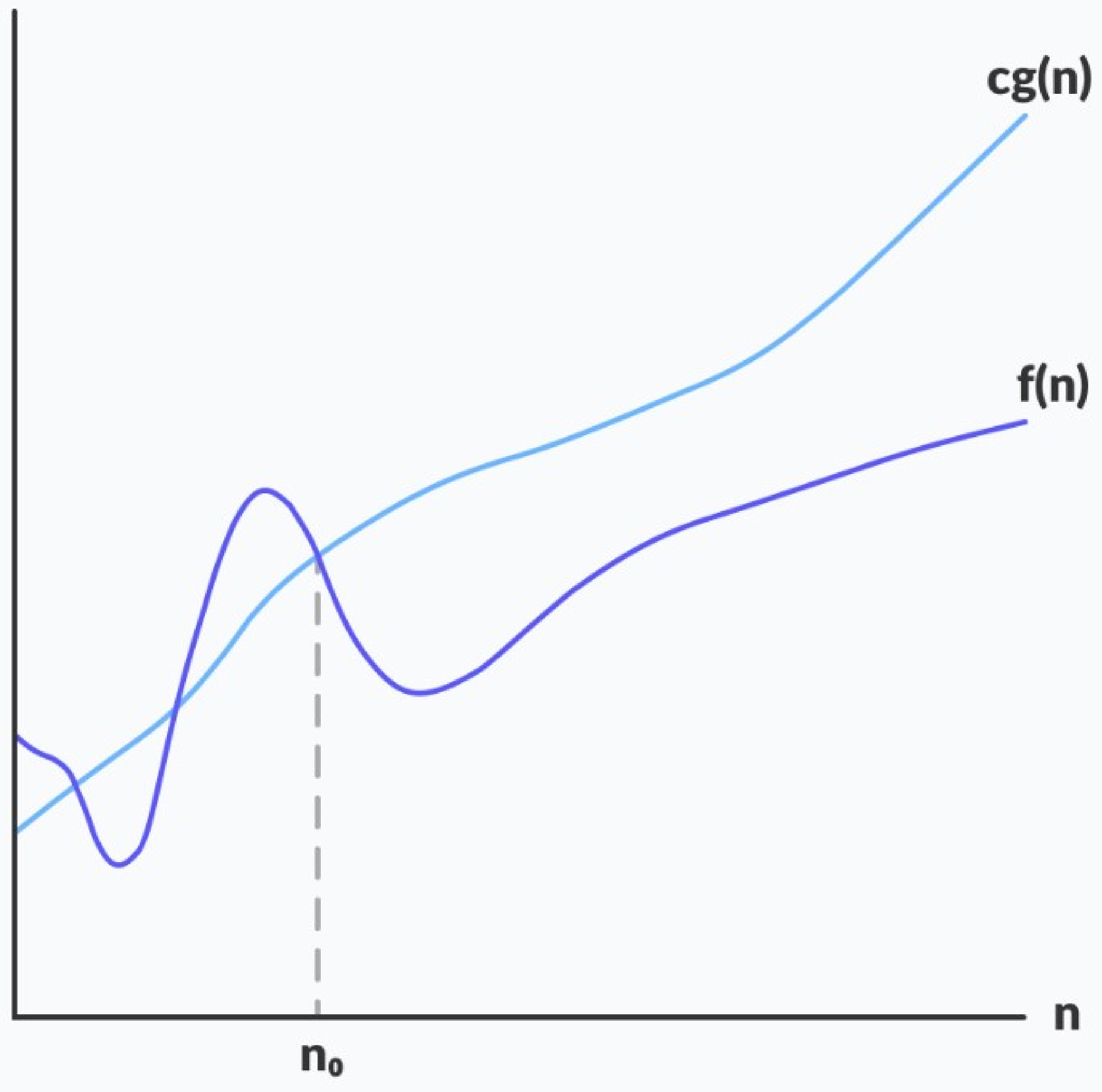
1. **Big-O Notation (O-notation)**: It represents the **upper bound** of the running time of an algorithm. It gives us the **worst-case complexity** of an algorithm.
2. **Omega Notation (Ω-notation)**: It represents the **lower bound** of the running time of an algorithm. It provides us with the **best-case complexity** of an algorithm.
3. **Theta Notation (Θ-notation)**: This notation **encloses the function from above and below**. It represents both the **upper and lower bounds** of the running time, making it suitable for **analyzing the average-case complexity** of an algorithm.



Let's take a closer look at each of these asymptotic notations and understand their significance.

Big-O Notation (O-notation)

Big-O notation gives us an **upper bound** on the running time of an algorithm. It tells us that the algorithm's performance won't exceed a certain level, even for large inputs. This makes it highly useful for analyzing the **worst-case scenario** of an algorithm.



$f(n) = O(g(n))$

```
O(g(n)) = {
  f(n): there exist positive constants c and n0
  such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0
}
```

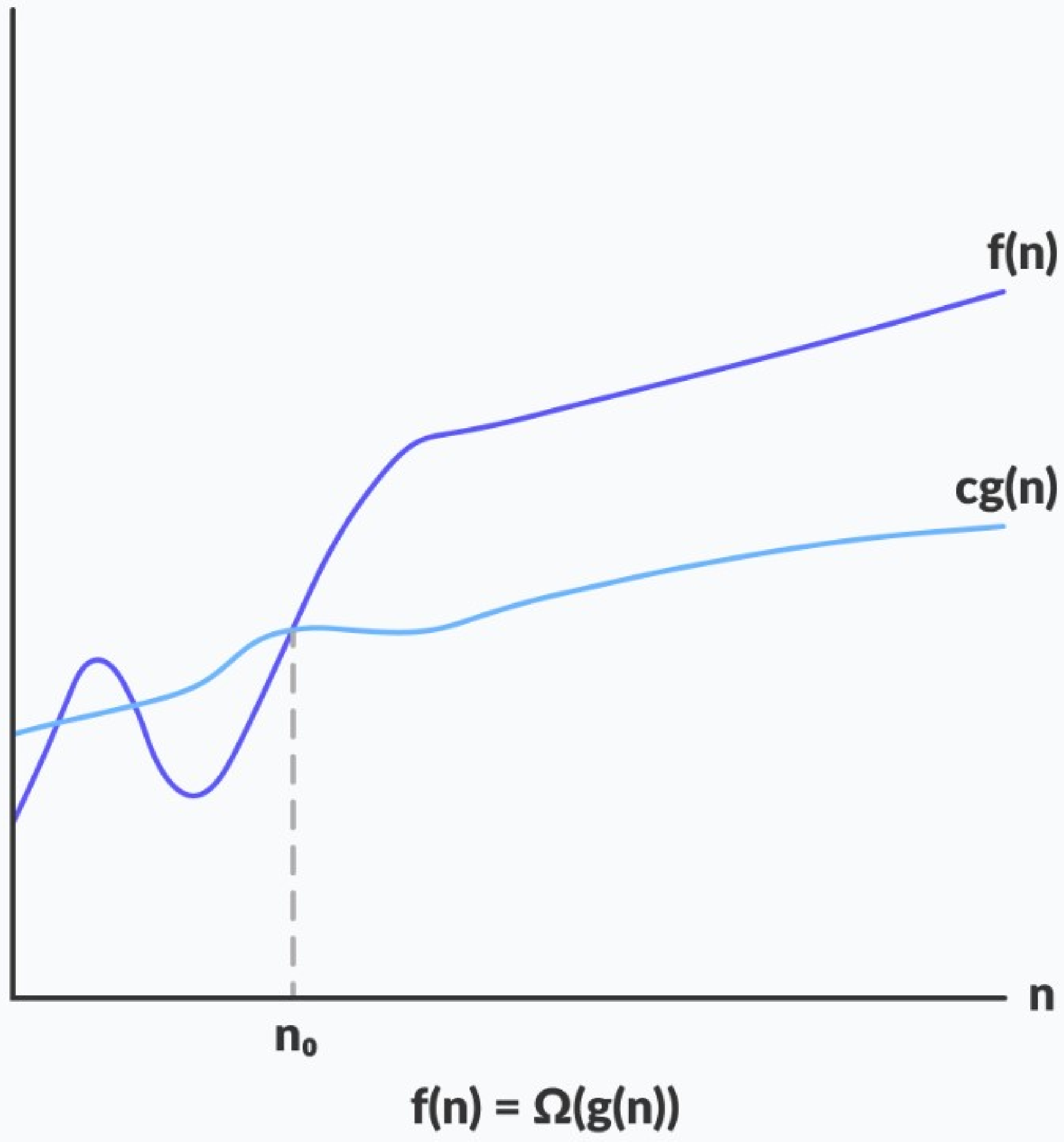
The above expression can be described as a function `f(n)` belongs to the set `O(g(n))` if there exists a positive constant `c` such that it lies between `0` and `cg(n)`, for sufficiently large `n`.

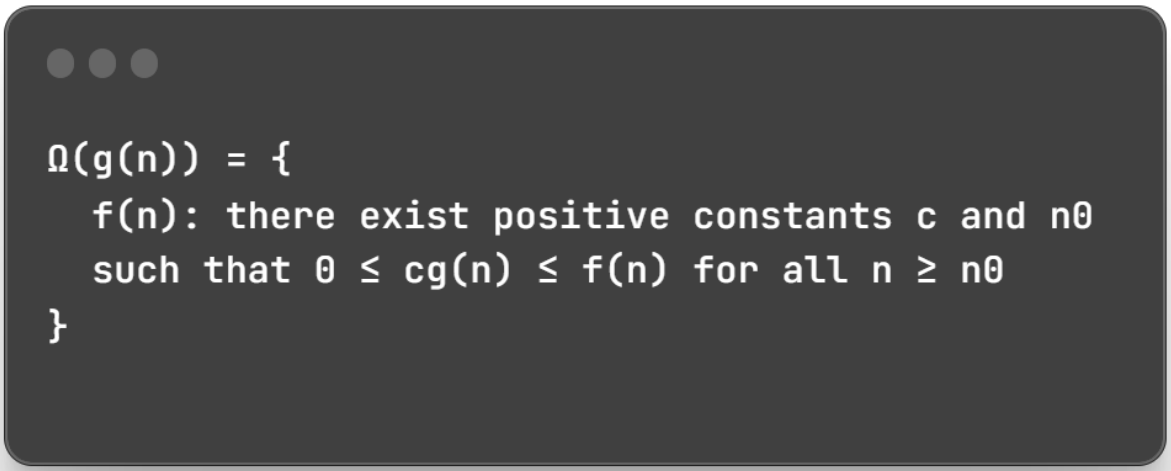
For any value of `n`, the running time of an algorithm does not cross the time provided by `O(g(n))`.

Since it gives the worst-case running time of an algorithm, it is widely used to analyze an algorithm as we are always interested in the worst-case scenario.

Omega Notation (Ω-notation)

Omega notation provides a **lower bound** on the running time of an algorithm. It assures us that the algorithm will perform at least as well as the specified bound, even for large inputs. Omega notation is handy for analyzing the **best-case scenario** of an algorithm.





```
Ω(g(n)) = {
  f(n): there exist positive constants c and n0
  such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0
}
```

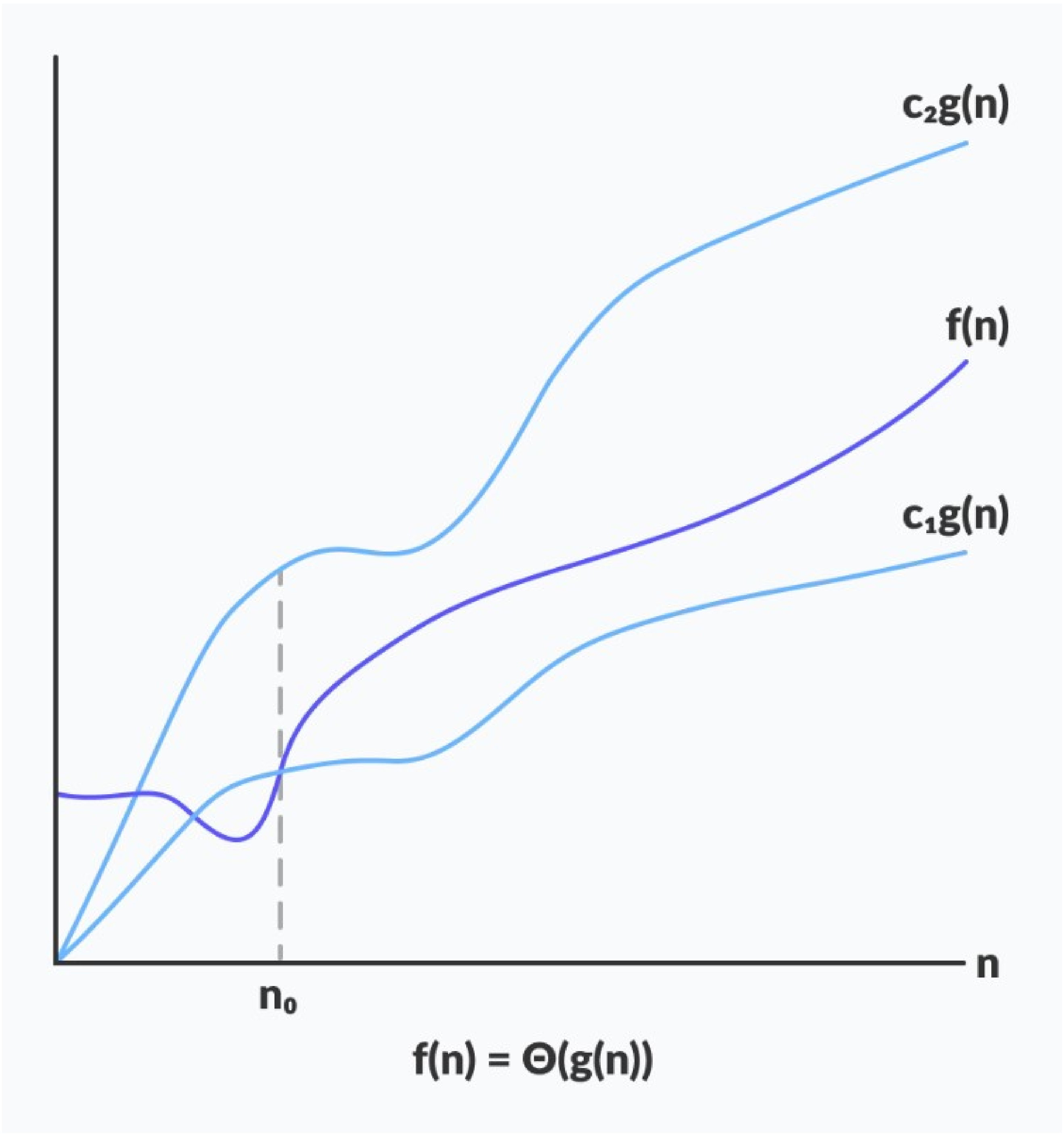
The above expression can be described as a function `f(n)` belongs to the set `Ω(g(n))` if there exists a positive constant `c` such that it lies above `cg(n)`, for sufficiently large `n`.

For any value of `n`, the minimum time required by the algorithm is given by Omega `Ω(g(n))`.

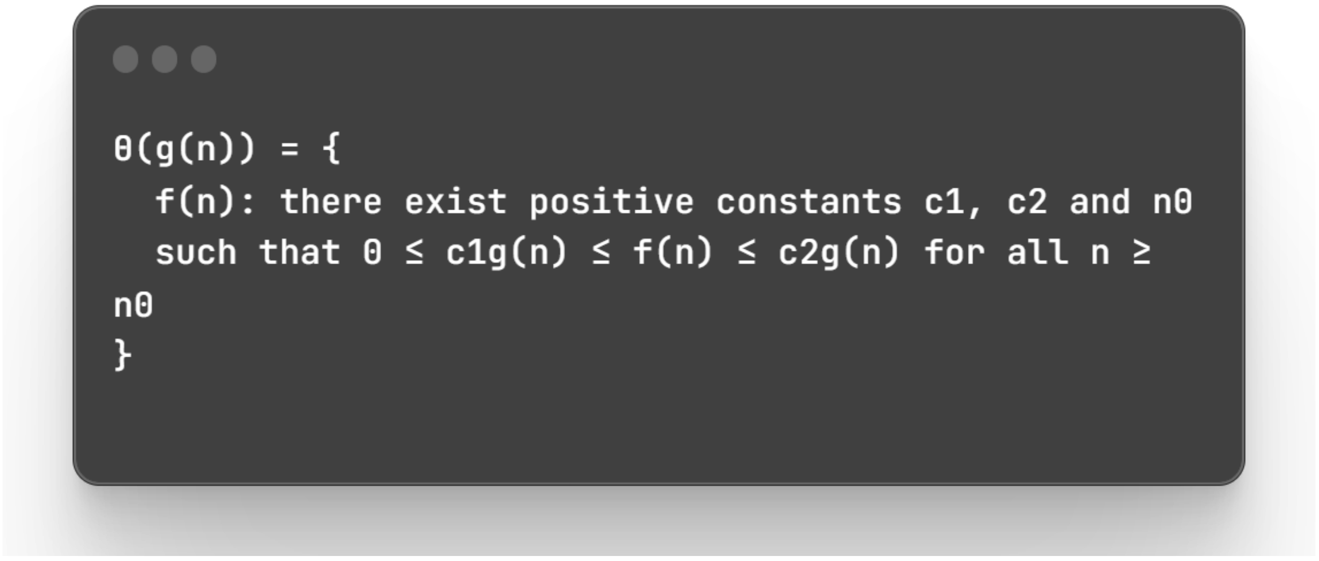
Theta Notation (Θ-notation)

Theta notation is like a sandwich;

It **encloses the function from above and below**. It represents both the **upper and lower bounds** of the running time. As a result, it is perfect for **analyzing the average-case complexity** of an algorithm when the performance can vary depending on the input.



For a function `g(n)`, `Θ(g(n))` is given by the relation:



```
θ(g(n)) = {
  f(n): there exist positive constants c1, c2 and n0
  such that 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) for all n ≥
n0
}
```

The above expression can be described as a function `f(n)` belongs to the set `Θ(g(n))` if there exist positive constants `c1` and `c2` such that it can be sandwiched between `c1g(n)` and `c2g(n)`, for sufficiently large n.

If a function `f(n)` lies anywhere in between `c1g(n)` and `c2g(n)` for all `n ≥ n0`, then `f(n)` is said to be asymptotically tight bound.