A expressão O(1) faz parte da **notação Big O**, que é usada para descrever a complexidade assintótica de algoritmos, ou seja, como o tempo ou o espaço requerido por um algoritmo cresce em função do tamanho da entrada. Aqui estão algumas variações dessa expressão que são usadas para descrever outras complexidades de tempo ou espaço:

1. O(n) - Complexidade linear

- A complexidade **linear** ocorre quando o tempo de execução do algoritmo cresce de forma proporcional ao tamanho da entrada. Ou seja, se você dobrar o tamanho da entrada, o tempo de execução também dobrará.
- Exemplo: Percorrer uma lista de tamanho n e realizar uma operação constante para cada elemento.

2. O(n^2) - Complexidade quadrática

- A complexidade **quadrática** ocorre quando o tempo de execução cresce proporcionalmente ao quadrado do tamanho da entrada. Este tipo de complexidade é frequentemente visto em algoritmos que utilizam loops aninhados.
- Exemplo: Algoritmos de ordenação como o **Bubble Sort** ou **Selection Sort**.

3. O(log n) - Complexidade logarítmica

- A complexidade **logarítmica** ocorre quando o tempo de execução cresce de forma proporcional ao logaritmo do tamanho da entrada. Esse tipo de complexidade é frequentemente visto em algoritmos que dividem repetidamente o problema em subproblemas menores, como na **busca binária**.
- Exemplo: Algoritmos que fazem busca em listas ordenadas ou em estruturas de dados como árvores balanceadas.

4. O(n log n) - Complexidade linear-logarítmica

- A complexidade **linear-logarítmica** ocorre quando o tempo de execução cresce

proporcionalmente ao tamanho da entrada multiplicado pelo logaritmo do tamanho da entrada. Este tipo de complexidade é comum em algoritmos eficientes de ordenação.
- Exemplo: Algoritmos como **Merge Sort** e **Quick Sort**.

5. O(n^k) - Complexidade polinomial (onde k é uma constante)

- A complexidade **polinomial** ocorre quando o tempo de execução cresce de acordo com uma potência do tamanho da entrada. A complexidade O(n^k), onde kk é um número fixo, é comum em algoritmos mais ineficientes, especialmente aqueles que exploram todas as combinações possíveis de entradas.
- Exemplo: Algoritmos que resolvem problemas combinatórios.

6. O(2^n) - Complexidade exponencial

- A complexidade **exponencial** ocorre quando o tempo de execução dobra a cada aumento unitário no tamanho da entrada. Algoritmos com essa complexidade são ineficientes e se tornam impraticáveis para entradas relativamente pequenas.
- Exemplo: Algoritmos que resolvem problemas NP-completos por força bruta, como a solução para o problema da mochila ou para o problema da sequência mais longa comum.

7. O(n!) - Complexidade fatorial

- A complexidade **fatorial** é um dos piores cenários, onde o tempo de execução cresce extremamente rápido. Isso ocorre, por exemplo, em algoritmos de força bruta que geram todas as permutações de uma lista.
- Exemplo: O problema do **caixeiro viajante** (Travelling Salesman Problem) na solução de força bruta.

8. O(√n) - Complexidade sublinear (raiz quadrada)

- A complexidade **sublinear** ocorre quando o tempo de execução cresce

proporcionalmente à raiz quadrada do tamanho da entrada. Isso é menos comum, mas ocorre em alguns algoritmos que fazem uma análise reduzida da entrada.

- Exemplo: Algoritmos de verificação de primalidade que utilizam a raiz quadrada do número.

Essas variações ajudam a descrever a eficiência de um algoritmo à medida que o tamanho da entrada cresce. Para comparar algoritmos, analisamos sua complexidade assintótica para entender o quão bem eles escalam à medida que a entrada se torna maior.