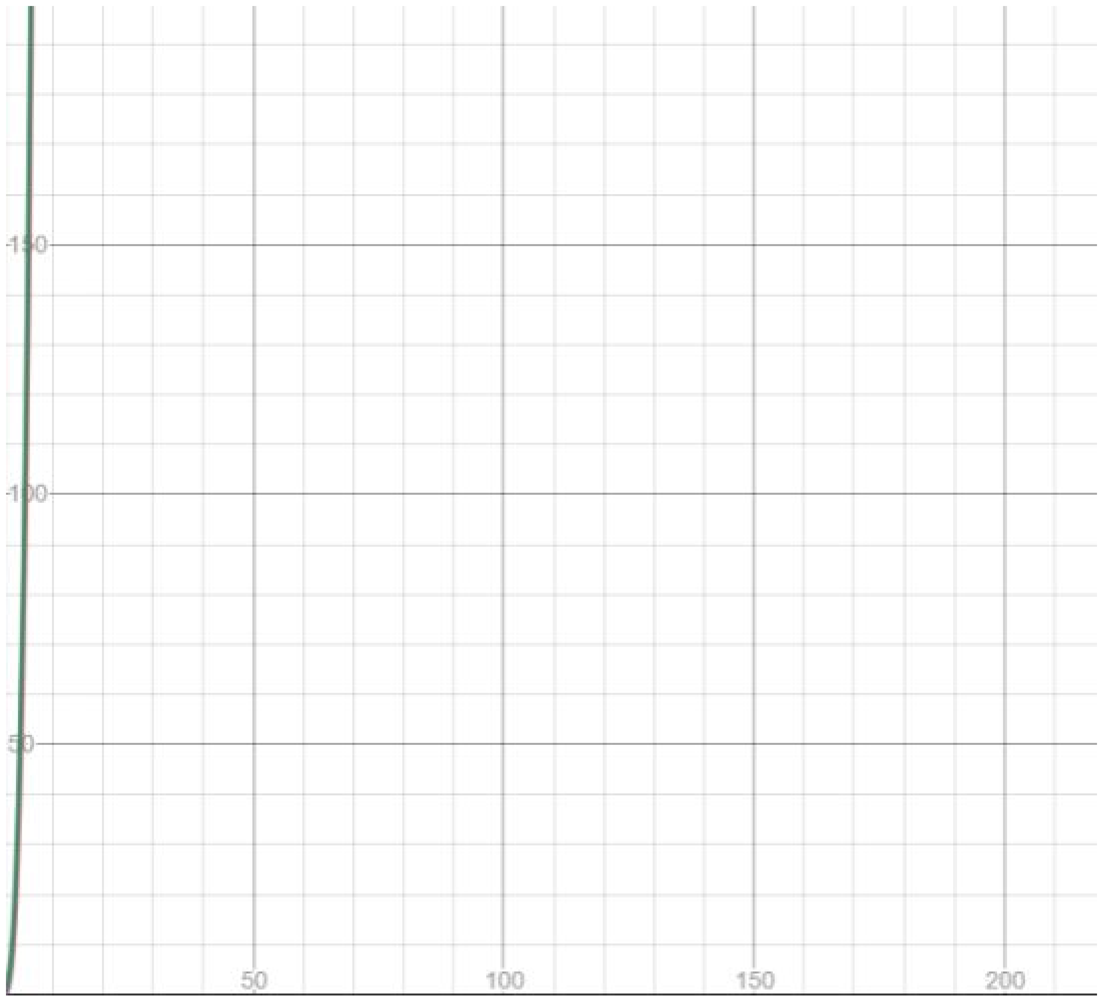
Análise de Algoritmos: Notação Assintótica

Como vimos na postagem passada (se não leu ainda, leia aqui), a análise de algoritmos é mais útil quando estamos lidando com problemas de larga escala. Além disso, para realizar a análise de um algoritmo a melhor abordagem é contabilizar o custo da execução das instruções do mesmo. O problema de contagem tradicional, que vimos na postagem passada, é que é bastante complexa, sendo computada em um nível de granularidade muito baixa onde precisamos saber o custo da execução de cada instrução de acordo com a arquitetura da máquina!

Com a **notação assintótica**, podemos realizar a análise da eficiência do algoritmo levando em consideração apenas o **tamanho da entrada** (para esse caso, o tamanho da lista). Além disso, a notação assintótica o foco da análise é na **taxa de crescimento do tempo de execução**, ignorando os fatores constantes (discutidos na postagem passada) e os termos de ordem inferior. Com relação aos termos de ordem inferior, a ideia é simples. Vamos assumir que temos 3 algoritmos: **A**, **B**, e **C**. O tempo de execução dos mesmos é, respectivamente, **$T(n) = n^3$**, **$T(n) = n^3 + n^2$**, e **$T(n) = n^3 + n^2 + n + 1$**, onde n é o tamanho do problema. Observando a imagem abaixo (as cores do texto e do gráfico estão mapeadas), podemos observar que o custo de **C** > **B** > **A**.

Com a imagem acima, temos a “ilusão” de que as taxas de crescimento do tempo de execução dos três algoritmos são diferentes, e isso é verdade, mas para valores de **n** pequenos! Se ampliarmos a área de análise (ver figura abaixo), podemos ver que os três são equivalantes para **n** suficientemente grandes!



Com isso, podemos concluir que apenas o **termo de ordem superior** (x^3) interessa quando estamos realizando a análise de algoritmos para problemas de larga escala. A pergunta é: **o que é que a notação assintótica tem a ver com isso?**

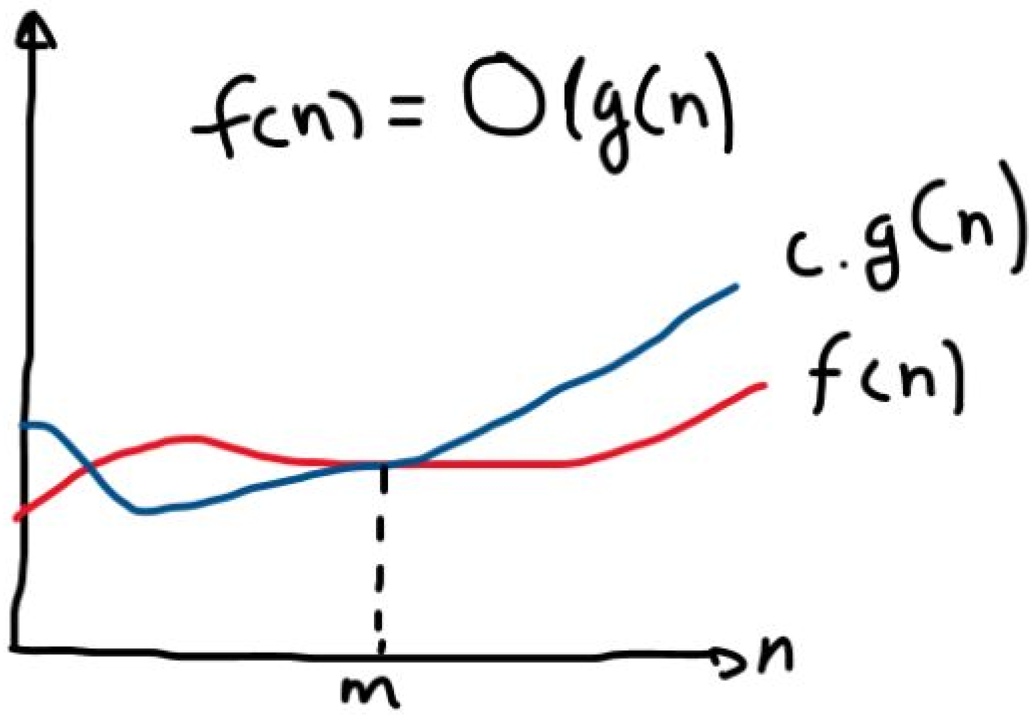
Não entrarei em tantos detalhes técnicos sobre a notação assintótica em si, pois foge do escopo desta postagem. Para quem quiser conhece-la mais profundamente, eu recomendo fortemente o ótimo livro do Eric Larman e seus colegas entitulado ***Mathematics for Computer Science***, que você pode acessar gratuitamente aqui. Quem sabe, eu não entro em mais detalhes sobre notação assintótica em outra postagem

Na postagem passada, ao analisar agoritmos por meio da **contagem**, a gente está realizaou o somatório de todos os custos das instruções executadas. A notação assintótica é uma ferramenta matemática para a gente mensurar de forma rápida o comportamento de uma função ***f(n)*** quando ***n*** é grande. Ela não é uma medida exata da eficiência de algoritmos, mas é boa o suficiente!

Nesta postagem, estudaremos três notações: **O** (lido como Big Ou), **Ω** (lido como Big Omega), e θ (lido como Big Theta).

Notação O

Vamos começar com uma definição informal. Considere uma função **f(n)**. Essa função representa o custo para a execução de um algoritmo. A notação **O** é útil para que possamos representar um limite superior do custo de execução de f(n). Por exemplo, para o algoritmo da **busca linear**, podemos afirmar que **f(n) = O(n)** (lido como **f(n)** tem um limite superior **n**). Isso significa que a **busca linear** tem um **limite superior assintótico linear**. Mas, o que isso significa? Significa que, no **pior caso**, dado um problema com tamanho grande o suficiente **m**, **f(n)** cresce mais lentamente do que **n**. Isso pode ser visualizado na imagem abaixo, considerando o caso genérico **f(n) = O(g(n))**.



Na figura acima, **c** e **m** são quais quer constantes postivas de forma que **f(n)** ≤ **c** * **g(n)** para todo **n** ≥**m**. É interessante notar que é possível definirmos várias funções

g(n) de modo que **f(n) = O(g(n))**. Por exemplo, vamos supor que **f(n) = n + 5**, g(n) poderia ser qualquer uma das opções abaixo:

- g(n) = 2*n
- g(n) = 10*n
- g(n) = n^2
- g(n) = 10*n*logn

Vamos verificar a primeira afirmação. Para tal, precisamos lembrar da fórmula abaixo:

$$f(n) \leq c * g(n) \quad \forall n \geq m$$

O nosso objetivo é encontrar valores para c e m que façam a expressão ser verdadeira. Podemos iniciar substituindo os valores de f(n) e g(n):

$$n + 5 \leq c * 2n \quad \forall n \geq m$$

Depois, podemos assumir que c = 1 (você pode escolher qualquer outro valor positivo, caso deseje). Substituindo na expressão, temos:

$$n + 5 \leq 2n$$

Agora, o objetivo é descobrir o valor de m. Para tal, podemos construir a tabela abaixo:

m	f(n) = n + 5	g(n) 2 * n
1	6	2
2	7	4

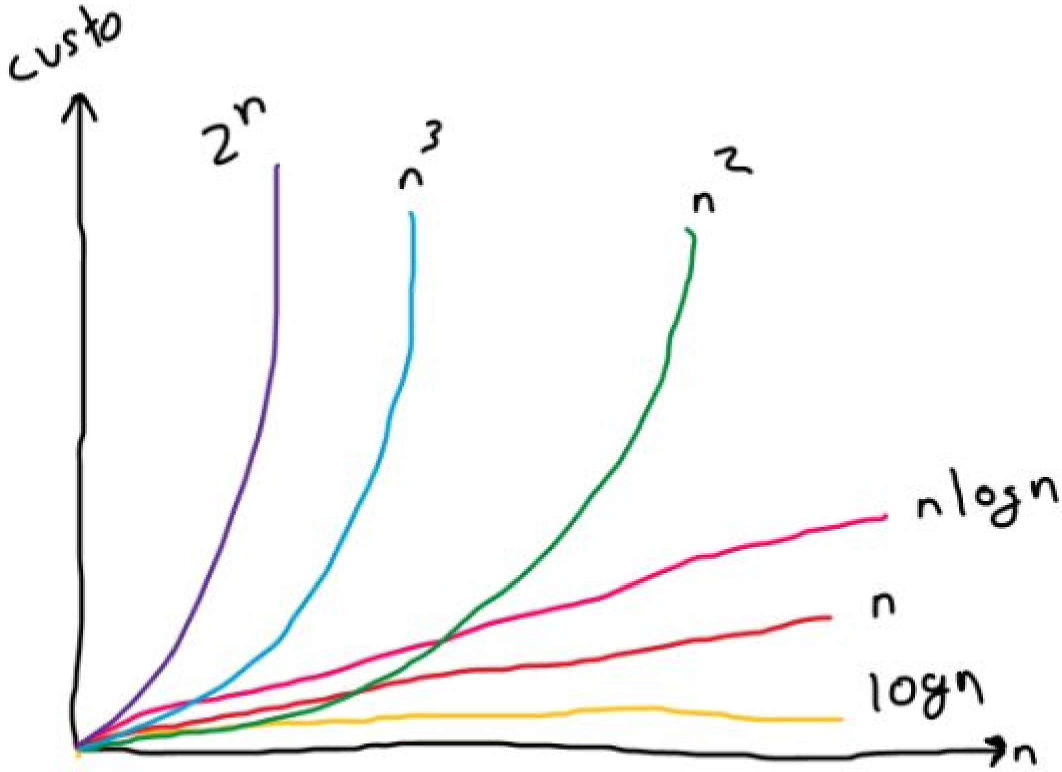
3	8	6
4	9	8
5	10	10
6	11	12
7	11	12
8	12	14

Conferindo os valores da mesma, podemos verificar que a partir de **m = 5** a função **g(n)** (ou seja, 2*n) cresce mais rapidamente do que **f(n)** (ou seja, n + 5). Com isso, fica provado que **n + 5 = O(2*n)**. Como exercício tente fazer para os demais casos. Qualquer dúvida, poste nos comentários aqui

---

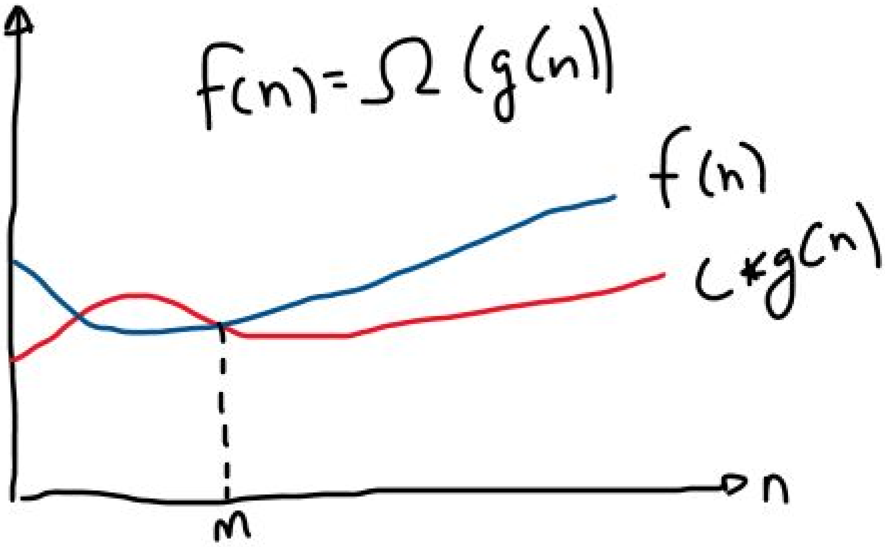
Então, na realidade, **O(g(n))** representa um **conjunto de funções que crescem mais rapidamente do que f(n)** a partir de certo ponto **m**. Por causa disso, alguns autores utilizam a notação **f(n) ∈ O(g(n))** para denotar que **f(n) tem limite superior assintótico g(n)**.

Na Figura abaixo, eu apresento as taxas de crescimento para algumas funções comuns (ou classes de algoritmos):



Notação Ω

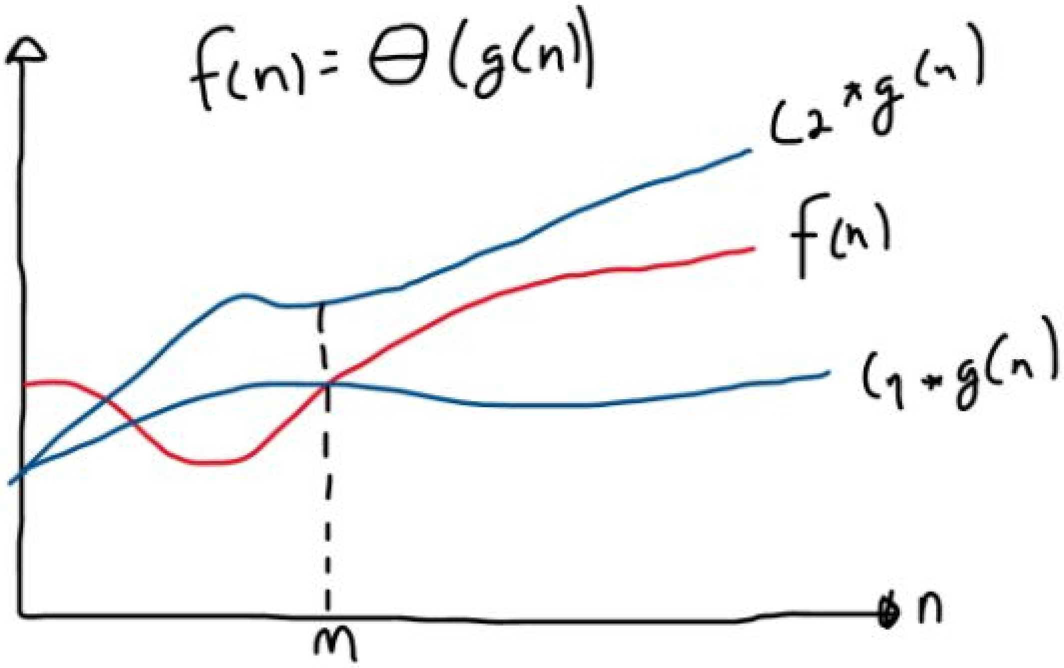
A **notação Ω** é similiar à O, mas ao invés de definir o limite superior assintótico, ela define o **limite inferior assintótico**. Ou seja, da mesma forma que O está relacionado ao pior caso, **Ω está relacionado ao melhor caso**. O comportamento da notação **Ω** é ilustrado abaixo:



Formalmente, temos que **f(n) = Ω (g(n))**, se existem duas constantes positivas **c** e **m** tais que **f(n) ≥ c * g(n)** para todo **n** ≥ **m**.

Notação θ

A **Notação θ** é uma afirmação assintoticamente mais forte do que **O**. **I**sso se dá pois limita-se a taxa de crescimento da função superiormente e inferiormente, como apresentado na figura abaixo:



Informalmente, temos que **θ(g(n))** significa que assim que **n** for grande o suficiente (**m**), o tempo de execução será pelo menos c1*g(n) e no máximo c2*g(n). Na prática, aplicar a **notação θ** é muito simples, basta ignorar os fatores constantes e os termos de ordem inferior.

Formalmente, temos que **f(n) = θ(g(n))** se existem três constantes positivas c1, c2 e m tais que 0 ≤ c1*g(n) ≤ f(n) ≤ c2*g(n), para todo n ≥ m. Alguns exemplos são apresentados abaixo:

- 2*n = θ(n)
- 2*n + 10 = θ(n)
- 10*n^2 + n – 123 = θ(n^2)
-

Como exercício, você pode verificar os exemplos acima utilizando a formalização apresentada. =)

Como apresentado anteriormente, podemos ter diversos **g(n)** tal que **f(n) = O(g(n))**. Por outro lado, é uma boa prática utilizarmos os **Princípios da Justeza** e

Simplicidade. O **Princípio da Justeza**afirma que se deve utilizar as notações para estabelecer relações com a **maior precisão possível**. Ou seja, evite usar, por exemplo, **n^2 = O(n^3)**, pois, o mais preciso é

n^2 = O(n^2). O Principio da Simplicidade afirma que se deve manter as funções o mais simples o possível. Ou seja, evite usar **n+1 = O(n+1)**, pois o mais simples é **n + 1 = O(n)**. Ou seja, ao aplicarmos os **Princípios da Justeza** e **Simplicidade** para a **notação O**, temos resultados similares à notação

θ. Uma ótima analogia é utilizada por Thomas Cormen e seus colegas no livro *Introduction to Algorithms*:

> "Suponha que você tem R$10 no bolso. Você pode chegar em um amigo e dizer: 'Eu tenho dinheiro no bolso, e posso garantir que não é mais que R$1.000.000.' A sua afirmação está correta, mas com uma precisão terrível."

É importante salientar que há um conflito entre a terminologia utilizada na indústria e na academia. Na indústria, utiliza-se **O** para denotar o que na academia (terminologia utilizada nesta postagem) denota-se com **θ**. Então, tome cuidado para não se confundir!

Na próxima postagem, vamos **provar matematicamente que a análise assintótica realmente faz o que se propõe**. Não se assuste com o "provar matematicamente", pois veremos que é bem simples. Posteriormente, vamos para a parte realmente interessante e começar a aplicar a análise assintótica em algoritmos com laços e condicionais, e até algoritmos recursivos!

Sumário

Nesta postagem, você aprendeu sobre:

- A **notação assintótica** como um mecanismo **mais simples** para realizar a medida aproximada da taxa de crescimento do custo de execução de um algoritmo.

- A **notação O** como meio para medirmos a taxa de crescimento do custo de execução de um algoritmo no **pior caso**.
- A **notação Ω** como meio para medirmos a taxa de crescimento do custo de execução de um algoritmo no **melhor caso**.
- A **notação θ** como meio para medirmos a taxa de crescimento do custo de execução de um algoritmo no **pior caso**. A diferença com relação à notação O é que é uma **afirmação mais forte**. Geralmente, na prática, utiliza-se o termo **notação O** para se referir para a **notação θ.**
- Na prática, aplicar a **notação θ** é muito simples, basta ignorar os fatores constantes e os termos de ordem inferior.