Gilberto Aula 3 Resumo

As três notações mais utilizadas são:

= “Big-O”
Ω = Omega
Θ = Theta

Elas estabelecem tipos de limite diferentes para uma determinada função f(n).

Big-O → Limite Superior / geralmente usada para definir a análise assintótica do PIOR CASO:

> O algoritmo não requer mais tempo que O(g(n)

$f(n) \leq c \cdot g(n)$

Determina ***um limite superior para uma f(n) por meio de uma g(n).***

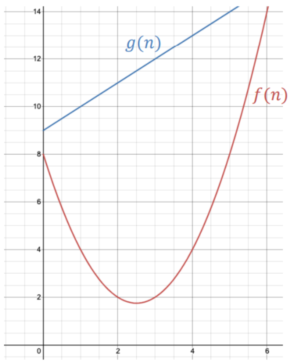
- **O tempo de execução do nosso algoritmo não pode nunca ser maior do que g(n) a partir de um determinado valor de entrada.**
- É interessante estudar o comportamento a partir de um número grande suficiente, porque se for pequeno ele pode mudar drasticamente.
- Limite superior = Big O!!! Nos dá uma ideia do tempo máximo que um algoritmo pode levar para ser executado. O(n^2) significa que, **no pior caso**, o algoritmo pode executar algo proporcional ao quadrado do tamanho da entrada.

> 💡 Exemplo: se tivermos um algoritmo com O(n^2), isso significa que, no pior caso, o algoritmo pode fazer até n^2 operações. Então, se meu n é 10, ele pode fazer até no MÁXIMO 10^2 = 100 operações.

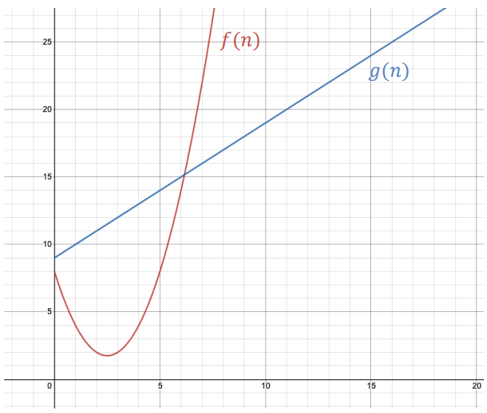
💡 Limite inferior = Big Omega!!! Quanto tempo um algoritmo deve levar para ser executado **no melhor cenário.**

Exemplo:

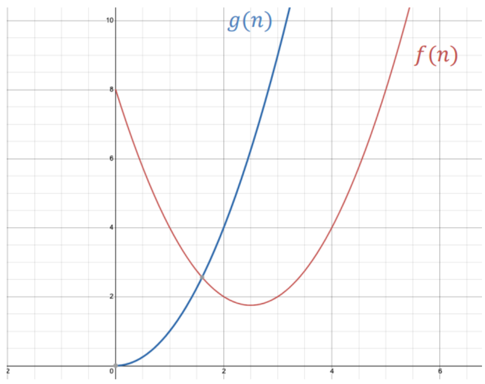
g(n) = 10n → o g(n) realmente ficaria maior do que *f(n) = n² - 5n + 8* para valores pequenos de n! por exemplo até 6:



Mas para a análise assintótica valores pequenos não importam tanto, então, quando aumentamos o valor de n isso acontece com a g(n):



Alternativamente, poderíamos pensar em g(n) = n²



Mas agora temos um problema novo, g(n) só é um limite superior para f(n) para valores de n≥2.
Visto que na análise assintótica, não importa o comportamento do algoritmo para valores pequenos de n utilizamos uma constante n0 para informar a partir de que ponto o limite é válido.

Lê-se f O(g) como:

"A função f está em Ó de g" ou...
"A função f está na ordem Ó de g" ou ainda...
"A função f é Ó de g"

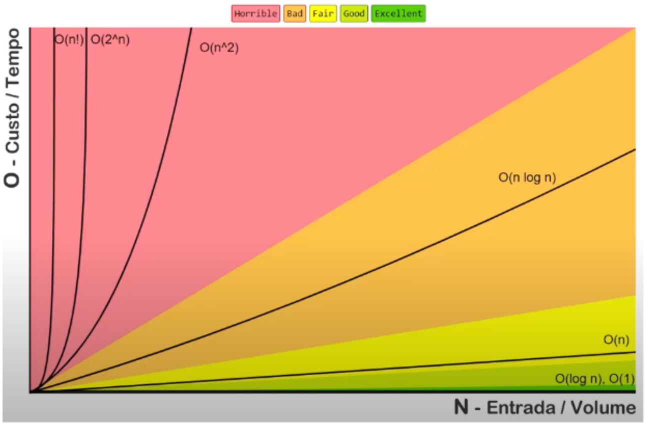
De modo geral:

- Sejam f(n) e g(n) duas funções de inteiros não negativos
- Dizer que f=O(g) significa que a função f não cresce mais rapidamente que a função g.
- Não tem nada que impeça que f e g cresçam na mesma proporção.
- Dizer que f=O(g) é uma leve analogia a dizer que f ≤ g

Conclusão

Estamos **interessados em determinar** o que chamamos de **taxa de crescimento/curva de crescimento do tempo** de execução do algoritmo.

Algumas delas: linear, quadrática, cúbica, logarítmica, etc;



Outro exemplo: $f(n) = 2n^2 + 1$ (vamos ignorar o 1 porque ele se torna insignificante)

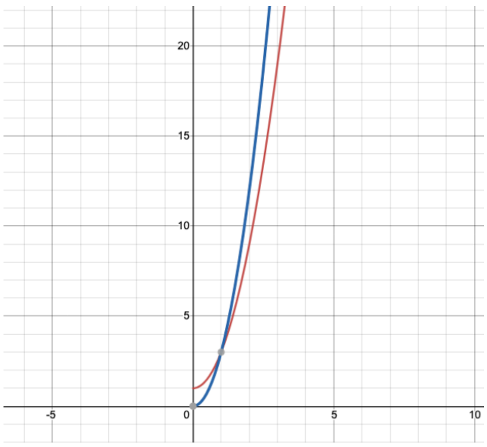
Não podemos concluir que $g(n) = n^2$ é um limite superior válido nesse caso porque

`f(n) = 2n² > g(n) = n²`

Para resolver isso, além de n0, temos à disposição uma segunda constante, chamada de C.

- Constante positiva

> 💡 Ela resolve o problema porque f(n) não vai crescer mais rapidamente do que c.g(n)

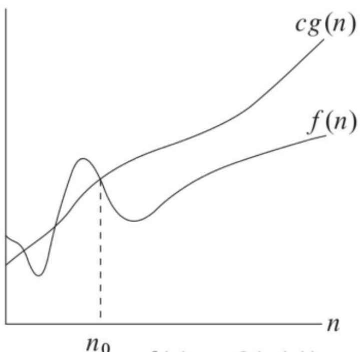


Com *c=3* já é possível "encaixar" g na definição. (c=3 não é o único valor possível)

💡 O que são n0 e c?
- n0: é o valor a partir do qual começamos a medir o crescimento da f(n) em relação a função g(n). É um ponto a partir do qual a nossa análise faz sentido. Não nos preocupamos com valores menores que n0.
- c: é uma constante positiva que serve como fator de escala para garantir que f(n) não cresça mais rápido que c.g(n) para n≥ n0.

Logo, f(n) = O(g(n)) se, e somente se

- f e g são funções assintoticamente não-negativas.
- Existem duas constantes positivas c e n0, tais que: f(n) ≤ c.g(n) para todo n≥ n0



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

💡 Não precisamos obrigatoriamente que a relação entre f(n) e g(n) (QUE É DADA POR f(n) ≤ c . g(n)) seja verdadeira para todos os possíveis valores de n.
Em vez disso, só nos importamos com o comportamento da função para valores grandes de n.