INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Campos do Jordão

Estruturas de Dados

Análise Assintótica

Prof. Igor de Moraes Sampaio
igor.sampaio@ifsp.edu.br

Análise Assintótica

No material introdutório de análise de algoritmos aprendemos a definir a função que descreve o custo de execução de algoritmos. Vimos exemplos simples cujas funções são também simples. Contudo, vamos supor que a função que descreve o tempo de execução de um algoritmo é dada por:

$$1.1 * n^2 + (10 + sin(n + 15) * n^{1.5}) + 9000$$

Análise Assintótica

A Solução. Simplificar.

A ideia é determinar como o algoritmo se comporta para valores muito grandes de entrada. Neste caso, ignoramos as constantes e os valores de menor magnitude por entender que eles não são significativos diante dos valores de maior magnitude.

Na prática, isso significa dizer que podemos:

- ignorar as constantes;
- ignorar os expoentes de menor magnitude.

Análise Assintótica

No exemplo fictício da função acima, podemos então fazer as seguintes simplificações.

$$\cancel{1.1 *}\, n^2 \cancel{+ (10 + sin(n + 15) * n^{1.5}) + 9000}$$

Análise Assintótica

A análise da ordem de crescimento de um algoritmo permite ignorar constantes e termos de menor magnitude para focar apenas no comportamento dominante. Para valores grandes de n, esses detalhes se tornam insignificantes, permitindo simplificar a expressão para $\Theta(n^2)$, por exemplo. Isso facilita a comparação entre algoritmos, tornando mais claro o impacto do tamanho da entrada no tempo de execução. Assim, escolher um algoritmo $\Theta(\log n)$ em vez de $\Theta(n)$ é vantajoso, pois o primeiro cresce mais lentamente à medida que a entrada aumenta.

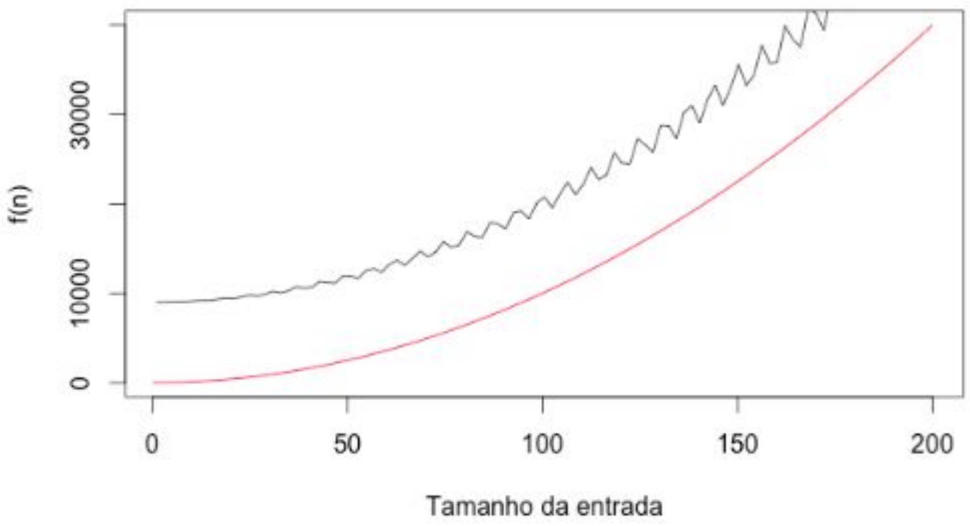
Análise Assintótica

Então $f(n) = 1.1 * n^2 + (10 + sin(n + 15) * n^{1.5}) + 9000 = g(n) = n^2$?

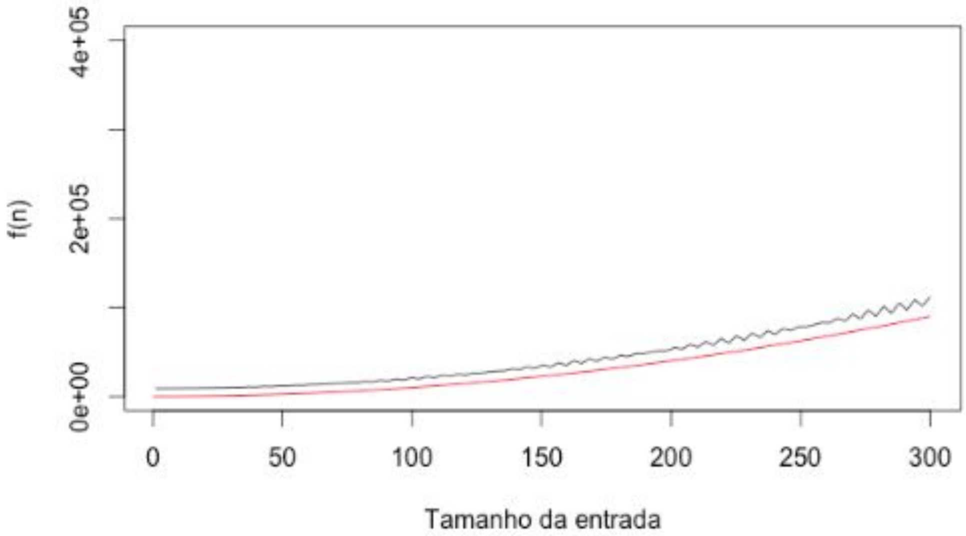
Não. E sim elas pertencem à mesma classe de funções, as funções quadráticas. Eu estou querendo dizer que essas duas funções possuem a mesma ordem de crescimento para grandes entradas e que se aproximam muito uma da outra para grandes valores de n.

Vou te mostrar. Os gráficos da sequência abaixo ilustra essas duas funções. $f(n)$ está destacada em azul e $g(n)$ em vermelho. A única diferença é que a entrada (eixo x) vai aumentando de um gráfico para outro.

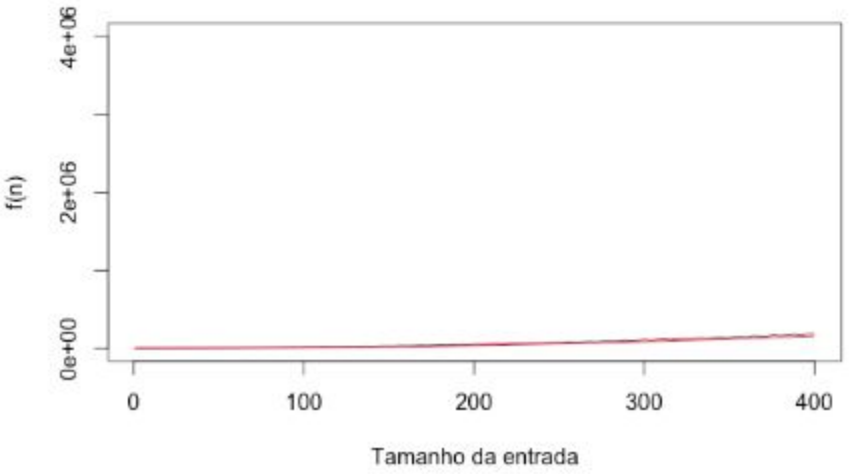
Análise Assintótica



Análise Assintótica



Análise Assintótica



A notação Θ

Agora vamos definir formalmente o que significa essa notação. Se encontrarmos c1, c2 e n0 que satisfaçam a equação, temos que f(n) é θ(g(n)). c1, c2 e n0 maiores que 0.

$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n), \forall n \geq n_0$$

Em um resumo bem simplista essa equação está dizendo que se a gente “imprensar” f(n) com g(n) multiplicada por duas constantes diferentes, dizemos que f(n) é Θ(g(n)).

.

A notação Θ

$$g(n) = n$$

$$f(n) = 3 * n + 1$$

$$c_1 = 1, c_2 = 6$$

$$0 \leq 1 * n \leq 3 * n + 1 \leq 6 * n$$

$$n = 1$$

$$0 \leq 1 * 1 \leq 3 * 1 + 1 \leq 6 * 1$$

$$0 \leq 1 \leq 4 \leq 6$$

A notação Θ

$$3 * n + 1 = \Theta(n)$$

- Formalmente, dizemos que g(n) é um limite assintótico restrito para f(n).

- f(n) é limitada inferiormente e superiormente por f(n) = n.
  - Na verdade, todas as funções lineares são.

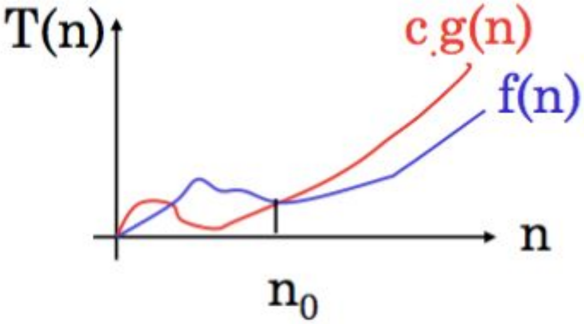
Análise Assintótica

- **Queremos também ser capazes de dizer:**
  - Big O: g(n) é um limite superior para f(n).
  - Ω: g(n) é um limite inferior para g(n).
- **E ainda tem mais...**
  - o: g(n) é um limite superior (não incluso) para f(n).
  - ω: g(n) é um limite inferior (não incluso) para g(n).

A notação Big O

Se encontrarmos c e n0 que satisfaçam a equação, temos que f(n) é O(g(n)).

$$0 \leq f(n) \leq c * g(n), \forall n \geq n_0$$

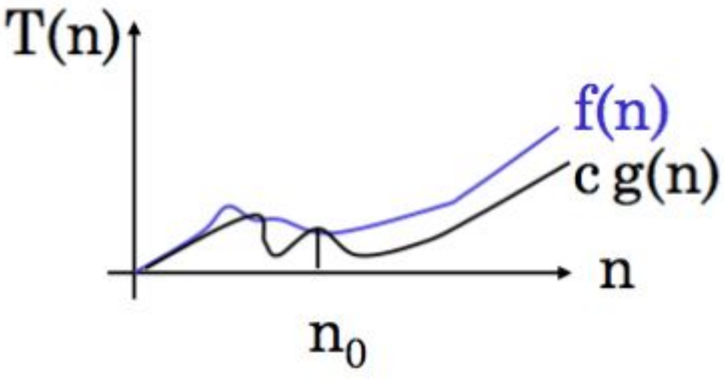


Note que g(n) é apenas o limite superior.

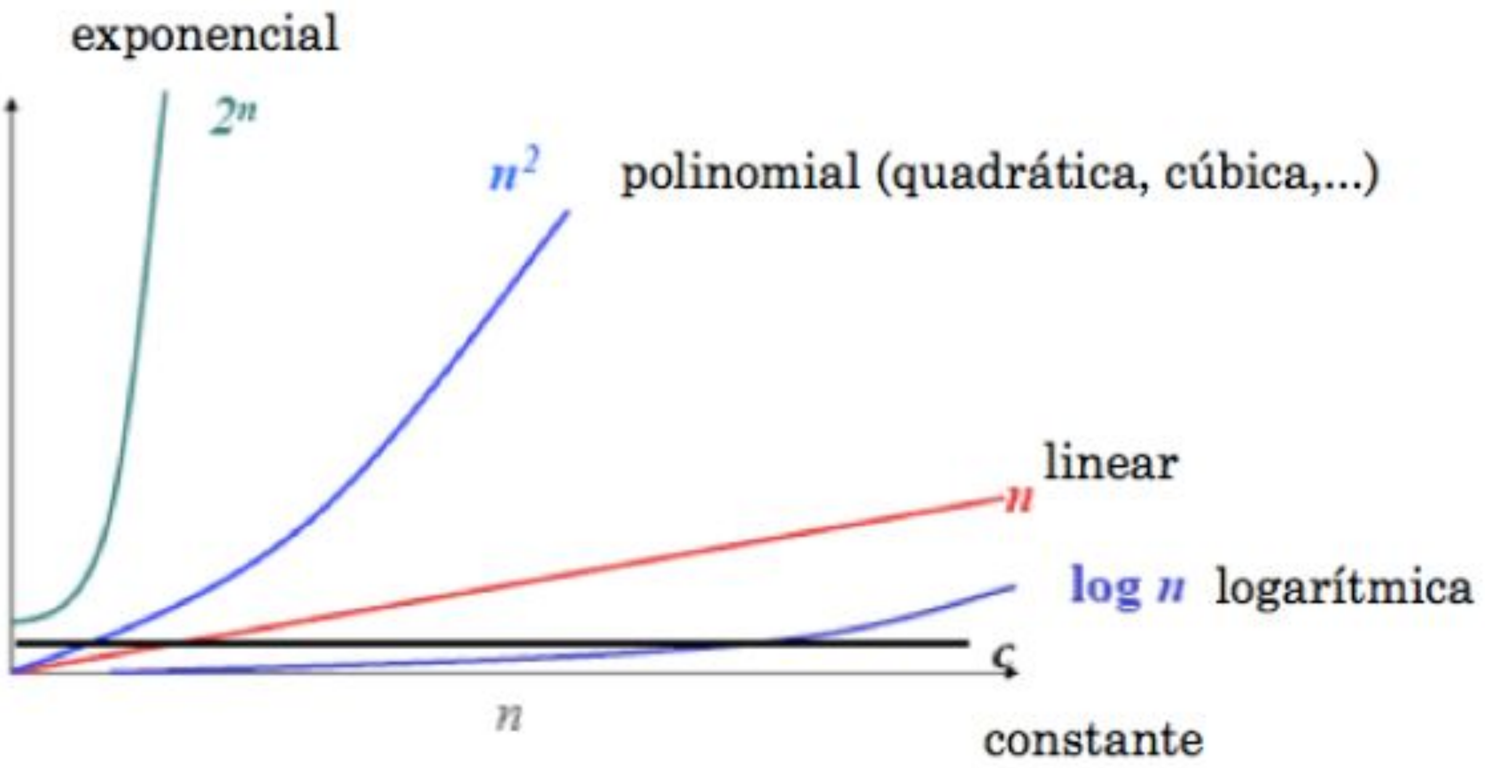
A notação Omega (Ω)

A notação Omega (Ω) define apenas o limite inferior. Para duas funções f(n) e g(n), dizemos que f(n) é Ω(g(n)) se:

$$0 \leq c * g(n) \leq f(n), \forall n \geq n_0$$



Classes Importantes



Conclusão

Análise assintótica leva em consideração grandes entradas para tornar relevante apenas a ordem de crescimento das funções de tempo de execução.

Na prática, ignoramos as constantes e os expoentes de menor magnitude.

Usamos análise assintótica para simplificar a comparação entre funções. Aplicando as diretrizes conseguimos rapidamente determinar a que classe pertence uma função.

As principais classes são: 1< logn< n< n*logn< n2< n3...2n

Usamos notações para descrever as classes de complexidade das funções. Por exemplo, f(n) ∈ Θ(n) significa que f(n) é cresce linearmente de acordo com o tamanho da entrada.

Em uma simplificação grosseira, podemos dizer que:

O -> ≤

Ω -> ≥

Θ -> =

o -> <

ω -> >

Referência

- João Arthur Brunet, 2019. Estruturas de Dados e Algoritmos, Computação @ UFCG, <http://joaoarthurbm.github.io/eda>.