Introdução à Complexidade de Algoritmos

Prof. Flavio B. Gonzaga
flavio.gonzaga@unifal-mg.edu.br
Universidade Federal de Alfenas
UNIFAL-MG

Introdução à Complexidade de Algoritmos

- Existem duas características que são de extrema importância em algoritmos:
  - Tempo de execução;
  - Quantidade de memória utilizada;
- O tempo de execução é possível de ser determinado através de métodos empíricos…
- No entanto, é possível ainda obter uma ordem de grandeza do tempo de execução através de métodos analíticos.

Introdução à Complexidade de Algoritmos

- É necessário definir a variável em relação à qual a expressão matemática avaliará o tempo de execução;

  → A ideia portanto é exprimir o tempo de execução em função da *entrada*;

Exemplos

- Exemplo de pseudocódigo para a inversão dos elementos de um vetor:

```
tam := tamanho(v);
lim := tam/2;
para i := 1, …, lim faça
   temp := v[i];
   v[i] := v[tam-i + 1];
   v[tam-i + 1] := temp;
```

- Noção de complexidade de espaço:

  → 4 + n considerando as variáveis

- Noção de complexidade de tempo:

  → 2 + 4 * (n/2) → 2 + 2 * n

considerando os comandos de atribuição (antes e internamente no laço)

Exemplos

- Exemplo de pseudocódigo para a inversão dos elementos de um vetor:

```
lim := tamanho(v);
para i := 1, …, lim faça
   v2[i] := v[tamanho - i + 1];
```

- Noção de complexidade de espaço:

  → 2 + 2 * n considerando as variáveis

- Noção de complexidade de tempo:

  → 1 + 2 * n

considerando os comandos de atribuição (antes e internamente no laço)

Introdução à Complexidade de Algoritmos

- A tarefa de obter uma expressão matemática para avaliar o tempo de um algoritmo em geral não é simples;
- Logo, algumas simplificações serão assumidas:
  - Suponha que a quantidade de dados manipulado pelo algoritmo seja suficientemente grande;
  - Não serão consideradas constantes aditivas ou multiplicativas.

Exemplos

- Suponha que a quantidade de dados manipulado pelo algoritmo seja suficientemente grande;
- Não serão consideradas constantes aditivas ou multiplicativas;

```
tam := tamanho(v);
lim := tam/2;
para i := 1, …, lim faça
   temp := v[i];
   v[i] := v[n-i + 1];
   v[n-i + 1] := temp;
```

- Noção de complexidade de espaço:

  → 4 + n → n

- Noção de complexidade de tempo:

  → 2 + 4 * (n/2) → 2 + 2 * n → n

```
lim := tamanho(v);
para i := 1, …, lim faça
   v2[i] := v[tamanho - i + 1];
```

- Noção de complexidade de espaço:

  → 2 + 2 * n → n

- Noção de complexidade de tempo:

  → 1 + 2 * n → n

Exemplos

- Soma de matrizes ($C \leftarrow A + B$), com dimensão n x n:

```
para i := 1, …, n faça
   para j := 1, …, n faça
      c[i][j] := a[i][j] + b[i][j];
```

- Complexidade de tempo:

  $\rightarrow n^2$

Exemplos

- Multiplicação de matrizes (C ← A x B), com dimensão n x n:

```
para i := 1, …, n faça
   para j := 1, …, n faça
      c[i][j] := 0;
      para k := 1, …, n faça
         c[i][j] := c[i][j] + a[i][k] * b[k][j];
```

- Complexidade de tempo:

  → n^3

Noção de complexidade de tempo

- A noção de complexidade de tempo é descrita como:
  - Seja *A* um algoritmo, {*E_1*, …, *E_m*}, o conjunto de todas as entradas possíveis de *A*. Denote por *t_i* o número de passos efetuados por *A*, quando a entrada for *E_i*. Definem-se:
    - Complexidade do pior caso = $\max_{E_i \in E}\{t_i\}$
    - Complexidade do melhor caso = $\min_{E_i \in E}\{t_i\}$
    - Complexidade do caso médio = $\sum_{1 \leq i \leq m} p_i\, t_i$
      - Onde *p_i* é a probabilidade de ocorrer a entrada *i*
  - De forma análoga, podem ser definidas *complexidades de espaço*.

Noção de complexidade de tempo

- As complexidades têm portanto o objetivo de avaliar a eficiência de tempo ou espaço. A complexidade de tempo de pior caso corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução, isto é, para a entrada mais desfavorável.
- De certa forma, a entrada de pior caso é a mais importante dentre as três mencionadas, pois fornece um limite superior para o número de passos que o algoritmo pode executar.
- O termo complexidade será, então, empregado com o significado de complexidade de pior caso.

A Notação O

- Ao se considerar a grande quantidade de dados, e por consequência, o número de passos efetuados por um algoritmo, podem-se desprezar constantes aditivas ou multiplicativas;
  - Por exemplo, um valor de número de passos igual a $3n$ será aproximado para n.
- Como o interesse é restrito a valores assintóticos, termos de menor grau também podem ser desconsiderados;
  - Por exemplo, um valor de número de passos igual a $n^2 + n$ será aproximado para n^2;
  - O valor $6n^3 + 4n - 9$ será transformado em n^3.

A Notação O

- Torna-se útil, portanto, descrever operadores matemáticos que sejam capazes de representar situações como essas. As notações O, Ω e θ são utilizadas com essa finalidade.
- Sejam f e h funções reais positivas de variável inteira n. Diz-se que f é O(h), escrevendo-se f = O(h), quando existir uma constante c > 0 e um valor inteiro n_0, tal que:

$$n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$$

A Notação O

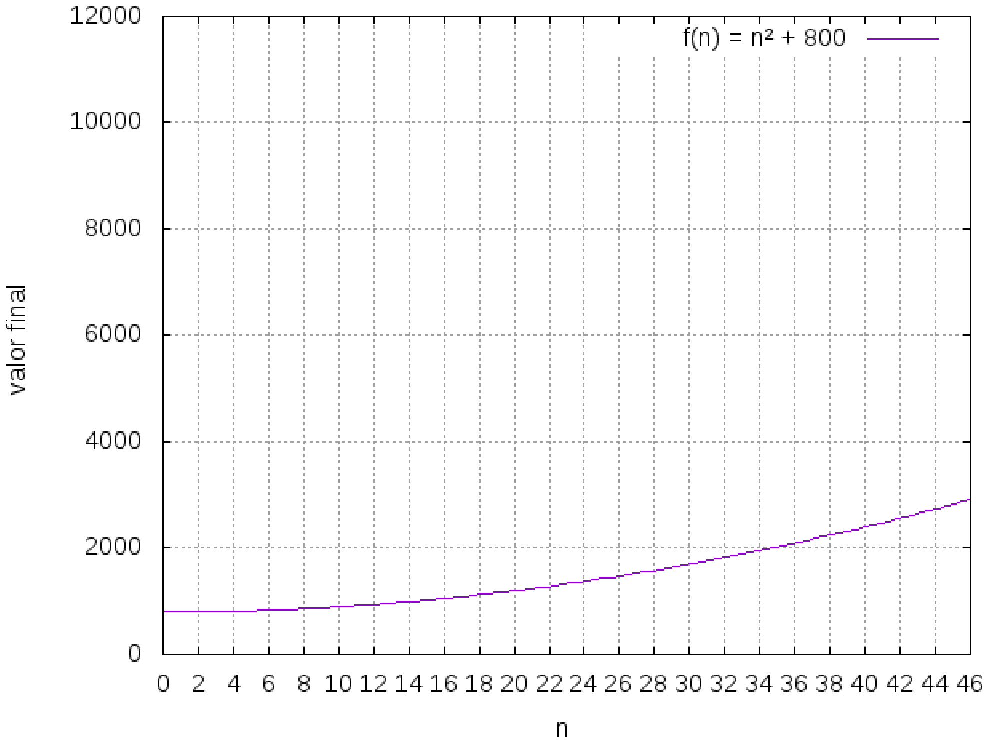
- Suponha então um algoritmo onde a complexidade de tempo dele possa ser expressada pela função *f*:
- $f(n) = n^2 + 800$
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação:

$$n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$$

  - Poderemos dizer que f é O(h).

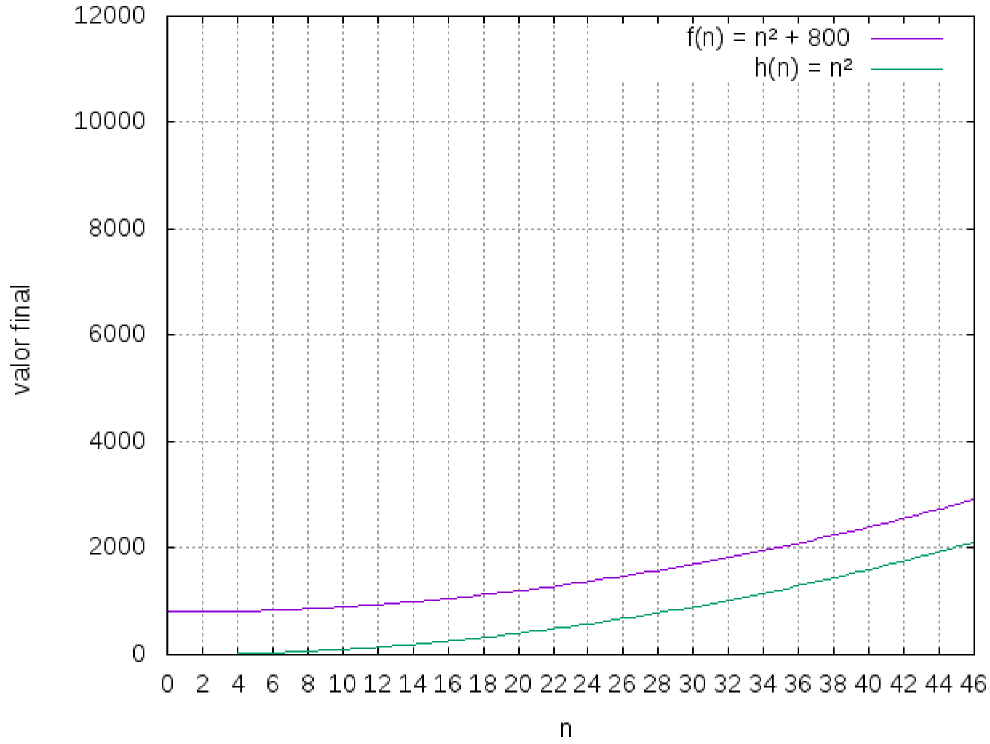
A Notação O

- f(n) = n² + 800
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação: $n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$
  - Poderemos afirmar que
    - f é O(h).



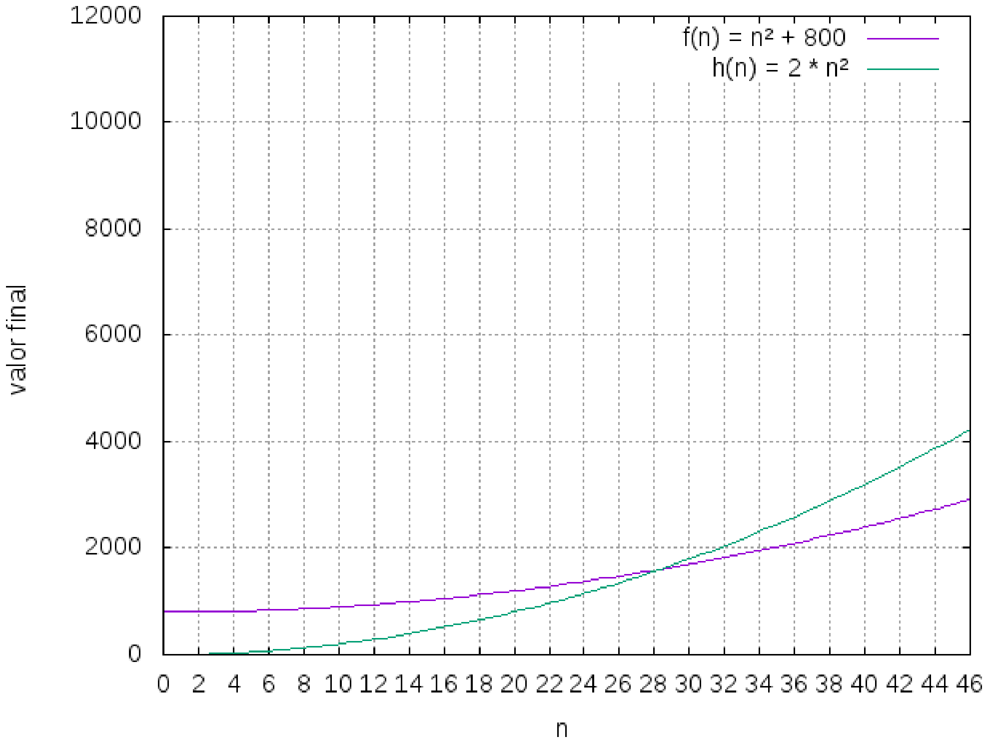
A Notação O

- f(n) = n^2 + 800
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação: $n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$
  - Poderemos afirmar que
    - f é O(h).



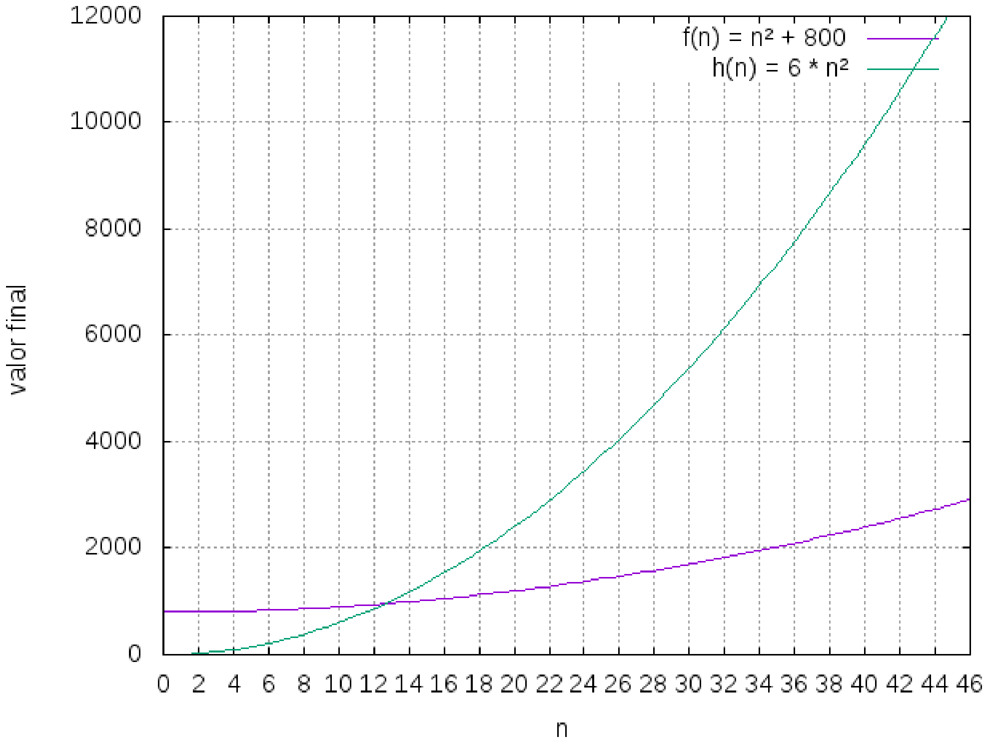
A Notação O

- f(n) = n² + 800
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação: $n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$
  - Poderemos afirmar que
    - f é O(n^2).



A Notação O

- f(n) = n² + 800
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação: $n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$
  - Poderemos afirmar que
    - f é O(n^2).



A Notação O

- f(n) = n² + 800
  - Se encontrarmos um valor de *n_0* e de constante *c* que satisfaça à relação:

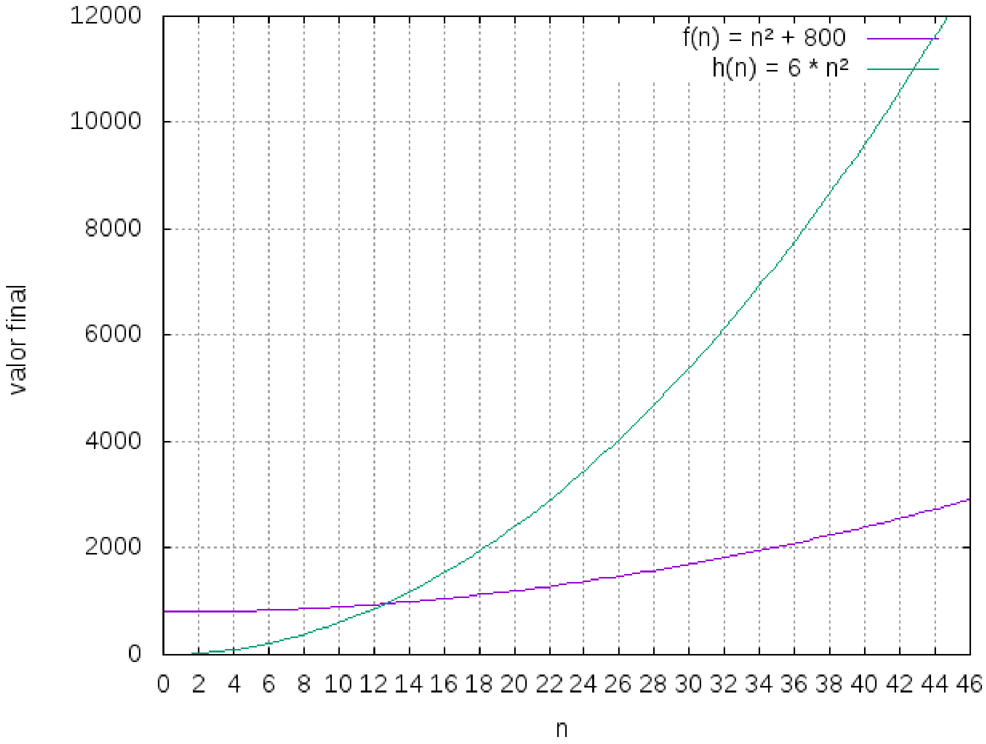
$$n > n_0 \Rightarrow f(n) \leq c \cdot h(n)$$

  - Encontramos graficamente

    um valor de n_0 = 14 e de constante

    c = 6 para a função h(n) = n².

    Logo:
    - A função f = O(n²)
    - Intuitivamente, é uma função que "não cresce mais rápido do que n²".



A Notação O

- As notações O, Ω e θ são utilizadas então com a seguinte ideia:
  - Se a comparação é ≤ (O);
  - Se a comparação é ≥ (Ω);
  - Se a comparação é = (θ);

Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos. Szwarcfiter J. L.; Markenzon L.. 3a Edição. Editora LTC. 2010.
- https://www.youtube.com/watch?v=ojCAnD7vrOY, acesso em 15/11/2021.