Breve Introdução à Complexidade Assintótica de Algoritmos

Letícia Rodrigues Bueno

Introdução

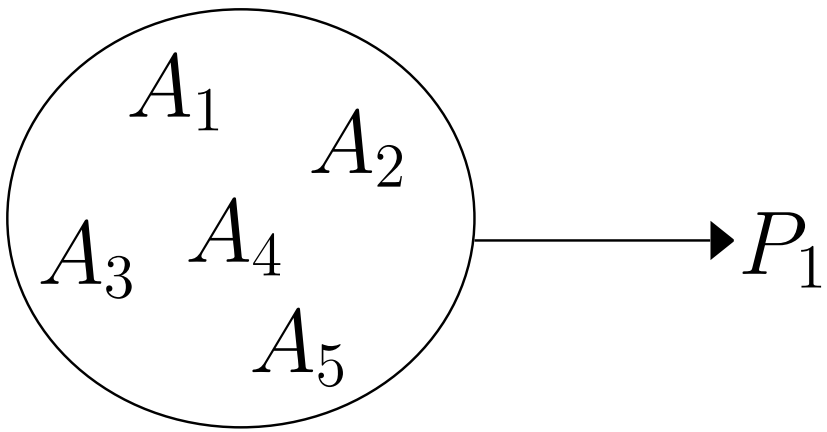
- **Objetivo**: possibilitar medir eficiência de algoritmos;

Introdução

- **Objetivo**: possibilitar medir eficiência de algoritmos;
- **Analisar um algoritmo**: prever os recursos que serão necessários;

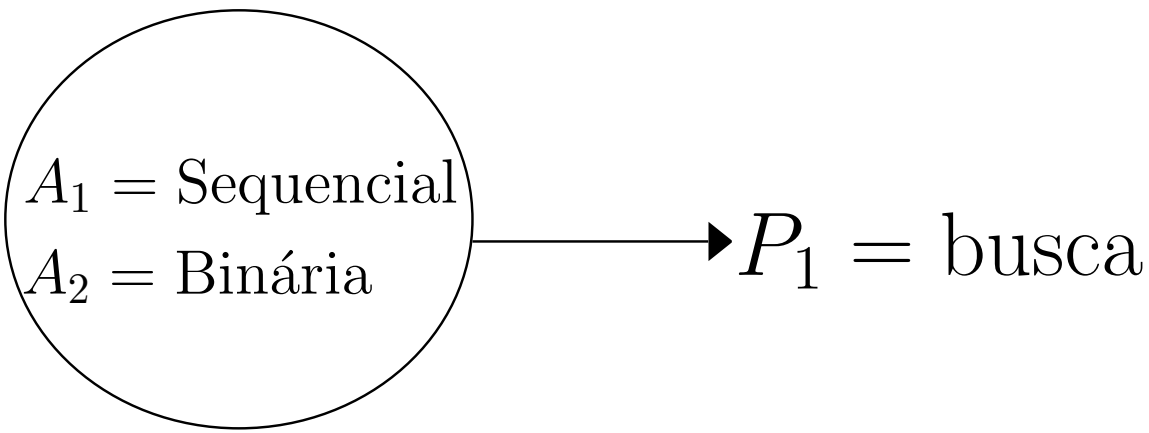
Introdução

- **Objetivo**: possibilitar medir eficiência de algoritmos;
- **Analisar um algoritmo**: prever os recursos que serão necessários;



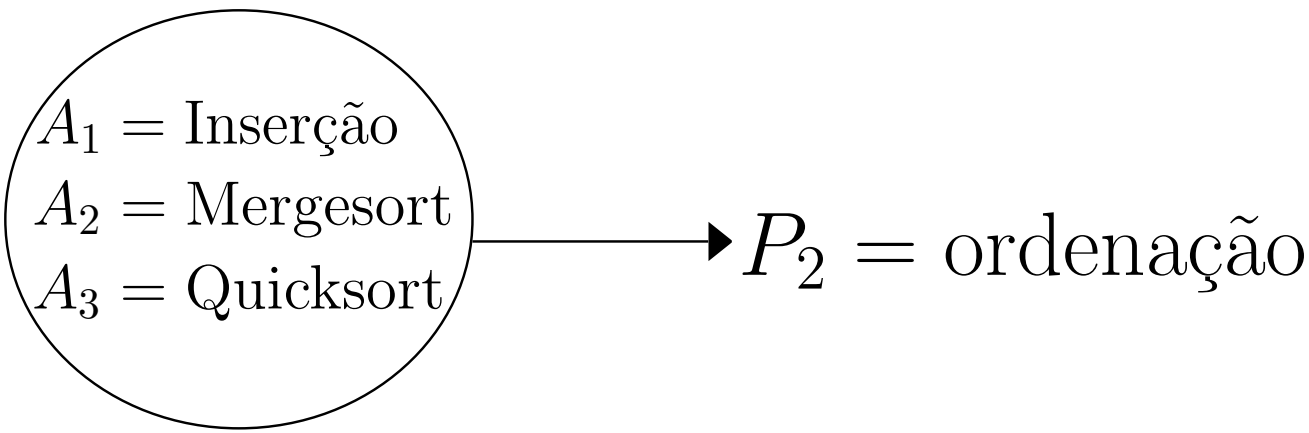
Introdução

- **Objetivo**: possibilitar medir eficiência de algoritmos;
- **Analisar um algoritmo**: prever os recursos que serão necessários;
- **Busca**: Sequencial ou Binária?



Introdução

- **Objetivo**: possibilitar medir eficiência de algoritmos;
- **Analisar um algoritmo**: prever os recursos que serão necessários;
- **Busca**: Sequencial ou Binária?
- **Ordenação**: Inserção, Seleção, Bolha, *Quicksort*, *Heapsort*, *Mergesort* ou *Shellsort*?



Conceitos Básicos: definições

- **modelo assumido**: instruções executadas uma após a outra, sem operações concorrentes (ou simultâneas);

Conceitos Básicos: definições

- **modelo assumido**: instruções executadas uma após a outra, sem operações concorrentes (ou simultâneas);
- **problema com entrada de tamanho** n: função de custo ou função de complexidade $f(n)$;

Conceitos Básicos: definições

- **modelo assumido**: instruções executadas uma após a outra, sem operações concorrentes (ou simultâneas);
- **problema com entrada de tamanho** n: função de custo ou função de complexidade $f(n)$;
- **função de complexidade de tempo**: número de execuções de determinada operação considerada relevante;

Conceitos Básicos: definições

- **modelo assumido**: instruções executadas uma após a outra, sem operações concorrentes (ou simultâneas);
- **problema com entrada de tamanho** n: função de custo ou função de complexidade $f(n)$;
- **função de complexidade de tempo**: número de execuções de determinada operação considerada relevante;
- **função de complexidade de espaço**: espaço de memória ocupada;

Conceitos Básicos: exemplo

Exemplo: calcular o fatorial de um número n.
Tamanho da entrada do problema: n;

```
     fatorial(n)
1:   fat ← 1;
2:   para (i ← 1; i ≤ n; i++) faça
3:        fat ← fat * i;
4:   retorna fat
```

Conceitos Básicos: fatorial

Exemplo: calcular o fatorial de um número n.

```
     fatorial(n)
1:   fat ← 1;
2:   para (i ← 1; i ≤ n; i++) faça    % executa n vezes
3:        fat ← fat * i;              % tempo constante c1
4:   retorna fat
```

Complexidade de tempo: $c_1.n$;

Complexidade de espaço: duas unidades de memória;

Conceitos Básicos: melhor caso, pior caso e caso médio

Três cenários dependentes da entrada:

- **Melhor caso**: menor tempo de execução;

Conceitos Básicos: melhor caso, pior caso e caso médio

Três cenários dependentes da entrada:

- **Melhor caso**: menor tempo de execução;
- **Pior caso**: maior tempo de execução. Geralmente, priorizamos determinar o pior caso;

Conceitos Básicos: melhor caso, pior caso e caso médio

Três cenários dependentes da entrada:

- **Melhor caso**: menor tempo de execução;
- **Pior caso**: maior tempo de execução. Geralmente, priorizamos determinar o pior caso;
- **Caso médio**: média dos tempos de execução. Mais difícil de obter;

Conceitos Básicos: melhor caso, pior caso e caso médio

Exemplo: busca sequencial.
Operação relevante: comparação de x com elementos de V;

```
     buscaSequencial(x, V)
1:   i ← 1;
2:   enquanto (i ≤ n) e (V[i] ≠ x) faça    // executa n vezes no máximo
3:        i ← i + 1;
4:   se i > n então retorna "Busca sem sucesso"
5:        senão retorna "Busca com sucesso"
```

Conceitos Básicos: melhor caso

Melhor caso da busca sequencial: x está em $V[1]$!

```
     buscaSequencial(x, V)
1:   i ← 1;
2:   enquanto (i ≤ n) e (V[i] ≠ x) faça      % executa n vezes no máximo
3:        i ← i + 1;
4:   se i > n então "Busca sem sucesso"
5:             senão "Busca com sucesso"
```

Complexidade de tempo: 1

Conceitos Básicos: pior caso

Pior caso da busca sequencial: x está em $V[n]$ ou não está em V!

```
     buscaSequencial(x, V)
1:   i ← 1;
2:   enquanto (i ≤ n) e (V[i] ≠ x) faça      % executa n vezes no máximo
3:        i ← i + 1;
4:   se i > n então "Busca sem sucesso"
5:        senão "Busca com sucesso"
```

Complexidade de tempo: n

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1$

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2$

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2 + 3$

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2 + 3 + \ldots + n)$

Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2 + 3 + \ldots + n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right)$

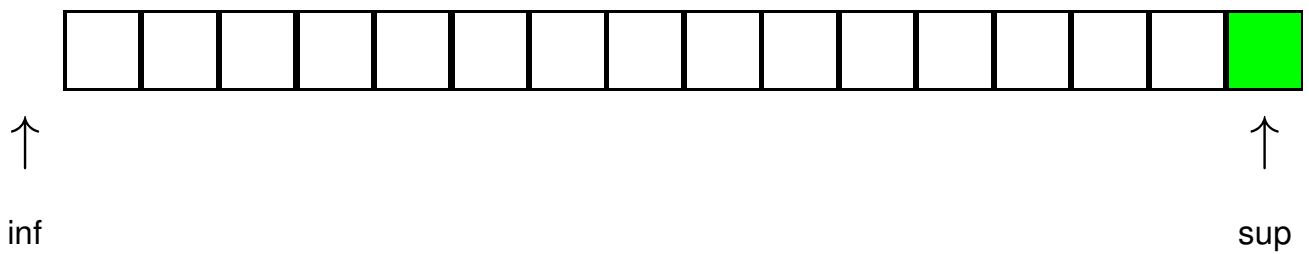
Conceitos Básicos: caso médio

- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2 + 3 + \ldots + n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$

Conceitos Básicos: caso médio

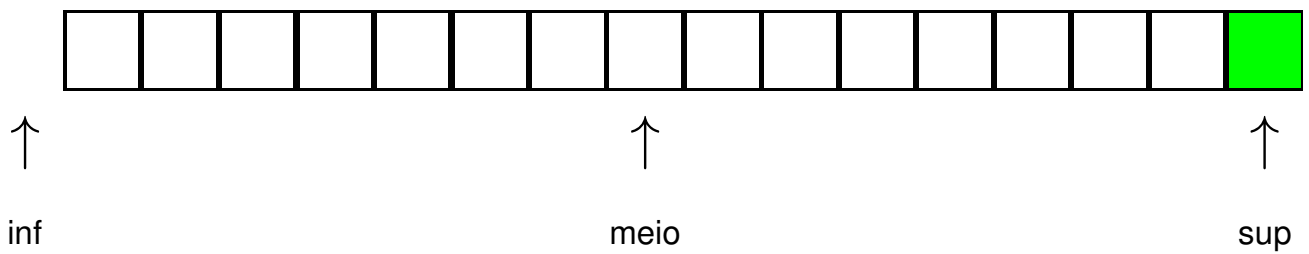
- **Caso médio da busca sequencial:** assumindo que
  - x está em V, e
  - $f(n) = 1 \times p_1 + 2 \times p_2 + \ldots + n \times p_n$ onde p_i é probabilidade de x estar na posição i;
  - **probabilidades são iguais**: $p_i = \frac{1}{n}$;
- $f(n) = \frac{1}{n}(1 + 2 + 3 + \ldots + n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$
- **Complexidade de tempo:** $\frac{n+1}{2}$

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



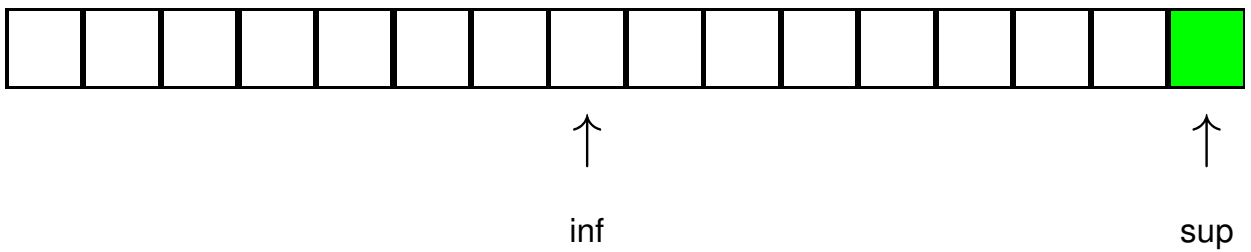
inicializ.

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



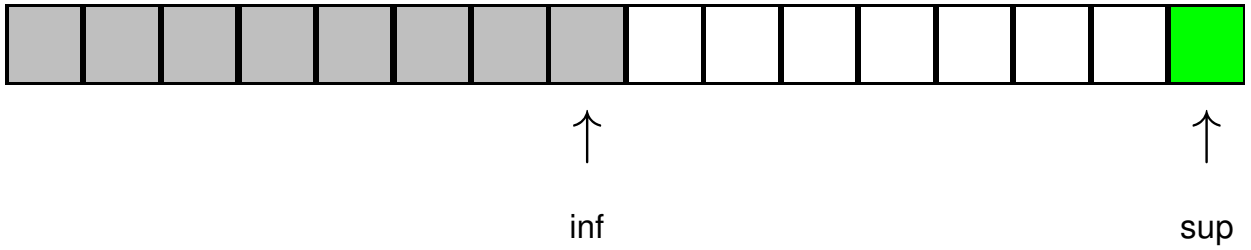
1a iter.

```
1: BuscaBinária(chave, lista[0 . . . n])
2: inf ← −1
3: sup ← n
4: enquanto inf < sup − 1 faça
5:     meio ← ⌊(inf+sup)/2⌋
6:     se chave ≤ lista[meio] então
7:         sup ← meio
8:     senão
9:         inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



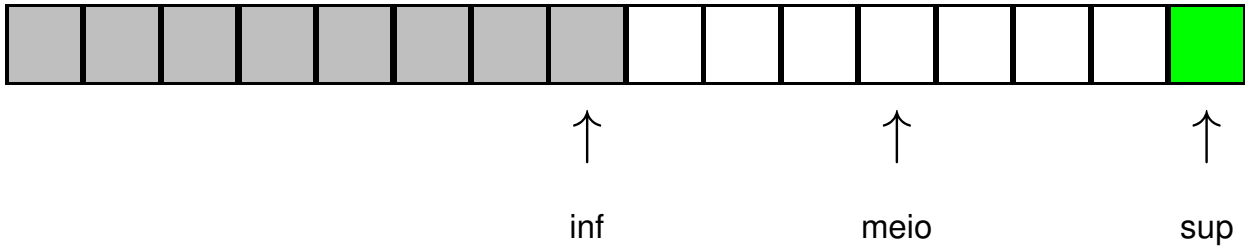
1a iter.

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



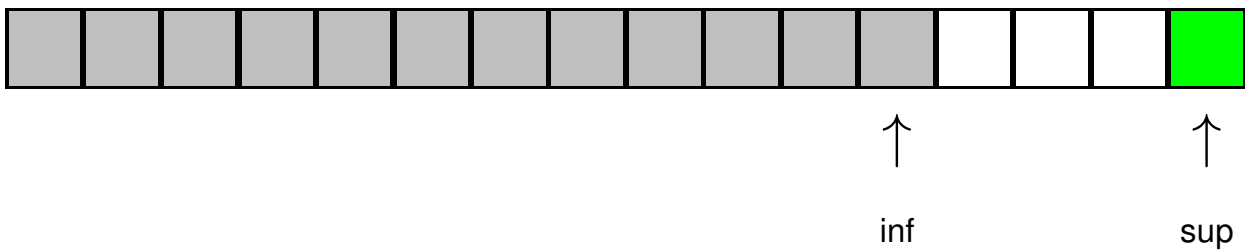
2a iter.

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



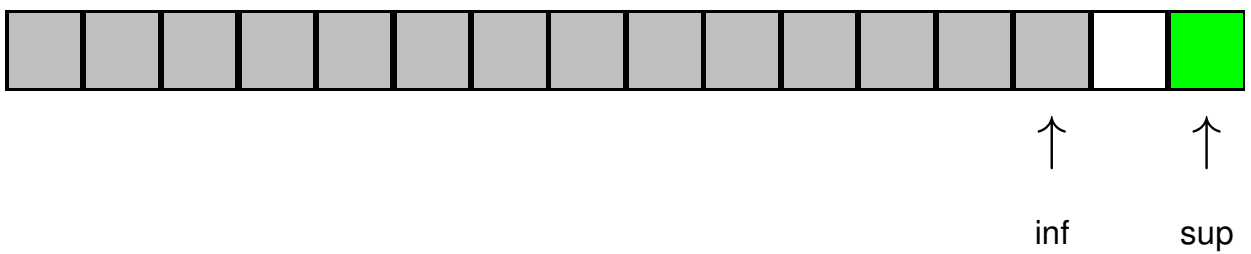
2a iter.

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



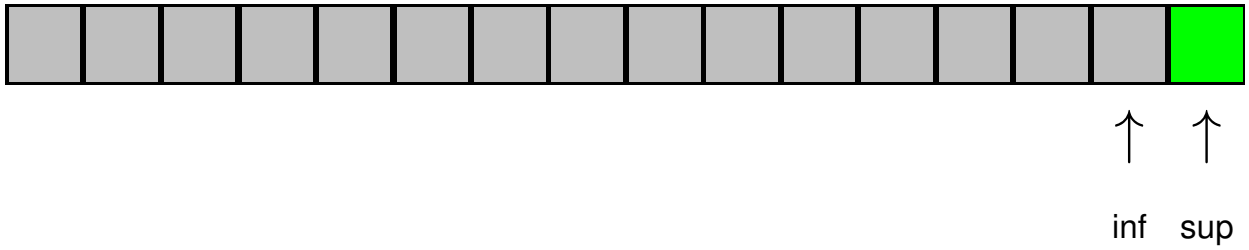
3a iter.

```
1:  BuscaBinária(chave, lista[0 . . . n])
2:  inf ← −1
3:  sup ← n
4:  enquanto inf < sup − 1 faça
5:      meio ← ⌊(inf+sup)/2⌋
6:      se chave ≤ lista[meio] então
7:          sup ← meio
8:      senão
9:          inf ← meio
10: se chave = lista[sup] então
11:     retorna lista[sup]
12: senão
13:     retorna elemento não encontrado
```



4a iter.

1: BuscaBinária($chave, lista[0 \ldots n]$)
2: $inf \leftarrow -1$
3: $sup \leftarrow n$
4: **enquanto** $inf < sup - 1$ **faça**
5: $\quad meio \leftarrow \lfloor \frac{inf+sup}{2} \rfloor$
6: $\quad$ **se** $chave \leq lista[meio]$ **então**
7: $\quad\quad sup \leftarrow meio$
8: $\quad$ **senão**
9: $\quad\quad inf \leftarrow meio$
10: **se** $chave = lista[sup]$ **então**
11: $\quad$ **retorna** $lista[sup]$
12: **senão**
13: $\quad$ **retorna** elemento não encontrado



5a iter.

Comparação: busca sequencial $\times$ busca binária

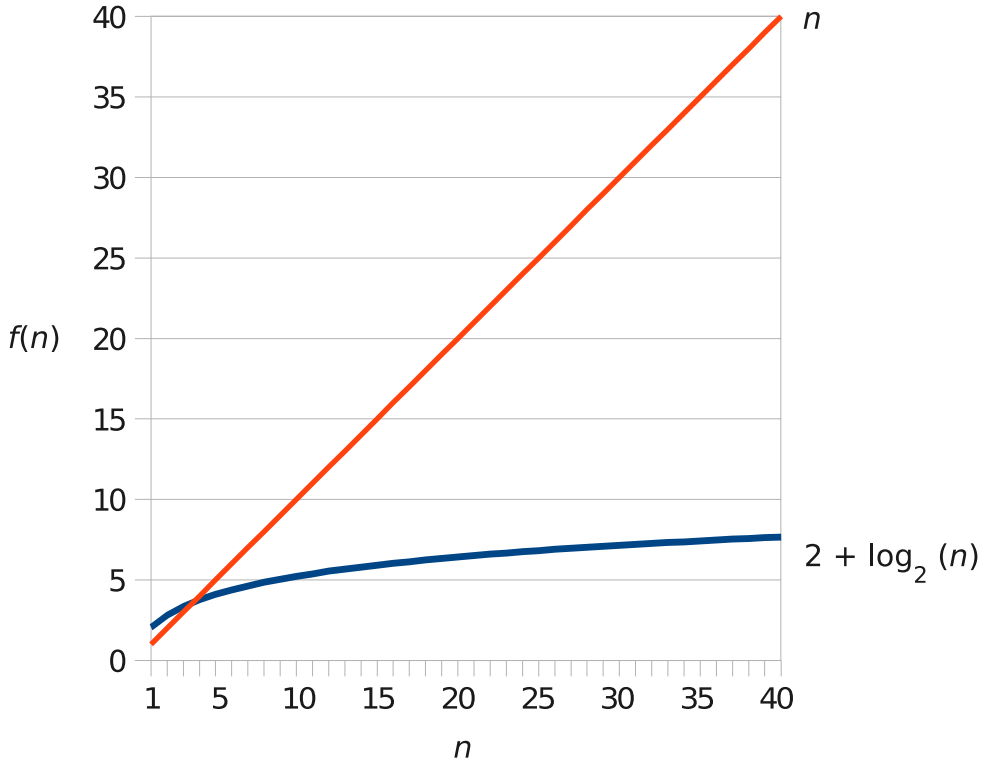
- **Problema de busca no pior caso**: busca binária leva $g(n) = 2 + \log_2 n$ passos (para n elementos) e busca sequencial leva $f(n) = n$ passos;

Comparação: busca sequencial $\times$ busca binária

- **Problema de busca no pior caso**: busca binária leva $g(n) = 2 + \log_2 n$ passos (para n elementos) e busca sequencial leva $f(n) = n$ passos;
- Vamos comparar $f(n) = n$ e $g(n) = 2 + \log_2 n$:

Comparação: busca sequencial $\times$ busca binária

- **Problema de busca no pior caso**: busca binária leva $g(n) = 2 + \log_2 n$ passos (para n elementos) e busca sequencial leva $f(n) = n$ passos;
- Vamos comparar $f(n) = n$ e $g(n) = 2 + \log_2 n$:

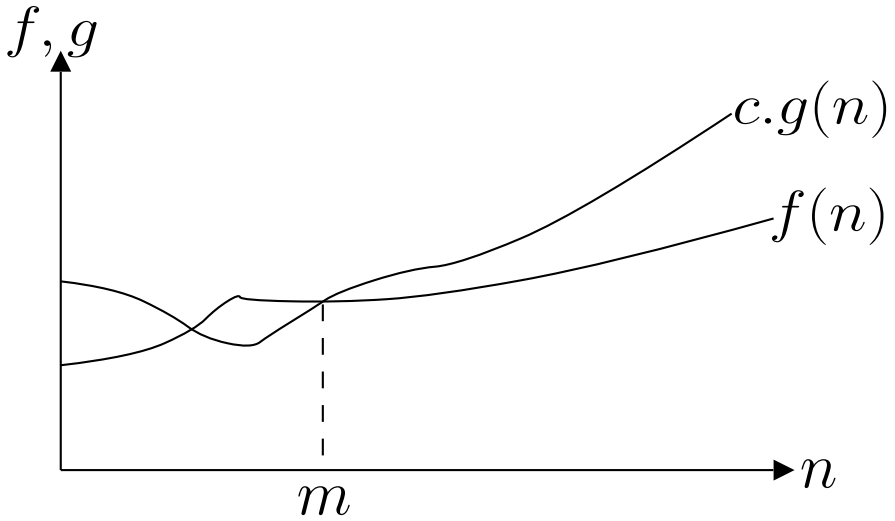


Notação Assintótica: Notação O

- **Análise assintótica dos programas:** mede como $f(n)$ se comporta conforme n aumenta indefinidamente;

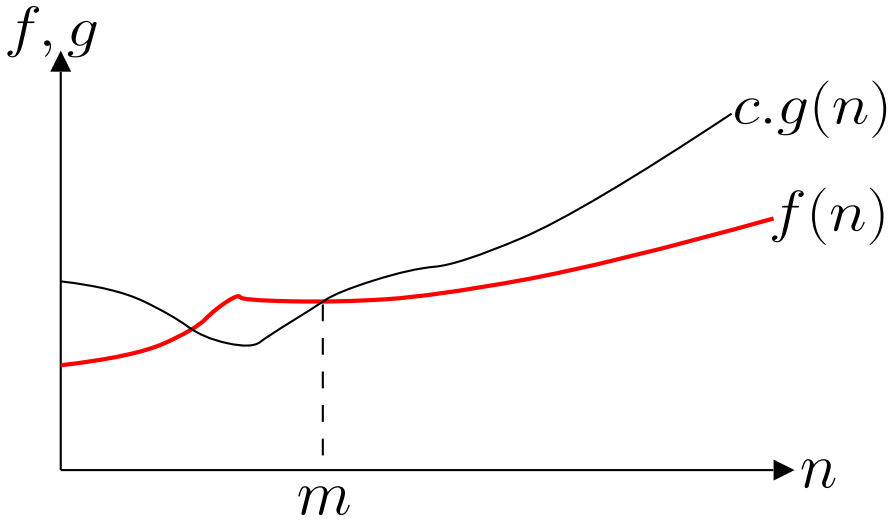
Notação Assintótica: Notação *O*

- **Análise assintótica dos programas:** mede como $f(n)$ se comporta conforme n aumenta indefinidamente;
- Uma função $f(n)$ é $O(g(n))$ (notação $f(n) = O(g(n))$) se existem duas constantes positivas c e m tais que $f(n) \leq c.g(n)$, para todo $n \geq m$;



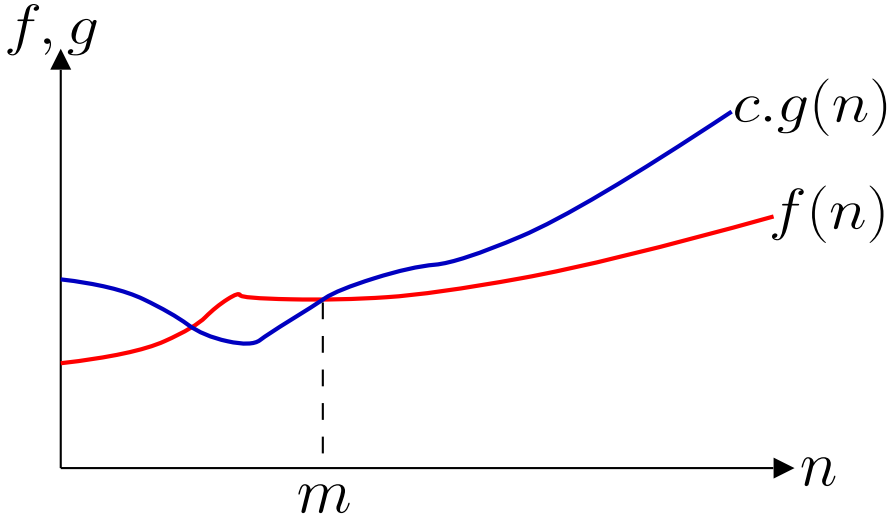
Notação Assintótica: Notação O

- **Análise assintótica dos programas:** mede como $f(n)$ se comporta conforme n aumenta indefinidamente;
- Uma função $f(n)$ é $O(g(n))$ (notação $f(n) = O(g(n))$) se existem duas constantes positivas c e m tais que $f(n) \leq c.g(n)$, para todo $n \geq m$;
- $f(n) = O(g(n))$: $g(n)$ é limite superior para $f(n)$;



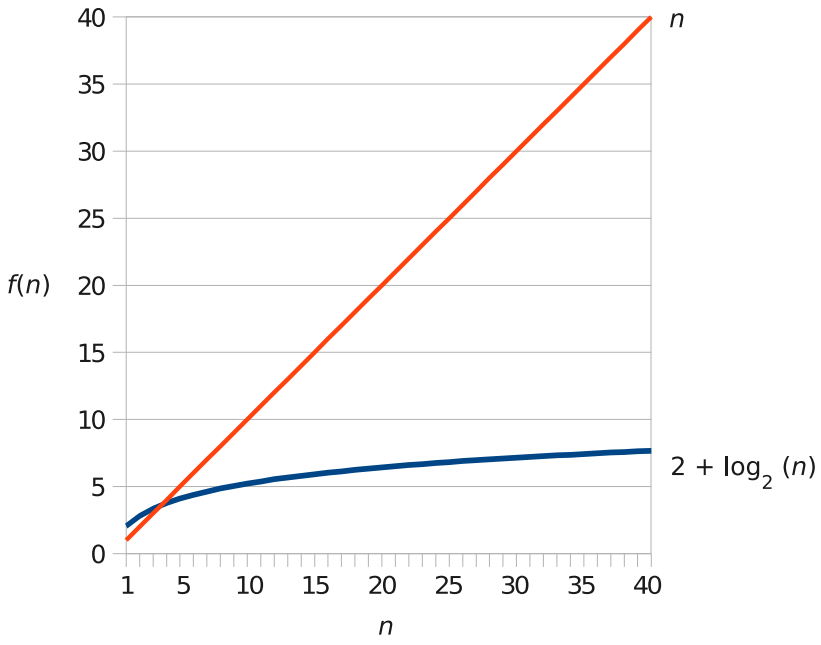
Notação Assintótica: Notação O

- **Análise assintótica dos programas:** mede como $f(n)$ se comporta conforme n aumenta indefinidamente;
- Uma função $f(n)$ é $O(g(n))$ (notação $f(n) = O(g(n))$) se existem duas constantes positivas c e m tais que $f(n) \leq c.g(n)$, para todo $n \geq m$;
- $f(n) = O(g(n))$: $g(n)$ é limite superior para $f(n)$;



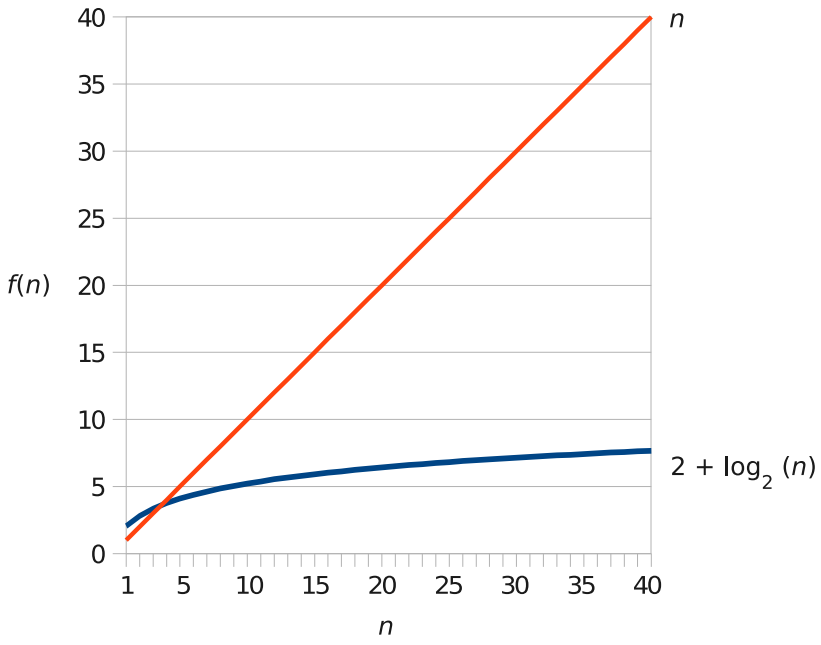
Notação Assintótica: Notação *O*

- Qual a complexidade assintótica da busca sequencial e da busca binária?



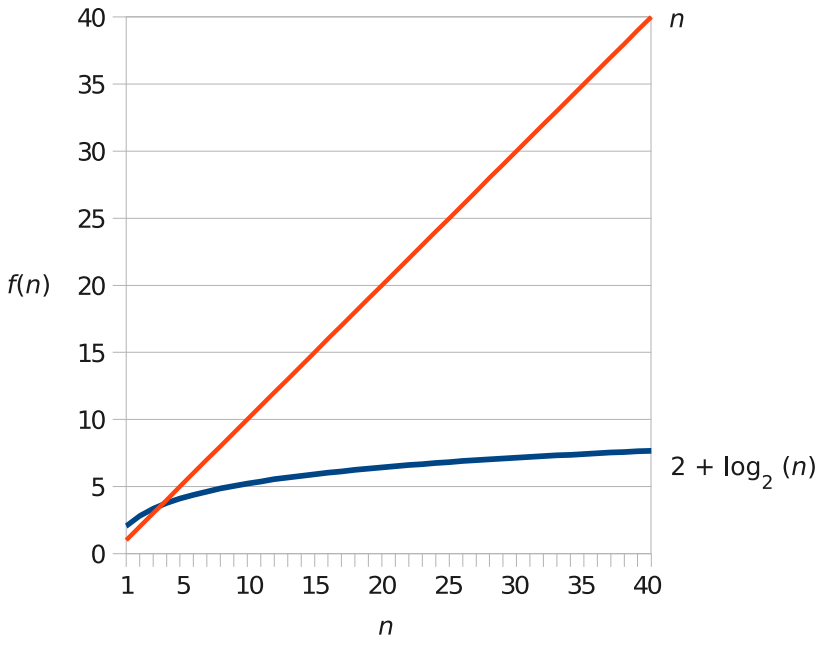
Notação Assintótica: Notação *O*

- Qual a complexidade assintótica da busca sequencial e da busca binária?
- Busca sequencial: $n = O(n)$, para $c \geq 1$, $m = 1$;



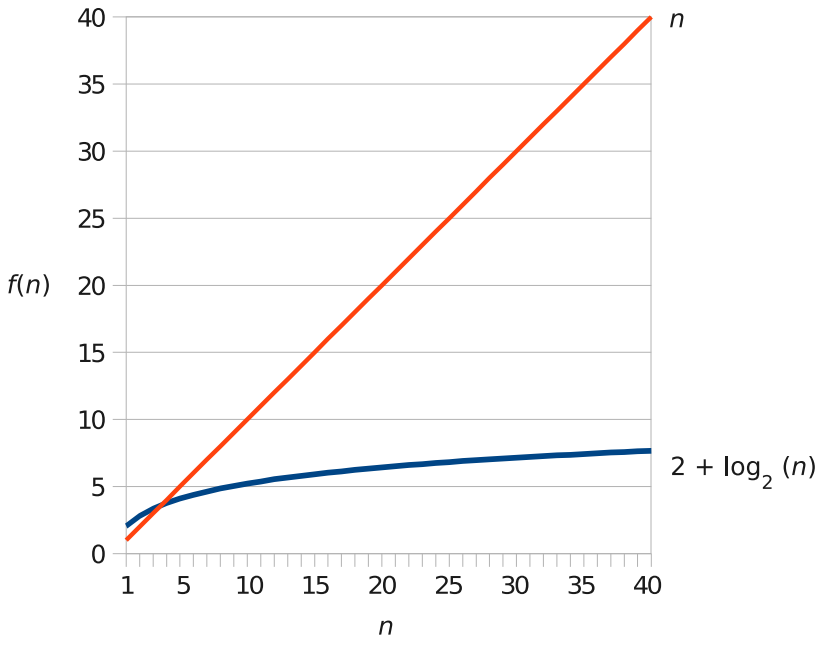
Notação Assintótica: Notação O

- Qual a complexidade assintótica da busca sequencial e da busca binária?
- Busca sequencial: $n = O(n)$, para $c \geq 1$, $m = 1$;
- Busca binária: $2 + \log_2 n = O(\log_2 n)$, para $c \geq 4$, $m = 2$;



Notação Assintótica: Notação O

- Qual a complexidade assintótica da busca sequencial e da busca binária?
- Busca sequencial: $n = O(n)$, para $c \geq 1$, $m = 1$;
- Busca binária: $2 + \log_2 n = O(\log_2 n)$, para $c \geq 4$, $m = 2$;
- Como $2 + \log_2 n = O(n)$, $c \geq 1$, $m = 4$, a busca binária é assintoticamente mais eficiente que a busca sequencial!



Notação Assintótica: Notação O

Vamos analisar o trecho de código abaixo:

```
1:  sum1 ← 0;
2:  para (i = 1; i ≤ n; i++) faça
3:      para (j = 1; j ≤ n; j++) faça
4:          sum1 ← sum1 + 1;
```

Notação Assintótica: Notação O

Vamos analisar o trecho de código abaixo:

```
1:  sum1 ← 0;
2:  para (i = 1; i ≤ n; i++) faça          % n vezes
3:      para (j = 1; j ≤ n; j++) faça
4:          sum1 ← sum1 + 1;
```

Notação Assintótica: Notação O

Vamos analisar o trecho de código abaixo:

```
1:  sum1 ← 0;
2:  para (i = 1; i ≤ n; i++) faça          % n vezes
3:      para (j = 1; j ≤ n; j++) faça      % n vezes
4:          sum1 ← sum1 + 1;
```

Notação Assintótica: Notação O

Vamos analisar o trecho de código abaixo:

```
1:  sum1 ← 0;
2:  para (i = 1; i ≤ n; i++) faça          % n vezes
3:      para (j = 1; j ≤ n; j++) faça      % n vezes
4:          sum1 ← sum1 + 1;               % n² vezes
```

Complexidade assintótica: $O(n^2)$!

Notação Assintótica: Notação O

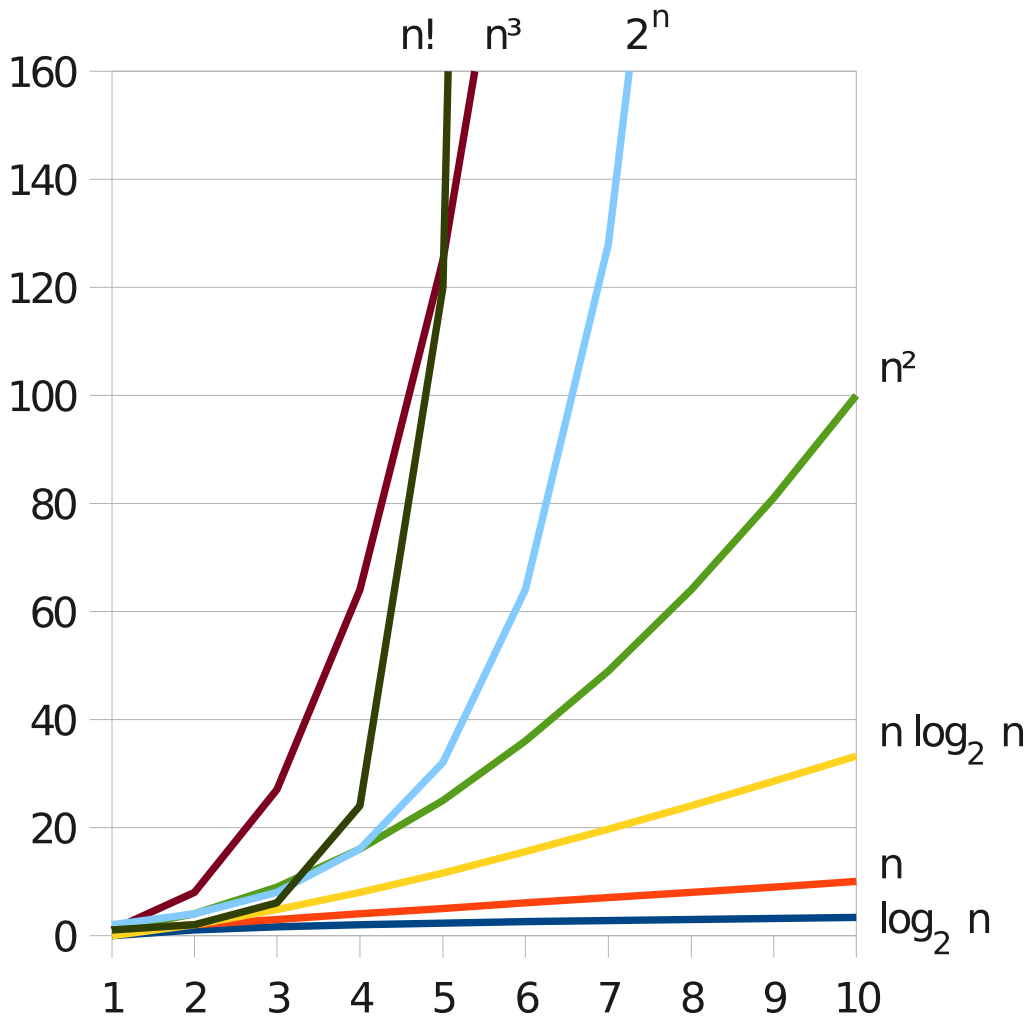
Outro trecho de código:

1: $sum2 \leftarrow 0$;
2: **para** $(i = 1; i \leq n; i++)$ **faça**
3: **para** $(j = 1; j \leq i; j++)$ **faça**
4: $sum2 \leftarrow sum2 + 1$;

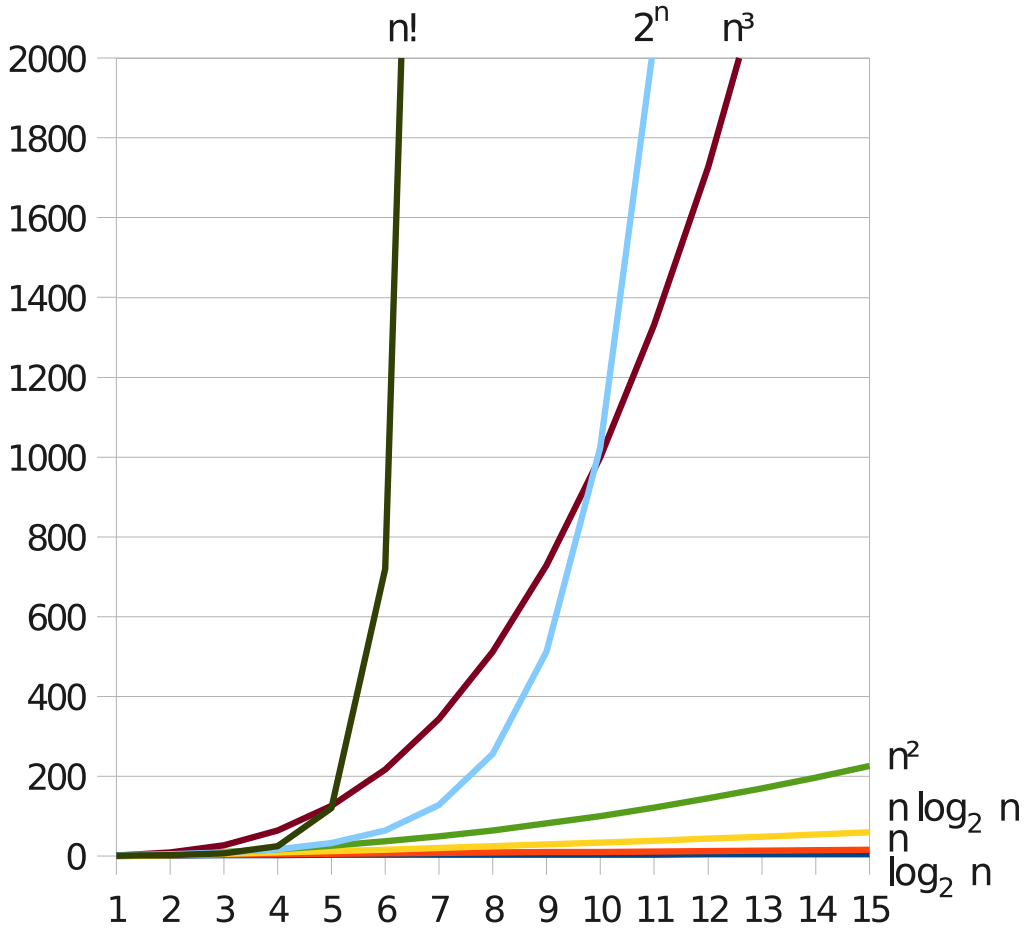
Complexidade assintótica:

$$1+2+3+\ldots+n = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = O(n^2), c \geq 1, m = 1.$$

Funções de complexidade frequentes



Funções de complexidade frequentes



Bibliografia

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. Algoritmos: teoria e prática. 2^a edição, Campus, 2002.

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

ZIVIANI, N. Projeto de Algoritmos com Implementações em Java e C++. Thomson, 2007.