Análise de Algoritmos
Complexidade

Introdução

- Algoritmos de ordenação
  - Seleção (Selection Sort)
  - Inserção (Insertion Sort)
  - Bolha (Bubble Sort)

Introdução

```
void insercaoDireta(int [] numeros)
 {

   for (int ivet=1; ivet < numeros.length; ivet++)
   {
      int numaInserir = numeros[ivet];
      int isubv = ivet;

      while ((isubv > 0) &&
             (numeros [isubv -1] > numoAInserir))
      {
        numeros[isubv] = numeros[isubv - 1];
        isubv--;
      }
      numeros[isubv] = numAInserir;
    }
 }
```

```
void bolha(int [] numeros)
 {

   for (ivet = numeros.length - 1; ivet > 0; ivet--)
   {
          for (isubv = 0; isubv < ivet; isubv++)

       if (numeros[isubv ] > numeros[isubv+1])
       {
          temp = numeros[isubv];
          numeros [isubv] = numeros [isubv+1];
          numeros [isubv+1] = temp;
       }
    }
 }
```

Análise de Algoritmos

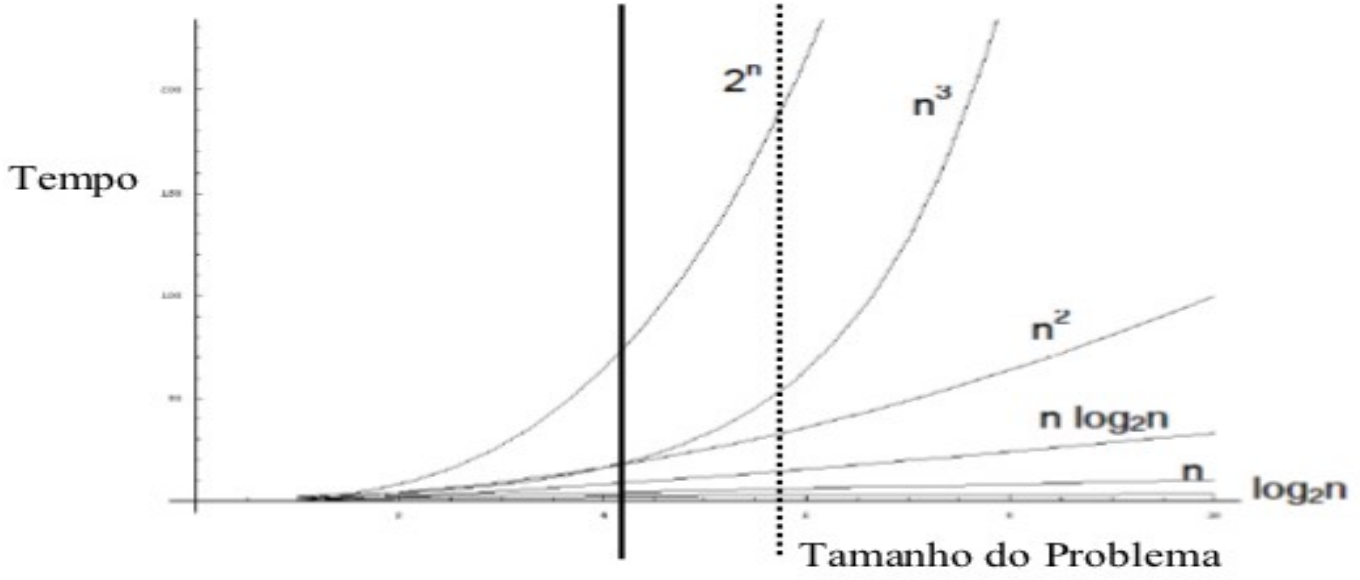
- Permite escolher o algoritmo mais **eficiente** dentre um conjunto de candidatos para resolver um problema
  - memória, largura de banda de comunicação, hardware
  - principal: **tempo de computação**

Análise de Algoritmos

- É usual descrever o tempo de execução de um programa como uma função do tamanho de sua entrada (**n**)
  - E na maioria das vezes analisa-se o algoritmo quando o valor de **n** tente a infinito
  - **Análise Assintótica**

Análise Assintótica

- Classes de complexidade



Análise Assintótica

- Classes de Complexidade

	10	100	10^3	10^4	10^5	10^6
$\log_2 n$	3	6	9	13	16	19
n	10	100	1000	10^4	10^5	10^6
$n \log_2 n$	30	664	9965	10^5	10^6	10^7
n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{300}	10^{300}	10^{3000}	10^{300000}

1 ano = $365 \times 24 \times 60 \times 60 \approx 3 \times 10^7$ segundos

1 século $\approx 3 \times 10^9$ segundos

1 milénio $\approx 3 \times 10^{10}$ segundos

Complexidade

- Usa-se notações especiais para representar a complexidade assintótica

  Notação O

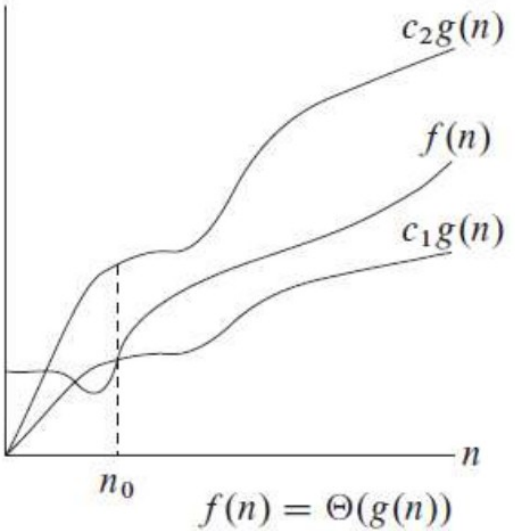
  Notação Ω

  Notação Θ

Notação Θ

- Limite assintótico forte

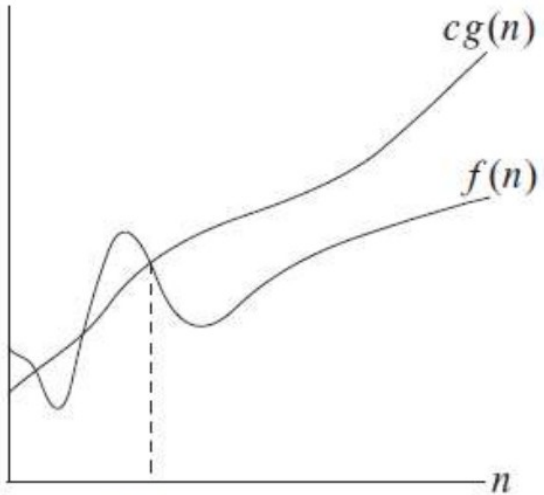
$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}.^1$$



Notação O

- Limite assintótico superior

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}.$$



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

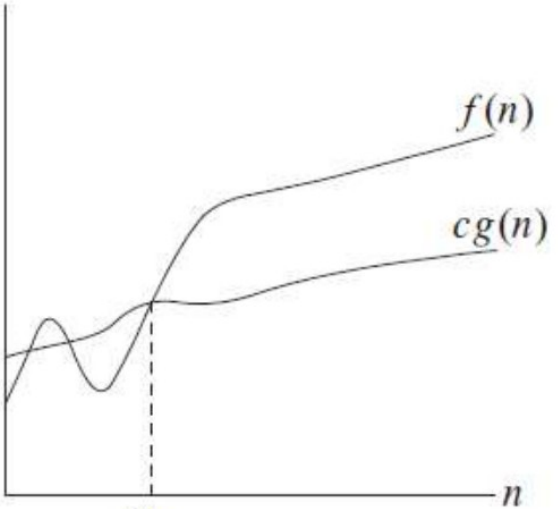
Notação O

- Como é um limite superior, é muito usado para análise de **pior caso**.

Notação Ω

- Limite assintótico inferior

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$



$f(n) = \Omega(g(n))$

Notação Ω

- Como é um limite inferior, é muito usado para análise de melhor **caso**.

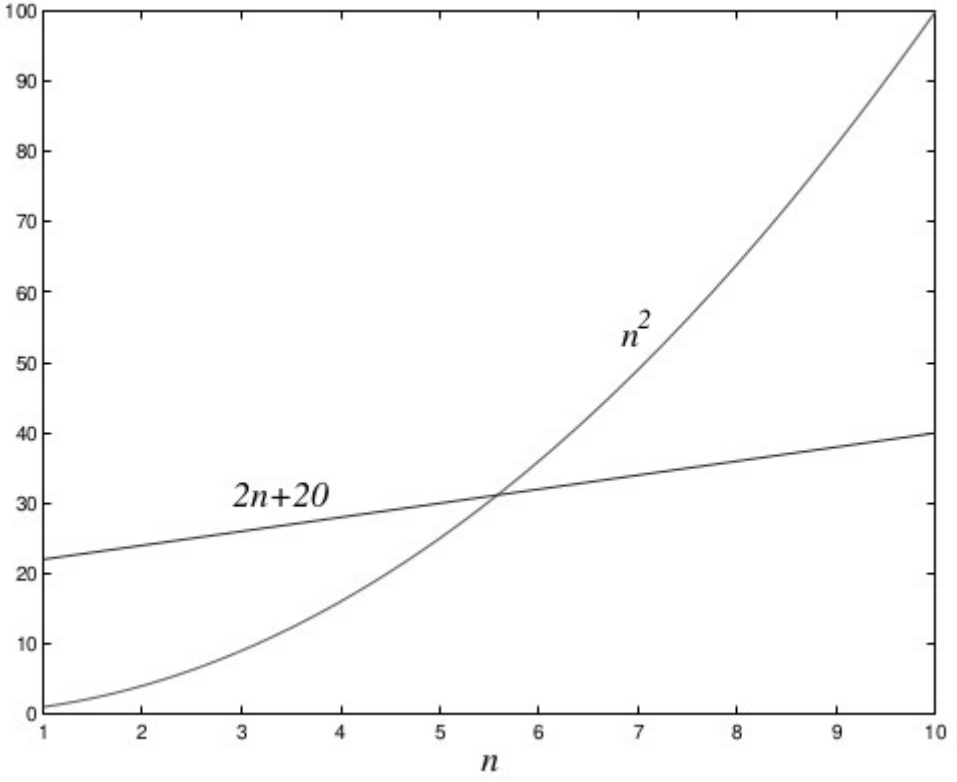
Análise Assintótica

- Exemplo:

$f_1(n) = n^2$
$f_2(n) = 2n + 20$

$f_2 = O(f_1)$:

$$\frac{f_2(n)}{f_1(n)} = \frac{2n+20}{n^2} \leq 22$$



Análise Assintótica

- Notação Ω
  - Ω define um limite inferior para a função, por um fator constante.
  - g(n) = Ω(f(n))
  - Formalmente:
    - g(n) = Ω(f(n)), c > 0 e n0 | 0 <= c.f(n) <= g(n), ∀ n >= n0

Análise Assintótica

- Notação Θ
  - A notação Θ limita a função por fatores constantes
  - g(n) = Θ(f(n)) se existirem constantes positivas c1 e c2 e n0 tais que para n >= n0, o valor de g(n) está sempre entre c1.f(n) e c2.f(n) inclusive.
  - Formalmente:

$$g(n) = \Theta(f(n)),\ c1 > 0 \text{ e } c2 > 0 \text{ e } n0 \mid$$

$$0 <= c1.f(n) <= g(n) <= c2.f(n),\ \forall\ n >= n0$$

Análise de Complexidade

```
BUSCA-BINÁRIA (V[], início, fim, e)
    i recebe o índice do meio entre início e fim
    se (v[i] = e) entao
        devolva o índice i   # elemento e encontrado
    fimse
    se (inicio = fim) entao
        não encontrou o elemento procurado
    senão
       se (V[i] vem antes de e) então
          faça a BUSCA-BINÁRIA(V, i+1, fim, e)
       senão
          faça a BUSCA-BINÁRIA(V, inicio, i-1, e)
       fimse
    fimse
```

Exercícios 1

- A qual classe de complexidade pertence cada uma das seguintes funções?

	$f(n)$
(a)	$n - 100$
(b)	$n^{1/2}$
(c)	$100n + \log n$
(d)	$n \log n$
(e)	$\log 2n$
(f)	$10 \log n$
(g)	$n^{1.01}$
(h)	$n^2 / \log n$
(i)	$n^{0.1}$
(j)	$(\log n)^{\log n}$
(k)	$\sqrt{n}$
(l)	$n^{1/2}$
(m)	$n2^n$

Exercícios 2

- Indique se f(n) = O(g(n)), ou se f(n) = Ω(g(n)), ou se f(n) = Θ(g(n))

	$f(n)$	$g(n)$
(a)	$n - 100$	$n - 200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n + \log n$	$n + (\log n)^2$
(d)	$n \log n$	$10n \log 10n$
(e)	$\log 2n$	$\log 3n$
(f)	$10 \log n$	$\log(n^2)$
(g)	$n^{1.01}$	$n \log^2 n$
(h)	$n^2 / \log n$	$n(\log n)^2$
(i)	$n^{0.1}$	$(\log n)^{10}$
(j)	$(\log n)^{\log n}$	$n / \log n$
(k)	$\sqrt{n}$	$(\log n)^3$
(l)	$n^{1/2}$	$5^{\log_2 n}$
(m)	$n2^n$	3^n

$$f = \Omega(g) \text{ means } g = O(f)$$
$$f = \Theta(g) \text{ means } f = O(g) \text{ and } f = \Omega(g).$$

Exercícios 3

- Faça a análise de pior e melhor caso dos seguintes algoritmos:

```
void insercaoDireta(int [] numeros)
 {

  for (int ivet=1; ivet < numeros.length; ivet++)
  {
     int numaInserir = numeros[ivet];
     int isubv = ivet;

     while ((isubv > 0) &&
            (numeros [isubv -1] > numoAInserir))
     {
      numeros[isubv] = numeros[isubv - 1];
      isubv--;
     }
     numeros[isubv] = numAInserir;
    }
 }
```

```
void bolha(int [] numeros)
 {

  for (ivet = numeros.length - 1; ivet > 0; ivet--)
  {
          for (isubv = 0; isubv < ivet; isubv++)

      if (numeros[isubv ] > numeros[isubv+1])
      {
         temp = numeros[isubv];
         numeros [isubv] = numeros [isubv+1];
         numeros [isubv+1] = temp;
      }
   }
}
```