Algoritmos e Estruturas de Dados I

Profª Priscilla Abreu

2022.1

Algoritmos e Estruturas de Dados I

Roteiro da aula

- Correção de exercício
- Complexidade de algoritmos

Algoritmos e Estruturas de Dados I

Disponibilização de materiais e avisos:

- Google Classroom:

vurjz3i

https://classroom.google.com/c/NDg4NDE4MDk5NTA4?cjc=vurjz3i

Complexidade de algoritmos

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos: como avaliar?

- Espaço

  Quantidade de recursos (memória) utilizados.

- Tempo

  Tempo de execução do algoritmo ou o total de instruções executadas.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos: como medir?

- Empiricamente

- Analiticamente

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos: como medir?

- Representaremos o tempo de execução de um algoritmo por uma função de custo T, considerando T(n) como a medida do tempo necessário para executar um algoritmo de tamanho *n*. T é denominada de **função de complexidade de tempo** do algoritmo.
- Se T representa a memória necessária para a execução de um algoritmo então T(n) é denominada **função de complexidade de espaço** do algoritmo.

Algoritmos e Estruturas de Dados I

Exemplo linear

```
funçao encontraMenor( vetor[n]: inteiro, n: inteiro):inteiro
inicio
	i, menor: inteiro
	menor = vetor[0]
	para i:=1 até n-1 faça
		se (vetor[i] <  menor) então
			menor := vetor[i]
	fim-para
	retorne menor
fim
```

Função de complexidade T(n):
número de comparações entre os elementos de vetor.

Serão realizadas n-1 comparações.
T(n) = n-1

Tempo de execução **uniforme**, dependente apenas do tamanho de entrada.

Algoritmos e Estruturas de Dados I

Exemplo linear

Considere a busca por um determinado valor em um vetor.

1	9	4	26	18	55		30
0	1	2	3	4	5	...	n

```
funcao busca(vetor[n]: inteiro, n:inteiro, valor: inteiro): inteiro
inicio
      i: inteiro
      para i:=0 até (n-1) faça
            se (vetor[i] = valor) então
                  retorne i
            fim-se
      fim-para
      retorne -1
fim
```

Algoritmos e Estruturas de Dados I

Exemplo linear: Considere a busca por um determinado valor em um vetor.

Esse algoritmo não se comporta de maneira uniforme. Temos três casos:

- Pior caso

  Maior tempo de execução sobre todas as entradas de tamanho n.

- Melhor caso

  Menor tempo de execução sobre todas as entradas de tamanho n.

- Caso médio

  Média dos tempos de execução do algoritmo sobre todas as entradas de tamanho n.

Algoritmos e Estruturas de Dados I

Exemplo Quadrático

Dadas duas matrizes $A = (a_{ij})$ e $B = (b_{ij})$, ambas nxn, determinar a matriz soma $C = (c_{ij})$.

```
funcao somaMatriz(m1: matriz, m2:matriz): matriz
inicio
    mSoma: matriz
    lin, col: inteiro
    para lin:=0 até (n-1) faça
        para col:=0 até (n-1) faça
            mSoma[lin][col] = m1[lin][col] + m2[lin][col]
        fim-para
    fim-para
    retorne -1
fim
```

Algoritmos e Estruturas de Dados I

Exemplo Quadrático

Dadas duas matrizes $A = (a_{ij})$ e $B = (b_{ij})$, ambas nxn, determinar a matriz soma $C = (c_{ij})$.

```
funcao somaMatriz(m1: matriz, m2:matriz): matriz
inicio
	mSoma: matriz
	lin, col: inteiro
	para lin:=0 até (n-1) faça
		para col:=0 até (n-1) faça
			mSoma[lin][col] = m1[lin][col] + m2[lin][col]
		fim-para
	fim-para
	retorne -1
fim
```

Tamanho da entrada: n^2
Tempo = total de passos = total de instruções executadas: n^2

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

- Ordem de grandeza
- Operação dominante
- Pior caso
- Descarte de constantes aditivas e/ou multiplicativas
- Comportamento assintótico = entradas com tamanho suficientemente grande.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

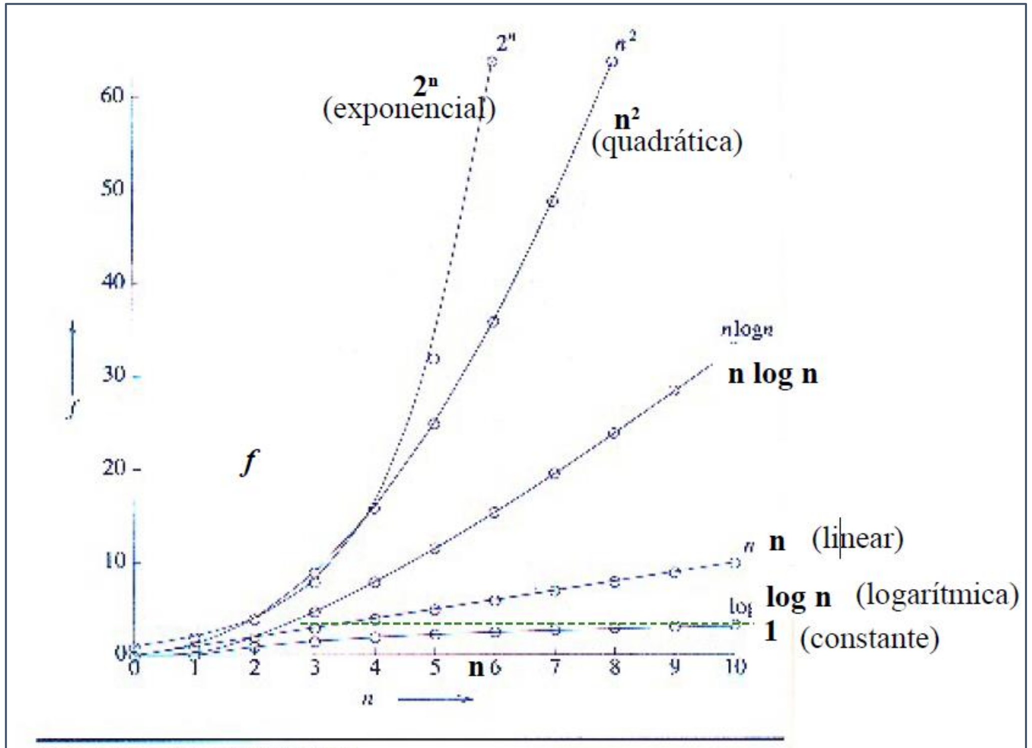
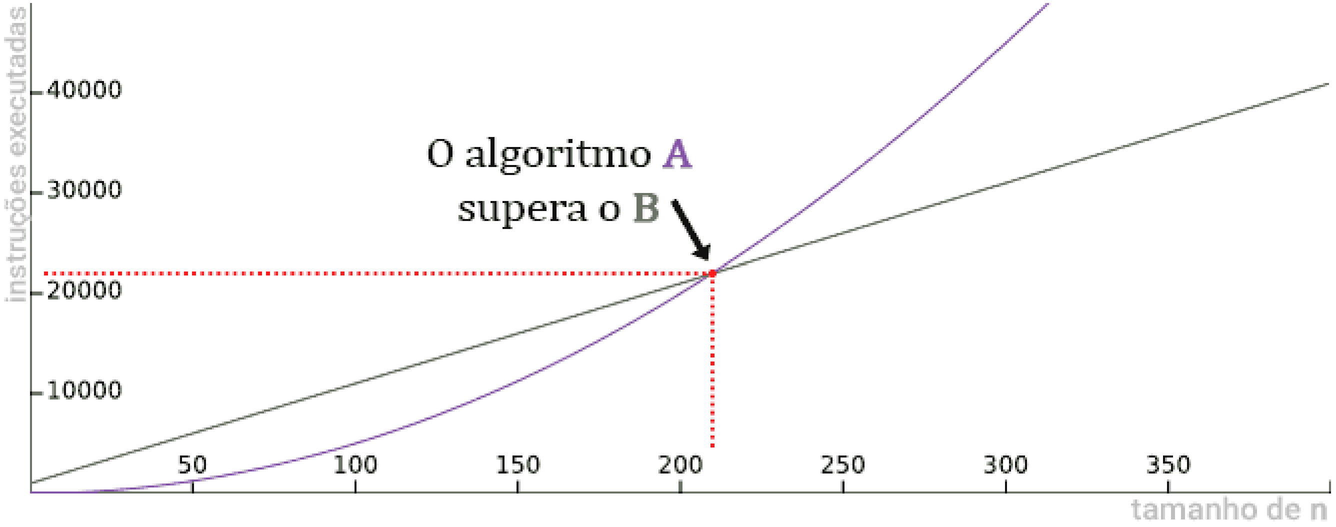


Figure 2.24 Plot of various functions

Fonte: Data Structures,Algorithms and Applications in C++, Sahni

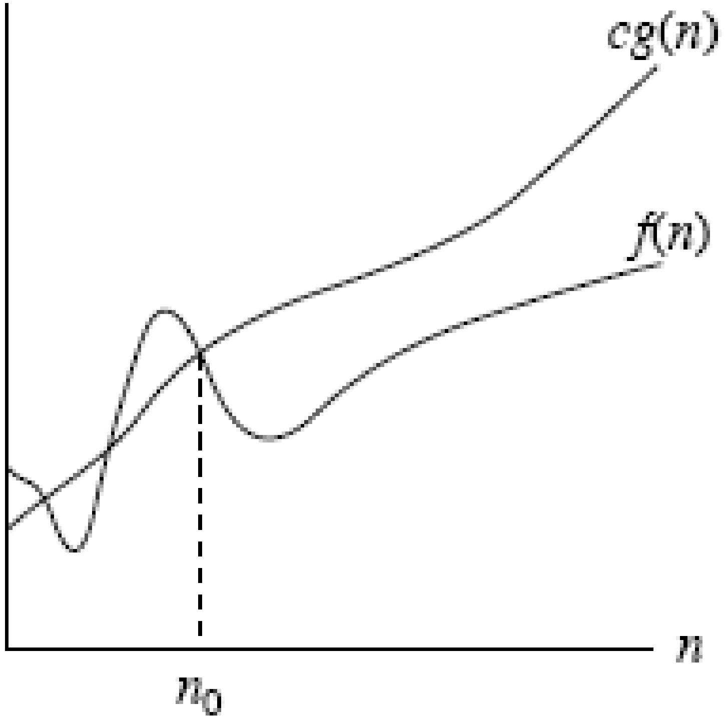


Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos – notação assintótica

Definição: uma função g(n) domina assintoticamente outra função f(n) se existem duas constantes positivas c e n_0 tais que, para n >= n_0 temos que:

$$|f(n)| \leq c \times |g(n)|$$



Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos – notação assintótica

- Alguns tipos:
  - Notação O
  - Notação Ω
  - Notação θ

Algoritmos e Estruturas de Dados I

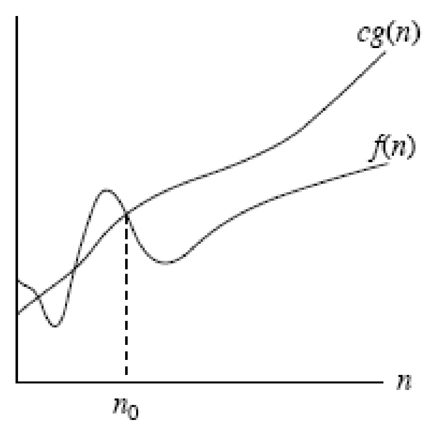
Complexidade de Algoritmos – notação assintótica

- Notação O

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

se existirem uma constante c e um valor n_0 tal que

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



A notação O se refere ao **limite superior** de uma função. Ou seja, existe uma função à qual o algoritmo analisado não terá um ponto acima da curva desta função.

Isto é, a função f(n) cresce no máximo como a função g(n).

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

<u>Exercício</u>

Construir um algoritmo para dada uma matriz nxn determinar o maior elemento da matriz. Calcular as complexidades de melhor e pior caso do seu algoritmo.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

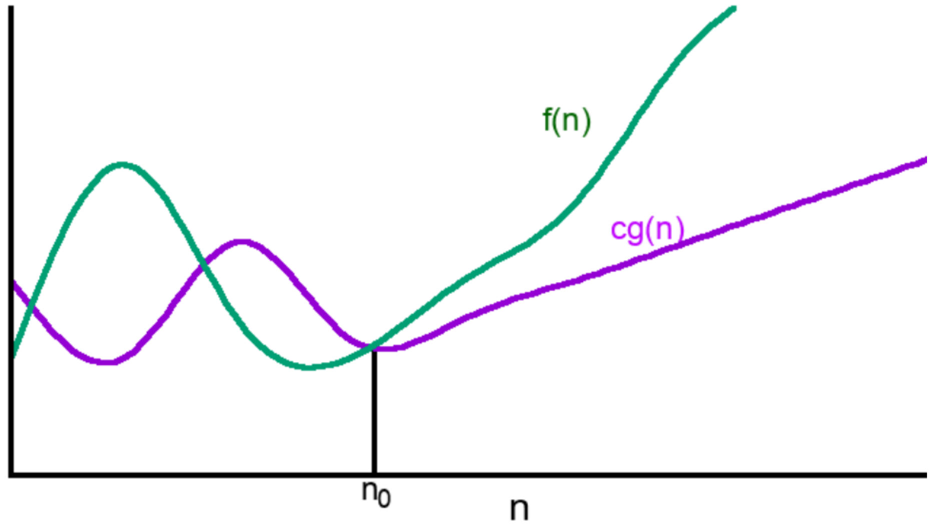
- Notação Ω

Diferente da notação O, aqui estamos focando em uma função que é o limite inferior do algoritmo estudado.

$$f(n) = \Omega(g(n)), \quad \text{se existirem uma constante}$$

$$c \text{ e um valor } n_0 \text{ tal que}$$

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



Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

- Notação Ω

Seja P um problema. Um limite inferior para P é uma função f(n), tal que a complexidade de pior caso de qualquer algoritmo que resolva P é Ω(f(n)).

Isto é: todo algoritmo que resolve o problema P executará, pelo menos, Ω(f(n)) passos!

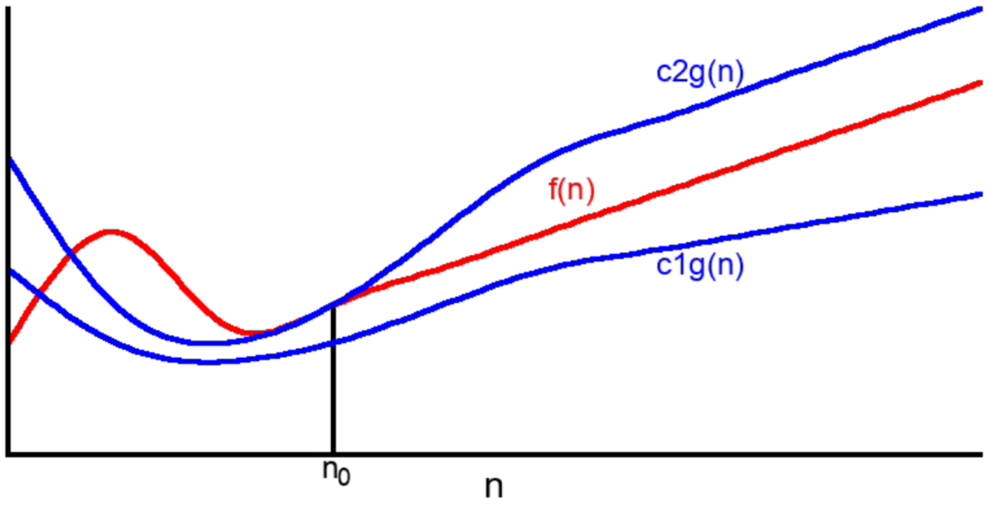
Se existir um algoritmo A para o problema P cuja complexidade de **pior caso é O(f(n))**, então A é denominado **algoritmo ótimo** para P.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

- Notação Θ

Descreve o comportamento assintótico de funções definindo ao mesmo tempo um limite superior e um limite inferior para a execução de um determinado algoritmo.



Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

- Notação Θ

Ao invés de utilizarmos uma constante c1, agora utilizaremos duas constantes: c1 e c2. Essas constantes mostram como apenas deslocando a curva no gráfico conseguimos ver que **uma função** é ao mesmo tempo os **limites superiores** e **inferiores** da função estudada.

$f(n) = \Theta(g(n))$, se existirem constantes c_1 e c_2 e um valor n_0 tal que

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

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Exemplos notação assintótica

Termo de maior crescimento	Notação	Funções T(N) que satisfazem notação	Funções T(N) que NÃO satisfazem notação
Igual a N^2	$\theta(N^2)$	$\frac{1}{2} N^2 + 10N$; $100N^2$	$10N$; $N^3 + 2N^2$; $N \log N$
No máximo N^2	$O(N^2)$	$100N^2$; 100; $N \log N$	$N^3 + 2N^2$; $N^2 \lg N$
No mínimo N^2	$\Omega(N^2)$	$100N^2$; $N^3 + 2$; $N^2 \lg N$	$10N$; 100; $N \log N$

Como chegar à complexidade de um algoritmo?

Complexidade de Algoritmos

1. Separe o programa em blocos.

   Um bloco é um trecho do código que atende a alguma das definições abaixo:

   - Um comando;
   - Um comando condicional incluindo os blocos correspondentes aos trechos *então*, *senão se*, e *senão ;*
   - Uma sequência de blocos;
   - Uma repetição, incluindo o bloco de seu corpo;

Complexidade de Algoritmos

```
funçao encontraMenor( vetor[n]: inteiro, n: inteiro):inteiro
inicio
        i, menor: inteiro
        menor = vetor[0]
        para i:=1 até n-1 faça
                se (vetor[i] <  menor) então
                        menor := vetor[i]
        fim-para
        retorne menor
fim
```

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

2. Determinação das complexidades dos blocos seguindo um estrutura "bottom-up":
    - A complexidade de um dado bloco B será dada de acordo com as complexidades dos blocos que ele contém.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Um comando

Comando	Complexidade de Tempo
`x ← x + 1`	$\Theta(1)$
`x ← x + y`	$\Theta(1)$
`A[i] ← 1`	$\Theta(1)$
`A[1..N] ← 1`	$\Theta(N)$
`A[i] ← máx{ A[j] | 1 ≤ j ≤ N }`	$\Theta(N)$
`A[i,j] ← 0,` `para todo 1 ≤ i,j ≤ N`	$\Theta(N^2)$
`m ← Calcular(A,N)`	complexidade de tempo de `Calcular(A,N)`

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Condicional

```
se C então                 Ω(N), O(N²)
    θ(N)
senão
    θ(N²)
```

```
se C então                 θ(N²)
    θ(N²)
senão
    θ(N²)
```

```
se C então                 Ω(1), O(N²)
    θ(N²)
```

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Sequência de passos

$$\Theta(N) + \Theta(N^2) = \Theta(N^2)$$

$\Theta(N)$

$\Theta(N^2)$

$$O(N^3) + \Theta(N^2) = \Omega(N^2), O(N^3)$$

$O(N^3)$

$\Theta(N^2)$

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Iteração

```
para i ← 1 até N faça
```
$\theta(N^3)$

$\theta(N^2)$

```
para i ← 1 até N faça
```
$\Omega(N^2)$, $O(N^3)$

$\Omega(N)$, $O(N^2)$

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Iteração

```
enquanto C faça
```

$\theta(k \times N)$

$\theta(N)$

onde *k* é o número de iterações necessárias até que C se torne falso. Tal valor deve ser determinado analisando-se o algoritmo.

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

Bloco:

- Iteração

$\theta(N^2)$

```
i ← 1
enquanto i ≤ N faça
    θ(N)
    i ← i + 1
```

$k = N$

$\theta(N \log N)$

```
i ← 1
enquanto i ≤ N faça
    θ(N)
    i ← i * 2
```

$k = \log N$

Complexidade de Algoritmos

```
funçao encontraMenor( vetor[n]: inteiro, n: inteiro):inteiro
inicio
        i, menor: inteiro
        menor = vetor[0]
        para i:=1 até n-1 faça
                se (vetor[i] <  menor) então
                        menor := vetor[i]
        fim-para
        retorne menor
fim
```

Algoritmos e Estruturas de Dados I

Complexidade de Algoritmos

```
funçao encontraMenor( vetor[n]: inteiro, n: inteiro):inteiro      θ(n)
inicio
        i, menor: inteiro
        menor = vetor[0]                θ(1)
        para i:=1 até n-1 faça          θ(n)
                se (vetor[i] <  menor) então        θ(1)
                        menor := vetor[i]           θ(1)
        fim-para
        retorne menor                   θ(1)
fim
```

Algoritmos e Estruturas de Dados I

EXERCÍCIOS

1) Preencha com V ou F se a função de cada linha corresponde ou às notações das colunas.

	$O(n)$	$\Omega(n)$	$\theta(n)$	$O(n^2)$	$\Omega(n^2)$	$\theta(n^2)$
50						
$5n + 2$						
$3n^2 + 2n$						
$4n^3$						

Algoritmos e Estruturas de Dados I

<u>EXERCÍCIOS</u>

2) Faça um algoritmo que armazene valores em um vetor e imprima “ORDENADO” se o vetor estiver em ordem crescente. Qual seria sua função de complexidade de pior caso e sua ordem de complexidade na notação O?

FIM