



DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Introdução à Análise de Algoritmos

Sedgewick: Capítulo 2
CLRS: Capítulo 3

IAED



Introdução à Análise de Algoritmos

- Análise de Algoritmos
- Crescimento de Funções
- Notação Assintótica
- Exemplos

Algoritmo

- Procedimento computacional bem definido que aceita uma dada entrada e produz uma dada saída
 - Ferramenta para resolver um problema computacional bem definido
 - Calcular a média de um conjunto de valores
 - Ordenação de sequências de valores
 - Caminhos mais curtos em grafos orientados
 - etc.

Mudar Maiúsculas para Minúsculas

- Entrada: Vector de caracteres com o texto
- Objectivo: Mudar todas as letras maiúsculas do texto de entrada para as respectivas letras minúsculas
- Saída: Vector de entrada alterado

Mudar Maiúsculas para Minúsculas (1)

```
void lower(char s[])
{
    int i;

    for (i=0; i < strlen(s); i++)
        if (s[i] >= 'A' && s[i] <= 'Z')
            s[i] -= ('A' - 'a');
}
```

Dada uma string s,
retorna o número
de caracteres de s

- CPU Time: 16,82s para 1.000.000 caracteres

Mudar Maiúsculas para Minúsculas (2)

```
void lower(char s[])
{
    int i, len = strlen(s);

    for (i=0; i < len; i++)
        if (s[i] >= 'A' && s[i] <= 'Z')
            s[i] -= ('A' - 'a');
}
```

O que mudou?

- CPU Time: 0,01s para 1.000.000 caracteres

Análise de Algoritmos

- *Eficiência*

- Escolha do algoritmo certo!!
- Vamos aprender a mapear um algoritmo numa determinada classe de eficiência / complexidade

- Medidas de eficiência & complexidade

- Tempo (tempo de execução) & Espaço (memória)

- Como comparar dois algoritmos diferentes?

- Poderia correr o programa e já está!! Porque é que isto não é uma boa ideia?
 - Depende da linguagem, da máquina, do input, etc.
 - Não sabemos como o programa escala nos dados maiores.

Análise de Algoritmos

- **Alternativa:** *encontrar uma relação matemática que dependa do tamanho da entrada*
 - Qual é o n^o de “passos básicos” requeridos pelo algoritmo em função do “**tamanho**” do problema (o *input size*)?
 - Exemplos:
 - No problema de mudar os caracteres de maiúsculas para minúsculas, o número de caracteres da string de entrada é uma medida razoável
 - Nos algoritmos de ordenação, uma medida razoável é o número de elementos a ordenar
 - Num grafo as medidas utilizadas são o número de vértices e o de arcos

Análise de Algoritmos

- **Alternativa:** encontrar uma relação matemática que dependa do tamanho da entrada
 - Qual é o nº de “passos básicos” requeridos pelo algoritmo em função do “tamanho” do problema (o *input size*)?
 - O que é um “passo básico”?

Vamos assumir que é um passo que toma um tempo constante (não depende do tamanho do problema)

Crescimento de Funções

- Padrões **típicos**

| | |
|------------|--|
| 1 | Se o número de instruções de um programa for executado um número limitado/constante de vezes |
| $\log N$ | Tempo de execução é logarítmico qd se divide continuamente o input ao meio (e.g. pesquisa binária) |
| N | Tempo de execução de um programa é linear quando existe algum processamento para cada elemento de entrada |
| $N \log N$ | Tipicamente, quando um problema é resolvido através da resolução de um conjunto de sub-problemas, e combinando posteriormente as suas soluções |

- **Cuidado:** um programa só pode ter uma complexidade sub-linear, se ignorar uma parte do input!
- Pesquisa binaria é logarítmica, supondo que o vector já foi lido e ordenado **previamente**

Crescimento de Funções

- Tempos de execução típicos

$$N^2$$

Tempo de execução de um programa é **quadrático**; quando a dimensão da entrada duplica, o tempo aumenta 4x

$$N^3$$

Tempo de execução de um programa é **cúbico**; quando a dimensão da entrada duplica, o tempo aumenta 8x

$$2^N$$

Tempo de execução de um programa é **exponencial**; quando a dimensão da entrada duplica, o tempo aumenta para o quadrado !

Crescimento de Funções

- Comparação (*1 passo = 1 segundo*)

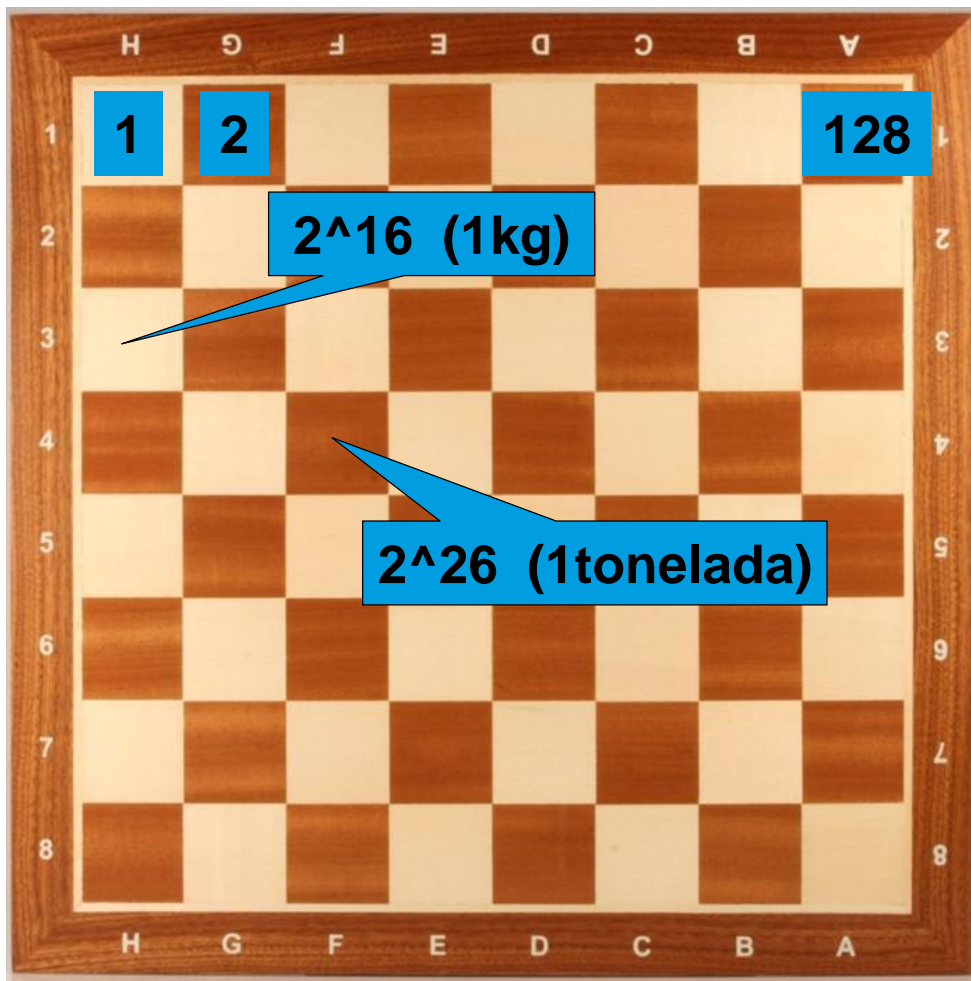
| Ordem | $N = 10$ | 10s |
|----------|-----------|-------------|
| N^2 | 10^2 | 1.7 minutos |
| N^4 | 10^4 | 2.8 horas |
| N^5 | 10^5 | 1.1 dias |
| N^6 | 10^6 | 1.6 semanas |
| N^7 | 10^7 | 3.8 meses |
| N^8 | 10^8 | 3.1 anos |
| N^9 | 10^9 | 3.1 décadas |
| N^{10} | 10^{10} | 3.1 séculos |

Crescimento de Funções

- Comparação

| $\log N$ | \sqrt{N} | N | $N \log N$ | N^2 |
|----------|------------|----------------|------------|---------------|
| 3 | 3 | 10 | 33 | 100 |
| 7 | 10 | 100 | 664 | 10000 |
| 10 | 32 | 1000 | 9966 | 1000000 |
| 13 | 100 | 10000 | 132877 | 100000000 |
| 20 | 1000 | 1000000 | 19931569 | 1000000000000 |

Arroz e funções exponenciais



- 1 grão de arroz na 1ª casa
- 2 grãos na 2ª
- Continua dobrando

Número de átomos no
universo observável
 2^{82}

$2^{64} =$
 $1.8446744e+19$
($3e+17$ toneladas)

Vamos voltar ao exemplo:

Mudar Maiúsculas para Minúsculas

- Entrada: Vector de caracteres com o texto
- Objectivo: Mudar todas as letras maiúsculas do texto de entrada para as respectivas letras minúsculas
- Saída: Vector de entrada alterado

Tempo de Execução

```
1 void lower(char s[])
2 {
3     int i;
4
5     for (i=0; i < strlen(s); i++)
6         if (s[i] >= 'A' && s[i] <= 'Z')
7             s[i] -= ('A' - 'a');
8 }
```

- Seja N o número de caracteres da string de entrada
 - Linha 5: `strlen(s)` é uma operação que percorre os N caracteres
 - Linha 5 é executada N vezes
 - Logo, algoritmo é **quadrático** N^2 no tamanho da string

Tempo de Execução

```
1 void lower(char s[])
2 {
3     int i, len = strlen(s);
4
5     for (i=0; i < len; i++)
6         if (s[i] >= 'A' && s[i] <= 'Z')
7             s[i] -= ('A' - 'a');
8 }
```

- Operação `strlen(s)` executada apenas 1 vez (linha 3)
- Ciclo `for` tem N iterações
- Operações do ciclo `for` são limitadas e cada uma é executada em tempo constante, ou seja, não depende da dimensão dos dados
- Logo, algoritmo é **linear** no tamanho da string.

Tempo de Execução

```
1 void lower(char s[])
2 {
3     int i, len = strlen(s), diff = ('A' - 'a');
4
5     for (i=0; i < len; i++)
6         if (s[i] >= 'A' && s[i] <= 'Z')
7             s[i] -= diff ;
8 }
```

- E neste caso?
- Mudança de operações realizadas em tempo constante não é relevante. Complexidade da solução algorítmica mantém-se.
- Se antes o tempo de execução “escalava” com N, agora continua a “escalar” com N.

Análise de Algoritmos

- Mas que inputs devemos usar para calcular a complexidade de um algoritmo?
- Uso do **pior caso** como valor para a complexidade
 - Representa um limite superior no tempo de execução
 - Ocorre numerosas vezes
 - O valor médio é muitas vezes próximo do pior-caso
 - É, geralmente, mais fácil de calcular
 - Evita surpresas!
- Uso do **melhor caso** como valor para a complexidade
- Uso do **caso médio** como valor para a complexidade
 - Importante em algoritmos probabilísticos
 - É necessário saber a distribuição dos problemas

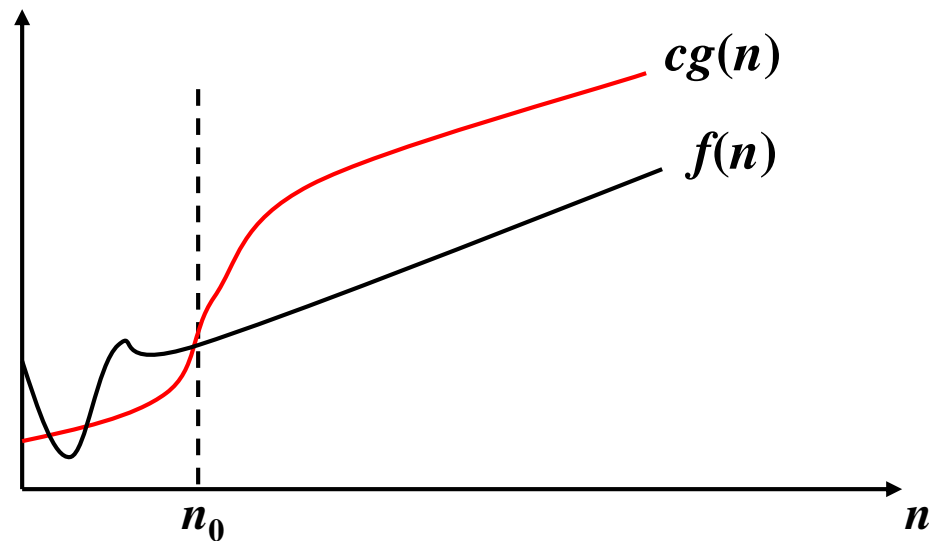
Notação Assimptótica

- **Objectivo:** caracterizar matematicamente os tempos de execução dos algoritmos para tamanhos arbitrários das entradas
- A notação assimptótica permite estabelecer **taxas de crescimento** dos tempo de execução dos algoritmos em função dos tamanhos das entradas
- Constantes multiplicativas e aditivas tornam-se irrelevantes
 - E.g.: tempo de execução de cada instrução não é essencial para o comportamento assimptótico de um algoritmo

Limite Assimptótico Superior

- Notação O : Limite Assimptótico **Superior**
- **Permite aferir a complexidade no *pior caso***

- $O(g(n)) =$
 $\{ f(n) : \text{existem constantes positivas } c \text{ e } n_0, \text{ tal que } 0 \leq f(n) \leq cg(n), \text{ para } n \geq n_0 \}$
- $f(n) = O(g(n))$, significa $f(n) \in O(g(n))$



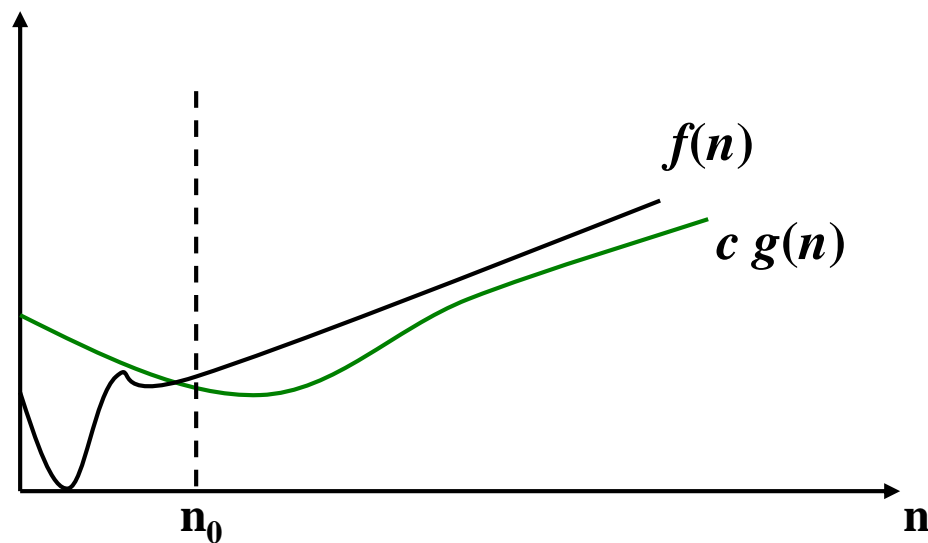
Exemplo: procura em vector

- Solução: Analisar sequencialmente todo o vector até encontrar o elemento
 - No **pior caso**, vector é analisado até ao fim
 - Complexidade: $O(N)$
 - Outra forma de pensar: Se $f(N)$ for a função que descreve o pior caso para o crescimento do tempo de execução do algoritmo, $f(N)=O(N)$ significa que $f(N)$ **pertence** ao conjunto de funções $O(N)$

Limite Assimptótico Inferior

- Notação Ω : Limite Assimptótico **Inferior**
- **Permite aferir a complexidade no *melhor caso***

- $\Omega(g(n)) =$
 $\{ f(n) : \text{existem constantes positivas } c \text{ e } n_0, \text{ tal que } 0 \leq cg(n) \leq f(n), \text{ para } n \geq n_0 \}$
- $f(n) = \Omega(g(n))$, significa $f(n) \in \Omega(g(n))$



Exemplo: procura em vector

- Solução: Analisar sequencialmente todo o vector até encontrar o elemento
 - No **pior caso**, vector é analisado até ao fim
 - Complexidade: $O(N)$
 - Outra forma de pensar: Se $f(N)$ for a função que descreve o pior caso para o crescimento do tempo de execução do algoritmo, $f(N)=O(N)$ significa que $f(N)$ **pertence** ao conjunto de funções $O(N)$
 - No **melhor caso**, a primeira posição do vector é o elemento que procuramos
 - Complexidade constante no melhor caso, i.e., **Limite Assintótico Inferior** = $\Omega(1)$ — o melhor caso pertence ao conjunto de funções descritas por $\Omega(1)$.

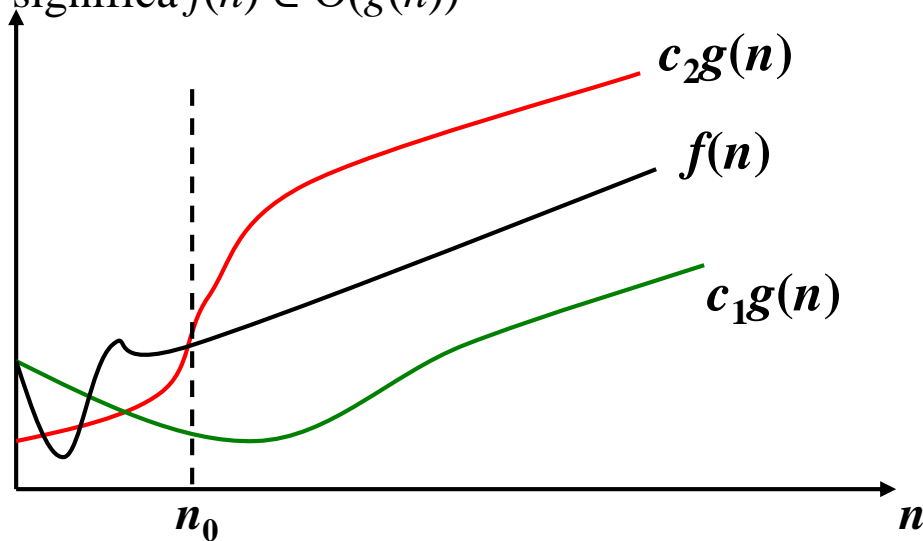
Limite Assimptótico Apertado

- Notação Θ : Limite Assimptótico **Apertado**
- Uma função $f(n)$ diz-se $\Theta(g(n))$ se e só se $f(n)$ for $O(g(n))$ e $\Omega(g(n))$

– $\Theta(g(n)) =$

$\{ f(n) : \text{existem constantes positivas } c_1, c_2, \text{ e } n_0, \text{ tal que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ para } n \in n_0 \}$

– $f(n) = \Theta(g(n))$, significa $f(n) \in \Theta(g(n))$



Limite Assimptótico Apertado

- Notação Θ : Limite Assimptótico **Apertado**
- Uma função $f(n)$ diz-se $\Theta(g(n))$ se e só se $f(n)$ for $O(g(n))$ e $\Omega(g(n))$

Vamos supor que eu sei exactamente como o meu algoritmo se comporta:

- $f(n) = 0.1 n^3 + 1000 n^2 + 10^9$

– É $O(n^3)$?

...e $O(n^7)$?

– É $\Omega(n^3)$?

...e $\Omega(n^2)$?

É $\Theta(n^3)$

– Não é $O(n^\alpha)$ para $\alpha < 3$. Porquê?

– Não é $\Omega(n^\alpha)$ para $\alpha > 3$. Porquê?

– Não é $\Theta(n^\alpha)$ para $\alpha \neq 3$. Porquê?

Exercício (Verdadeiro ou Falso)

- Se um algoritmo é $O(N^2)$ então também é $O(N^3)$.

Verdadeiro (embora, evidentemente, seja uma afirmação de pouca utilidade).

- Se o tempo de execução de um algoritmo, no pior caso, escala com $3N^2$ então é $O(N^2)$.

Verdadeiro.

- O tempo de execução do algoritmo G , escala com $2N^3 + N$ no pior caso e, no melhor caso, apenas com N^3 . Logo, G é $\Theta(N^3)$.

Verdadeiro.

```
#include <stdio.h>
#include <string.h>
```

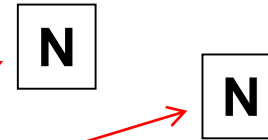
```
void toupper(char s[]){
    int i;

    for (i = 0; i < strlen(s); i++)
        if (s[i] >= 'a' && s[i] <= 'z')
            s[i] -= ('a' - 'A');
}
```

```
int main(){
    char s = "iaed";

    toupper(s);
    printf("%s\n", s);
    return 0;
}
```

Exercício de exame



Solução: $O(N^2)$

Indique a expressão da complexidade assintótica mais apertada para descrever o comportamento no pior caso da função `toupper` em função do comprimento N da sequência de caracteres `s`. Note que apenas será aceite como resposta correcta a expressão mais apertada para o limite assintótico.

II.c) Considere o seguinte programa na linguagem C:

Exercício de exame

```
#define DIM2 23

void test_1_funcao(float tempos[][DIM2], int n, int m)
{
    int i, j;
    float total;

    for (j = 0; j < m; j++) {
        for (i = 0, total = 0.0; i < n; i++)
            total += tempos[i][j];
        printf("Avg %d: %f\n", j, total/n);
    }
}
```

Qual é a complexidade assintótica desta função em termos dos parâmetros da mesma?

Solução:

Complexidade: $O(nm)$

II.d) Considere o seguinte programa na linguagem C.

Exercício de exame

```
#include <stdio.h>
#include <string.h>
```

```
#define N 3
#define M 3
```

```
void compara (char v[N][M], char c[]){
    int i;
```

```
    for (i = 0; i < N-1; i++)
```

```
        if (!strcmp(v[i], v[i+1]))
```

```
            c[i] = '1';
```

```
        else
```

```
            c[i] = '0';
```

```
}
```

```
int main(){
```

```
    char v[N][M] = {"s1", "s1", "s2"};
```

```
    char c[N-1];
```

```
    compara (v,c);
```

```
    printf("%c %c\n", c[0], c[1]);
```

```
    return 0;
```

```
}
```

Para cada string na posição i

Verifica se a string i é igual à string i+1

Guarda o resultado na posição i do array c

N: número de strings

c[]: array de tamanho N-1

M: tamanho de cada string

Para este programa, indique a expressão da complexidade assintótica mais apertada para descrever o comportamento no pior caso em função de N e de M. Note que apenas será aceite como resposta correcta a expressão mais apertada para o limite assintótico.

II.d) Considere o seguinte programa na linguagem C.

```
#include <stdio.h>
#include <string.h>

#define N 3
#define M 3

void compara (char v[N][M], char c[]){
    int i;

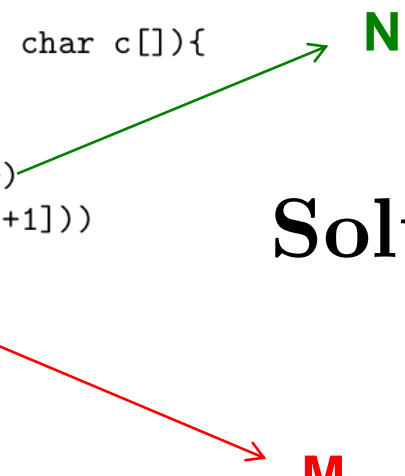
    for (i = 0; i < N-1; i++){
        if (!strcmp(v[i], v[i+1]))
            c[i] = '1';
        else
            c[i] = '0';
    }
}

int main(){
    char v[N][M] = {"s1", "s1", "s2"};
    char c[N-1];

    compara (v,c);
    printf("%c %c\n", c[0], c[1]);
    return 0;
}
```

Exercício de exame

Solução: $O(NM)$



Para este programa, indique a expressão da complexidade assintótica mais apertada para descrever o comportamento no pior caso em função de N e de M. Note que apenas será aceite como resposta correcta a expressão mais apertada para o limite assintótico.