

# Análise da Complexidade III

Joaquim Madeira

18/03/2021

# Sumário

- Recap
- Ordens de complexidade
- Linear Search – Procura sequencial num **array não ordenado**
- Linear Search – Procura sequencial num **array ordenado**
- Binary Search – Procura binária
- Sugestão de leitura

Let's  
RECAP

# Recapitulação

# Multiplicação de matrizes quadradas

```
for(int i=0; i<n; i++) {  
    for(int j=0; j<n; j++) {  
        c[i][j] = 0;  
        for(int k=0; k<n; k++) {  
            c[i][j] += a[i][k] * b[k][j];  
        }  
    }  
}
```



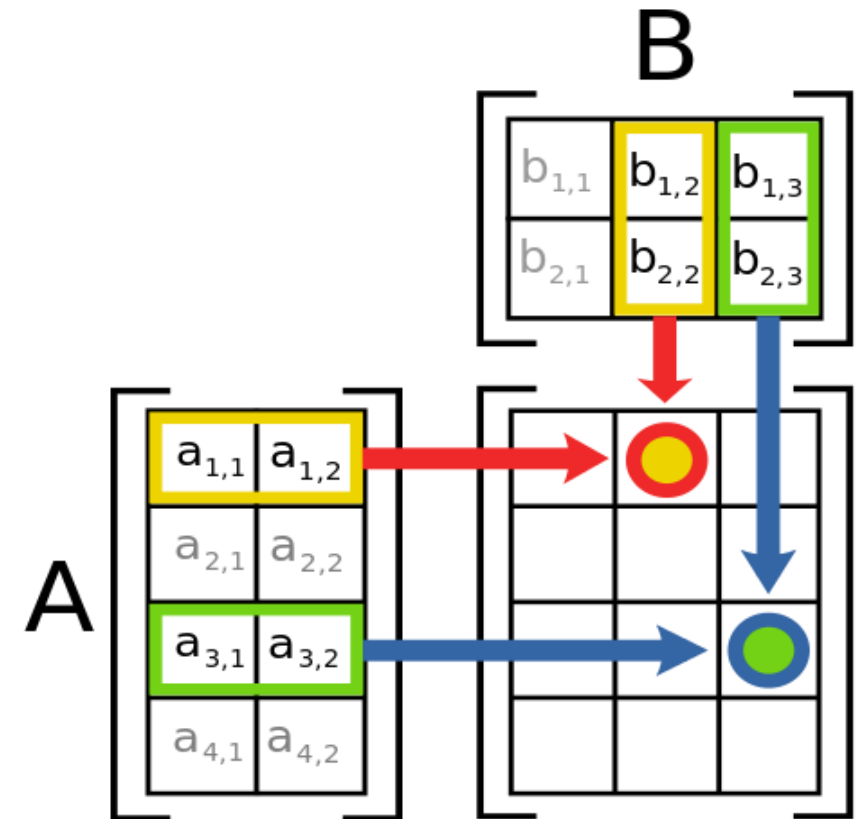
$$\sum_{i=0}^{n-1} \left[ \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} 1 \right) \right]$$

Algoritmo **cúbico**

# Multiplicação de matrizes – Caso geral

$$A(m \times n) \times B(n \times p) = C(m \times p)$$

- Quantas **multiplicações** são efetuadas ?  $m \times n$
- Ordem de complexidade ?  $m \times n \times p$






[Wikimedia.org]

Best case, Worst case, Average Case

$$B(n) = \min_{I \in D_n} t(I) \quad W(n) = \max_{I \in D_n} t(I)$$

$$A(n) = \sum_{I \in D_n} p(I) \times t(I)$$

# Procura do maior elemento

```
int searchMax( int a[], int n ) {  
    int indexMax = 0;   
    for( int i=1; i<n; i++ ) {  
        if( a[i] > a[indexMax] ) {  (n - 1) comps.  
            indexMax = i;   
        }  
    }  
    return indexMax;  
}
```

Atribuições a indexMax:  
 $B(n) = 1$   
 $W(n) = n$   
 $A(n) \approx n/2$

# Notações habituais

notation	provides	example	shorthand for	used to
<b>Big Theta</b>	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ $\vdots$	classify algorithms
<b>Big Oh</b>	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ $\vdots$	develop upper bounds
<b>Big Omega</b>	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ $N^5$ $N^3 + 22 N \log N + 3 N$ $\vdots$	develop lower bounds

[Sedgewick & Wayne]



# Fizeram ?

- $T(n) = 10 n^2 + 100 n - 23$

$$T(n) \stackrel{p}{?} O(n^2)$$

$$T(n) \stackrel{p}{?} O(n^3)$$

$$T(n) \stackrel{np}{?} O(n)$$

$$T(n) \stackrel{p}{?} \Omega(n^2)$$

$$T(n) \stackrel{np}{?} \Omega(n^3)$$

$$T(n) \stackrel{p}{?} \Omega(n)$$

$$T(n) \stackrel{p}{?} \Theta(n^2)$$

$$T(n) \stackrel{np}{?} \Theta(n^3)$$

$$T(n) \stackrel{np}{?} \Theta(n)$$

# Exemplo – Importância do termo de maior grau

- $f(n) = a n^2 + b n + c$ , com  $a = 0.0001724$ ,  $b = 0.0004$  e  $c = 0.1$

n	f(n)	$a n^2$	$a n^2 / f(n)$
125	2.8	2.7	94.7%
250	11.0	10.8	98.2%
500	43.4	43.1	99.3%
1000	172.9	172.4	99.7%

# Ordens de Complexidade

# Ordens de Complexidade/Classes de Eficiência

- $O(1)$  : constante
  - Que algoritmos?
- $O(\log n)$  : logarítmico
  - E.g., **diminuir-para-reinar**
- $O(n)$  : linear
  - Processar todos os elementos de um array, uma lista, etc.
- $O(n \log n)$  : n-log-n
  - E.g., **dividir-para-reinar**

# Ordens de Complexidade/Classes de Eficiência

- $O(n^k)$  : polinomial (quadrático, cúbico, etc.)
  - $k$  ciclos encastelados
- $O(2^n)$  : exponencial
  - Gerar **todos os subconjuntos** de um conjunto com  $n$  elementos
- $O(n!)$  : fatorial
  - Gerar **todas as permutações** de um conjunto com  $n$  elementos

# Ordens de Complexidade/Classes de Eficiência

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	constant	<code>a = b + c;</code>	statement	add two numbers	1
$\log N$	logarithmic	<pre>while (N &gt; 1) {   N = N / 2; ... }</pre>	divide in half	binary search	$\sim 1$
$N$	linear	<pre>for (int i = 0; i &lt; N; i++) {   ... }</pre>	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	$\sim 2$
$N^2$	quadratic	<pre>for (int i = 0; i &lt; N; i++)   for (int j = 0; j &lt; N; j++)   {     ...   }</pre>	double loop	check all pairs	4
$N^3$	cubic	<pre>for (int i = 0; i &lt; N; i++)   for (int j = 0; j &lt; N; j++)     for (int k = 0; k &lt; N; k++)     {       ...     }</pre>	triple loop	check all triples	8
$2^N$	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$



[Sedgewick & Wayne]

# Exemplo – Contagem de operações

$n$	1	2	4	8	16	32	64	128	256
$M(n)$	1	3	10	36	136	528	2080	8256	32896

- $M(n)$  : número de operações efetuadas
- Ordem de complexidade ?  $O(n^2)$
- Expressão para o número de operações ?  $f(n)=n(n+1)/2$

# Outro exemplo

$n$	1	2	3	4	5	6	7	8	9	10
$M(n)$	1	3	7	15	31	63	127	255	511	1023

- $M(n)$  : número de operações efetuadas
- Ordem de complexidade ?  $O(2^n)$
- Expressão para o número de operações ?  $f(n)=2^n - 1$



# Tempo de execução – Estimativas

order of growth of time		for a program that takes a few hours for input of size N			
description	function	2x factor	10x factor	predicted time for 10N	predicted time for 10N on a 10x faster computer
<i>linear</i>	$N$	2	10	a day	a few hours
<i>linearithmic</i>	$N \log N$	2	10	a day	a few hours
<i>quadratic</i>	$N^2$	4	100	a few weeks	a day
<i>cubic</i>	$N^3$	8	1,000	several months	a few weeks
<i>exponential</i>	$2^N$	$2^N$	$2^{9N}$	never	never

**Predictions on the basis of order-of-growth function**

[Sedgewick & Wayne]

Procura/Pesquisa sequencial

# Linear Search

- Array não ordenado

# Procura sequencial – Comparações ?

```
int search( int a[], int n, int x ) {  
    for( int i=0; i<n; i++ ) {  
        if( a[i] == x ) {  
            return i;  
        }  
    }  
    return -1;  
}
```



$B(n) = 1$

$W(n) = n$

$A(n) = ?$   $n/2$

# Caso Médio – Abordagem estruturada

- Estabelecer um **cenário**
- Identificar a **operação básica** a contar
- Identificar os **casos/configurações possíveis**
  - **Quantos** são ?
- Contar o **nº de operações** realizadas para cada um dos casos
- Atribuir uma **probabilidade** a cada um dos casos possíveis
- Construir uma **tabela**
- Calcular o **nº de operações para o caso médio**

# 1 – Valor procurado pertence ao array

- O valor procurado **pertence ao array**
- Contar o **número de comparações** necessárias para o encontrar
- Valor encontrado na **1ª posição**, ou na **2ª posição**, ou ...
- Necessária **1 comparação**, ou **2 comparações**, ou ...
- Situações **equiprováveis**
- Tabela **?**

# 1 – Elemento procurado pertence ao array

Casos possíveis		Nº de comparações	Probabilidade
$l_0$	É o 1º elemento	1	$1/n$
$l_1$	É o 2º elemento	2	$1/n$
$l_2$	É o 3º elemento	3	$1/n$
...	....	...	...
$l_{n-1}$	É o último elemento	$n$	$1/n$

$i+1$

$$A(n) = \sum_{i=0}^{n-1} \frac{1}{n} \times (i + 1) = \frac{1}{n} \sum_{i=0}^{n-1} (i + 1) = \frac{1}{n} \times \frac{n \times (n + 1)}{2} = \frac{n + 1}{2} \approx \frac{n}{2}$$

## 2 – Valor procurado pode não pertencer !

- O valor procurado **pertence** ao array com **probabilidade  $p$**
- O valor procurado **não pertence** ao array com **probabilidade  $(1 - p)$**
- Quantos casos são ?
- Contar o **número de comparações** para cada um dos casos
- Valor encontrado na **1ª posição**, ou na **2ª posição**, ou ...
- Necessária **1 comparação**, ou **2 comparações**, ou ...
- Que **probabilidade** atribuir a **cada caso** ?
- Tabela ?

## 2 – Valor procurado pode não pertencer !

Casos possíveis		Nº de comparações	Probabilidade
$I_0$	É o 1º elemento	1	$p/n$
$I_1$	É o 2º elemento	2	$p/n$
$I_2$	É o 3º elemento	3	$p/n$
...	....	...	...
$I_{n-1}$	É o último elemento	$n$	$p/n$
$I_n$	Não encontrado !	$n$	$(1 - p)$

$$A(n) = \sum_{i=0}^{n-1} \frac{p}{n} \times (i + 1) + (1 - p) \times n = \frac{p \times (n + 1)}{2} + (1 - p) \times n$$



## 2 – Valor procurado pode não pertencer !

$$A(n) = \frac{p \times (n + 1)}{2} + (1 - p) \times n$$

- Se  $p = 1$ , então  $A(n) = (n + 1) / 2 \approx n / 2$
- Se  $p = 50\%$ , então  $A(n) = (n + 1) / 4 + n / 2 \approx 3 \times n / 4$
- Se  $p = 25\%$ , então  $A(n) = (n + 1) / 8 + 3 \times n / 4 \approx 7 \times n / 8$

Procura/Pesquisa Sequencial

# Linear Search

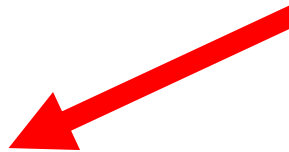
- Array ordenado

# Procura sequencial – Comparações ?

```
int search( int a[], int n, int x ) {  
    int stop = 0; int i;  
    for( i=0; i<n; i++ ) {  
        if( x <= a[i] ) {  
            stop = 1; break;  
        }  
    }  
    if( stop && x == a[i] ) return i;  
    return -1;  
}
```

nº de comparações Bc = 2

nº de comparações Wc = n+1



# Ordem da conjunção !!

# Melhor Caso e Pior Caso

- Valor procurado pode **pertencer** ou **não pertencer** ao array
- $B(n) = 2$                       - Quando ?      Existe duas atribuições ou comparações
- $W(n) = (n + 1)$               - Quando ?      Existem muitas comparações

# Melhor Caso e Pior Caso

- Valor procurado pode **pertencer** ou **não pertencer** ao array
- $B(n) = 2$ 
  - Valor procurado **é igual ao 1º elemento**  
**OU**
  - Valor procurado **é menor do que o 1º elemento**
- $W(n) = (n + 1)$ 
  - Valor procurado **é igual ao último elemento**  
**OU**
  - Valor procurado **está entre o penúltimo e o último**

# Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento		
Sucesso 1	É o 2º elemento		
...	...		
Sucesso (n – 1)	É o último elemento		
Insucesso 0	Menor do que o 1º		
Insucesso 1	Entre o 1º e o 2º		
...	...		
Insucesso (n – 1)	Entre o penúltimo e o último		
Insucesso n	Maior do que o último		

- Quantos são os casos possíveis ? Quantas comparações em cada um ?

# Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento	2	
Sucesso 1	É o 2º elemento	3	
...	...	...	
Sucesso (n - 1)	É o último elemento	n + 1	
Insucesso 0	Menor do que o 1º	2	
Insucesso 1	Entre o 1º e o 2º	3	
...	...	...	
Insucesso (n - 1)	Entre o penúltimo e o último	n + 1	
Insucesso n	Maior do que o último	n	

- Que probabilidade associar a cada caso possível ?

# Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento	2	$1 / (2n + 1)$
Sucesso 1	É o 2º elemento	3	$1 / (2n + 1)$
...	...	...	...
Sucesso (n - 1)	É o último elemento	n + 1	$1 / (2n + 1)$
Insucesso 0	Menor do que o 1º	2	$1 / (2n + 1)$
Insucesso 1	Entre o 1º e o 2º	3	$1 / (2n + 1)$
...	...	...	...
Insucesso (n - 1)	Entre o penúltimo e o último	n + 1	$1 / (2n + 1)$
Insucesso n	Maior do que o último	n	$1 / (2n + 1)$

- $A(n) = ?$



# Caso Médio

$$A(n) = \frac{1}{2n+1} \left\{ \left( \sum_{i=0}^{n-1} (i+2) \right) + \left( \sum_{i=0}^{n-1} (i+2) \right) + n \right\}$$

$$A(n) = \frac{1}{2n+1} \left\{ \frac{n \times (n+3)}{2} + \frac{n \times (n+3)}{2} + n \right\} = n/2 + 7/2 - 7/4 \text{div}((2n+1))$$

$$A(n) \approx \frac{n}{2}$$

- **Comparar** com o cenário do array não ordenado
- É melhor ou pior ? Para array ordenado é melhor

# Tarefa 1

- Considerar uma **probabilidade de 25%** de o valor procurado pertencer ao array **Está resolvido em papel**
- Calcular o **nº de comparações** associadas ao **caso médio** para este cenário


Procura/Pesquisa Binária

# Binary Search

## – Array ordenado

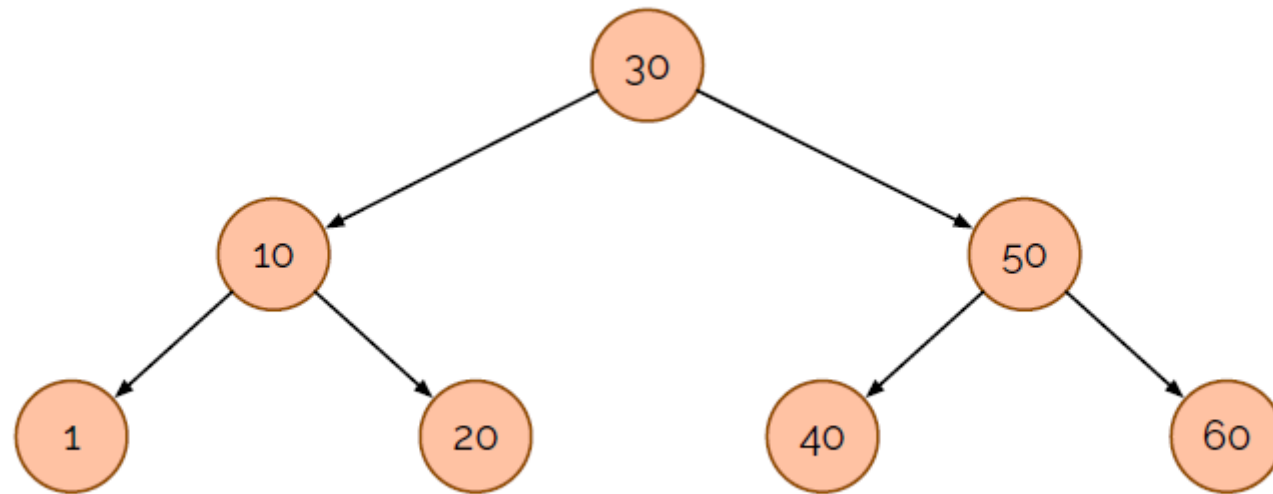
# Procura binária – Nº de iterações do ciclo ?

```
int binSearch( int a[], int n, int x ) {  
    int left = 0; int right = n - 1;  
    while( left <= right ) {  
        int middle = (left + right) / 2;  
        if(a[middle] == x) return middle;  
        if(a[middle] > x) right = middle - 1;  
        else left = middle + 1;  
    }  
    return -1;  
}
```



# Árvore binária

- Dado um **array ordenado com n elementos**
- A sua **representação gráfica como árvore binária** auxilia a compreensão do funcionamento do algoritmo

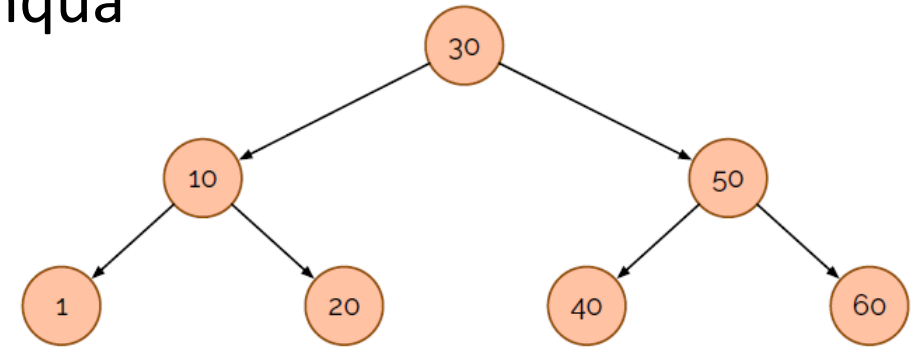


# Melhor Caso

- O 1º elemento consultado é o valor procurado
- 1 comparação
- 1 iteração do ciclo while
- $B(n) = 1$

# Pior Caso – Valor procurado pertence ao array

- Percorrer a árvore até atingir a folha mais longínqua
- Em geral, qual é a forma da árvore ?
- E qual é a sua altura ?



- Array com 1 elemento : ?
- Array com 2 elementos : ?
- Array com 3 elementos : ?
- Array com 4 elementos : ?
- ...
- Array com n elementos : ?

Nº de comparações =  $2^n - 1$

Nº de nodes =  $2^j - 1$  onde j é o nível do nó

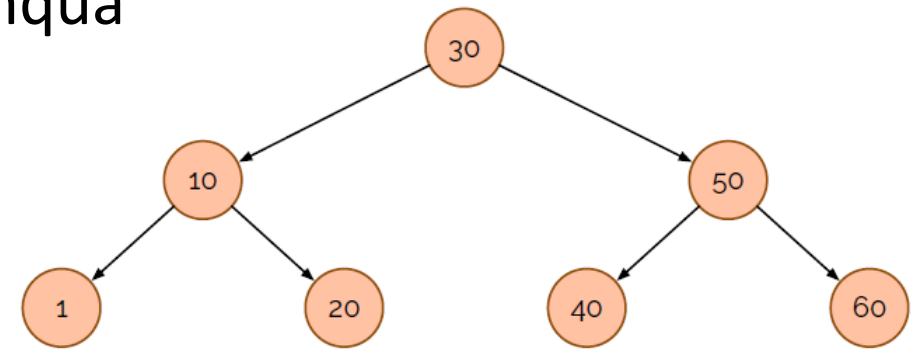
$j = \log(n) + 1$

# Pior Caso – Valor procurado pertence ao array

??

- Percorrer a árvore até atingir a folha mais longínqua
- Em geral, qual é a forma da árvore ?
- E qual é a sua altura ?  $h = 1 + \lceil \lg(n) \rceil$  ou  $\log(n+1)$  base 2

- Array com 1 elemento : 1 iteração
- Array com 2 elementos : 2 iterações
- Array com 3 elementos : 2 iterações
- Array com 4 elementos : 3 iterações
- ...
- **Array com n elementos : ?**   iterações =  $\log(n + 1)$



$$Bc(n) = 1$$

$$Wc(n) = \log(n+1) \text{ ou } \log(n), \text{ base } 2$$

$$Ac(n) = \log(n+1)-1 \text{ ou } \log(n)-1, \text{ base } 2, \text{ for } n=2^k - 1$$



# Tarefa 2

- Para o mesmo cenário – **valor procurado pertence ao array**
- **Array com 3 elementos** – cada um desses valores é **procurado uma vez**
- Qual o **nº médio de iterações** realizadas ?  $M_i = (1 \times 1 + 3 \times 2) / 3 = 2$   
 $M_i$  : é nº o médio de iterações
- **Array com 7 elementos** – cada um desses valores é **procurado uma vez**
- Qual o **nº médio de iterações** realizadas ?  $M_i = (1 \times 1 + 2 \times 2 + 3 \times 4) / 7 = 2$
- **Generalizar !**  $\text{itera\c{c}\~{o}es} = (j_1 \times j_1 + j_2 \times j_2 + \dots + j_n \times (j_n + 1)) / n$ , onde  $j$  é nível de iterações da árvore

# Sugestão de leitura

# Sugestão de leitura

- J. J. McConnell, Analysis of Algorithms, 1<sup>st</sup> Edition, 2001
  - Capítulo 2: secções 2.1, 2.2