

Análise da Complexidade de Algoritmos Recursivos I

09/10/2023

Sumário

- Recap
- Algoritmos recursivos
- Calcular x^n
- Inverter a ordem dos elementos de um array
- Calcular o valor de um determinante
- As Torres de Hanói
- Exercícios adicionais
- Sugestões de leitura

Let's
RECAP

Recapitulação

Selection Sort

- Número fixo de **comparações** :

$$C(n) \approx \frac{n^2}{2}$$

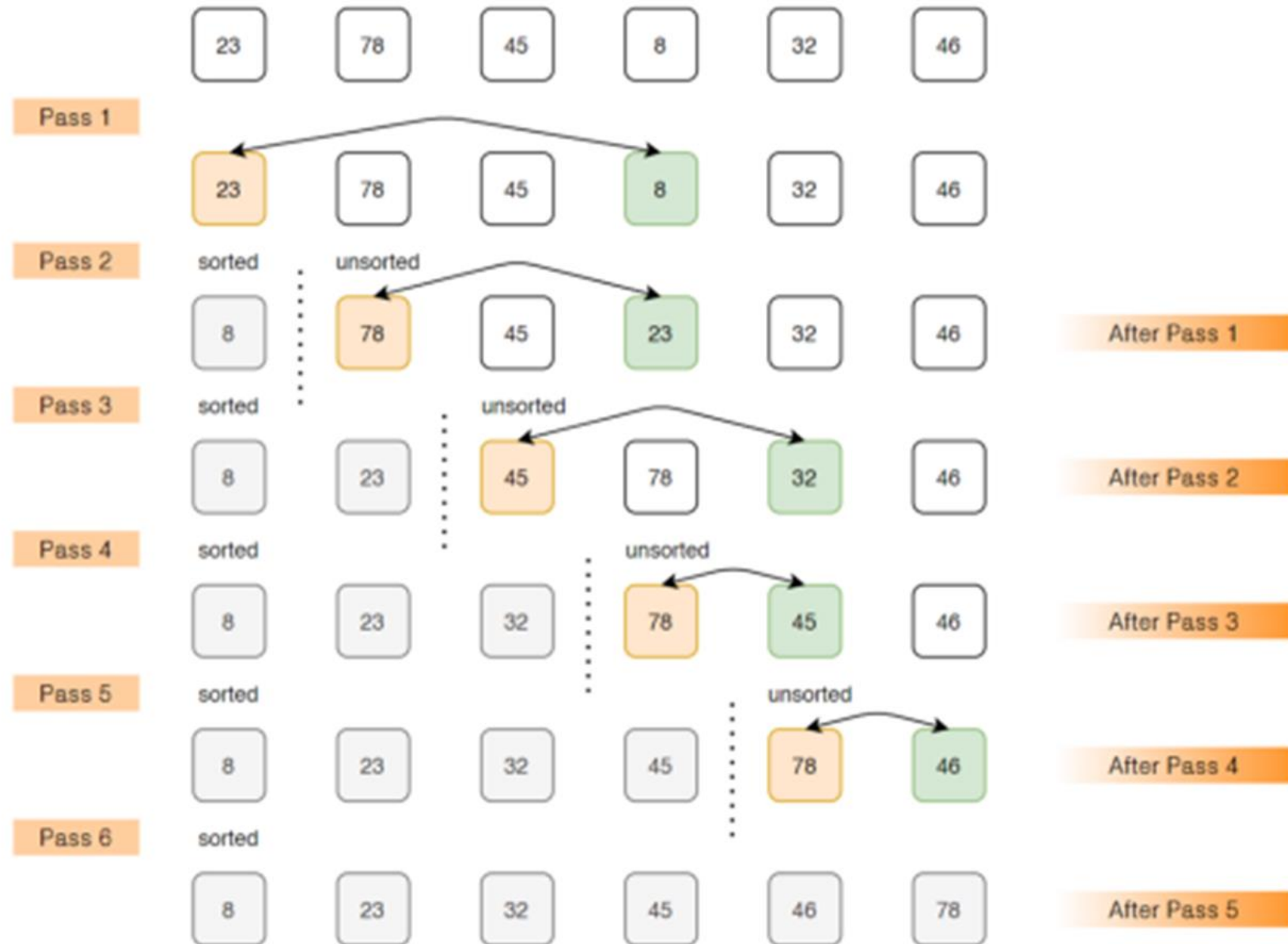
- **Trocas** :

$$W_T(n) = n - 1$$

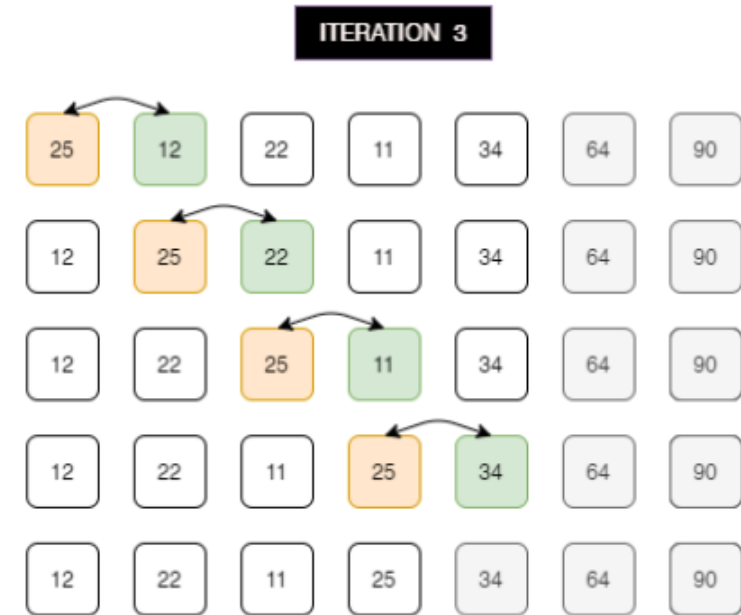
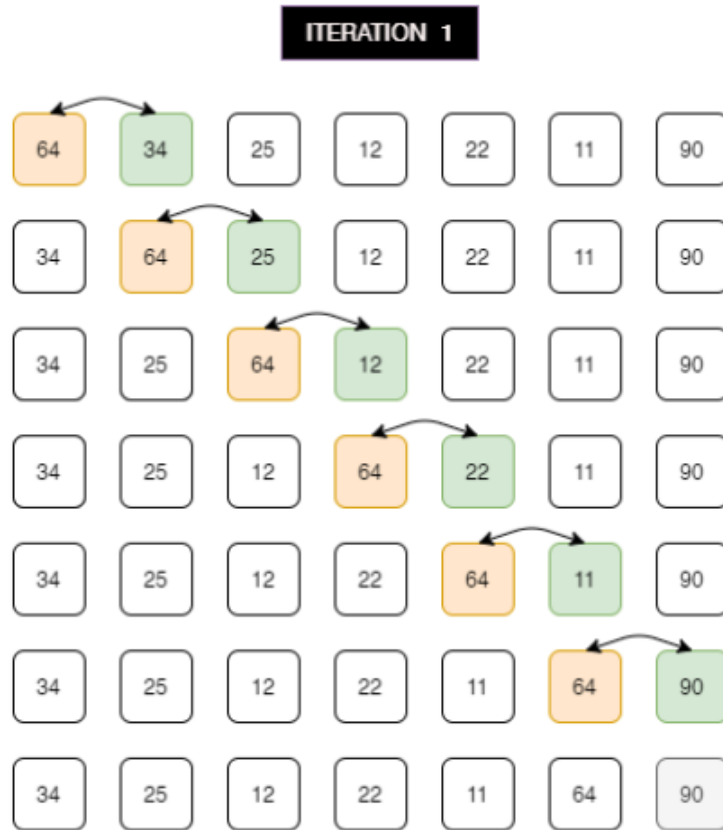
$$A_T(n) \approx n - \ln n$$

$$B_T(n) = 0$$

[Adwiteeya Reyna]



Bubble Sort



[Adwiteeya Reyna]

Bubble Sort

ITERATION 4



ITERATION 7



ITERATION 5



ITERATION 6



- **Comparações :**

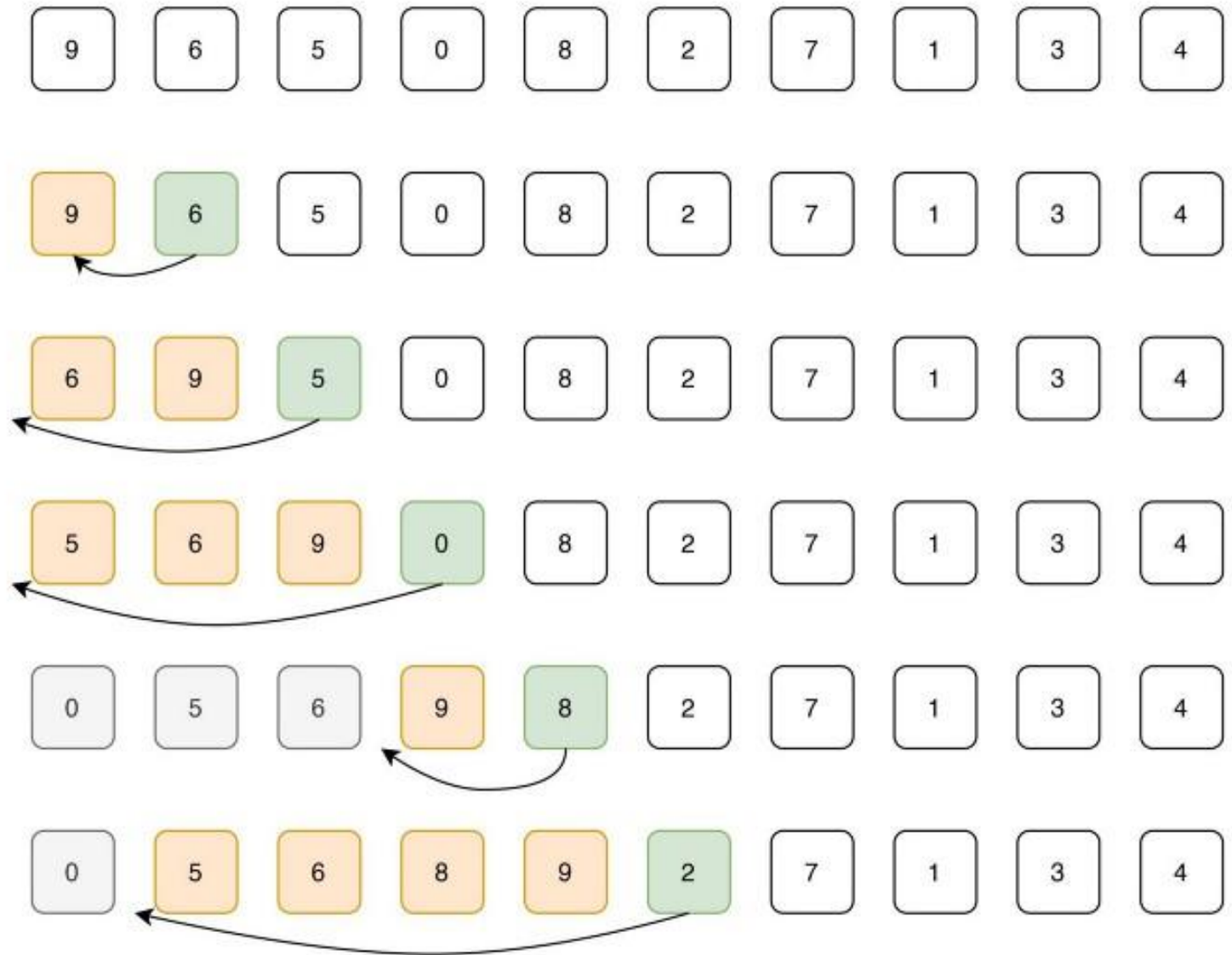
$$W_C(n) \approx \frac{n^2}{2} \quad A_C(n) \approx \frac{n^2}{3} \quad B_C(n) = n - 1$$

- **Trocas :**

$$W_T(n) = W_C(n) \quad A_T(n) \approx \frac{n^2}{6} \quad B_T(n) = 0$$

[Adwiteeya Reyna]

Insertion Sort



[Adwiteeya Reyna]

Insertion Sort

- **Comparações :**

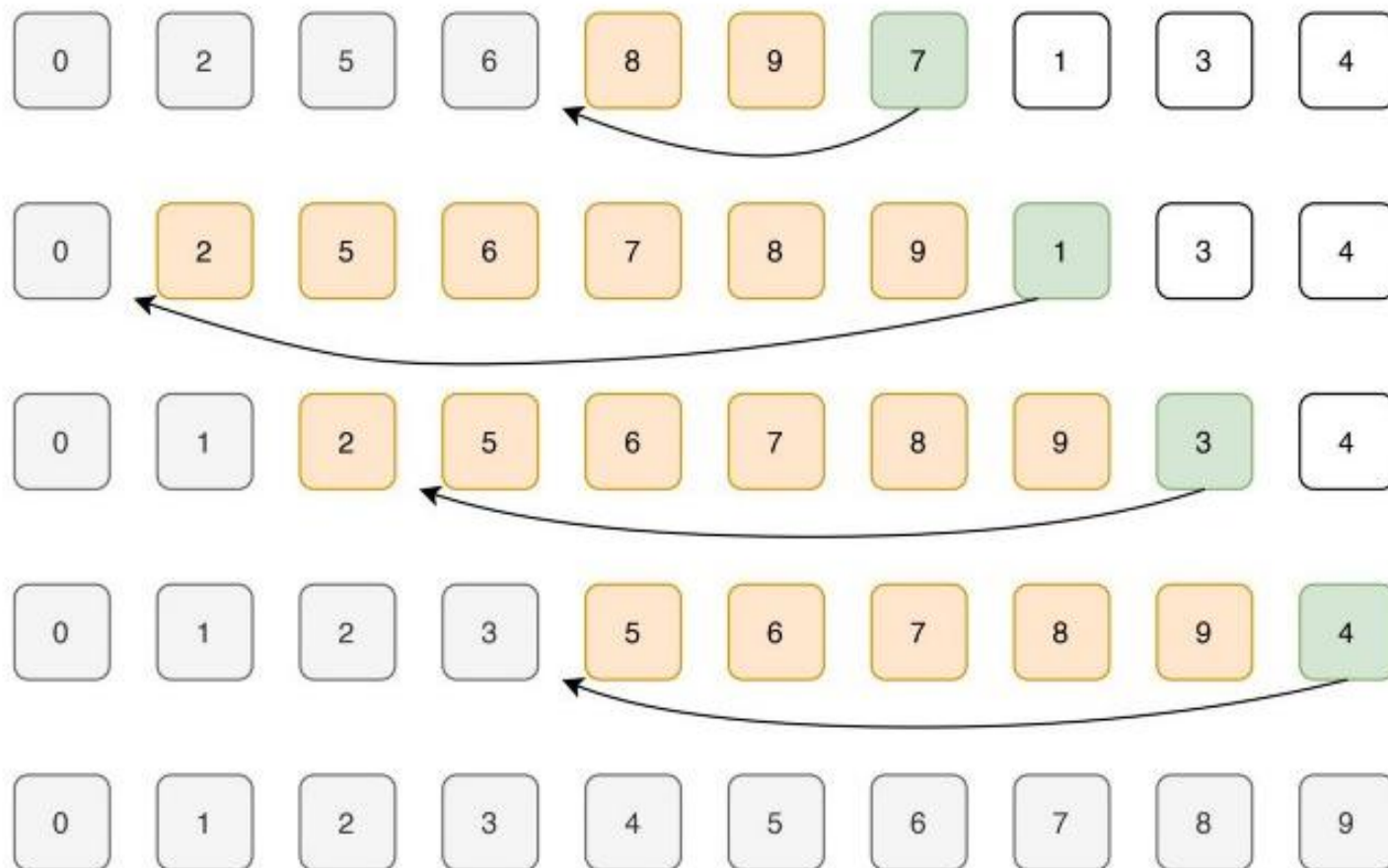
$$W_C(n) \approx \frac{n^2}{2} \quad A_C(n) \approx \frac{n^2}{4}$$

$$B_C(n) = n - 1$$

- **Deslocamentos :**

$$W_D(n) \approx \frac{n^2}{2} \quad A_D(n) \approx \frac{n^2}{8}$$

$$B_D(n) = 0$$



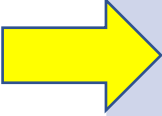
[Adwiteeya Reyna]

Tarefa 1

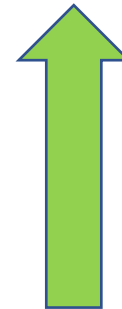
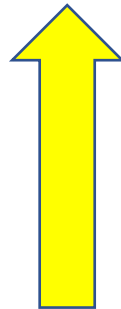
- toptal.com/developers/sorting-algorithms
- Analisar as **animações** disponibilizadas
- Comparar :
 - **Diferentes arrays** para um **mesmo algoritmo**
 - O **mesmo array** para **diferentes algoritmos**




Comparações – Algoritmos Quadráticos



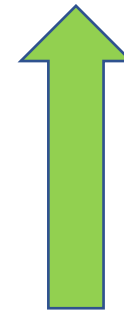
	Pior Caso	Caso Médio	Melhor Caso
Selection Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{2}$
Bubble Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{3}$	$n - 1$
Insertion Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{4}$	$n - 1$



Trocas / Deslocamentos



	Pior Caso	Caso Médio	Melhor Caso
Selection Sort	$n - 1$	$\approx n - \ln n$	0
Bubble Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{6}$	0
Insertion Sort	$\approx \frac{n^2}{2}$	$\approx \frac{n^2}{8}$	0



Bubble Sort – Testes computacionais

- Arrays Ordenados

n	# Comparações	Rácio	# Atribuições
2500	2499		0
5000	4999	2.000	0
10000	9999	2.000	0
20000	19999	2.000	0

Bubble Sort – Testes computacionais

- Arrays por Ordem Inversa

n	# Comparações	Rácio	# Atribuições	Rácio
2500	3123750		9371250	
5000	12497500	4.001	37492500	4.001
10000	49995000	4.000	149985000	4.000
20000	199990000	4.000	599970000	4.000

Bubble Sort – Testes computacionais

- Arrays Aleatórios

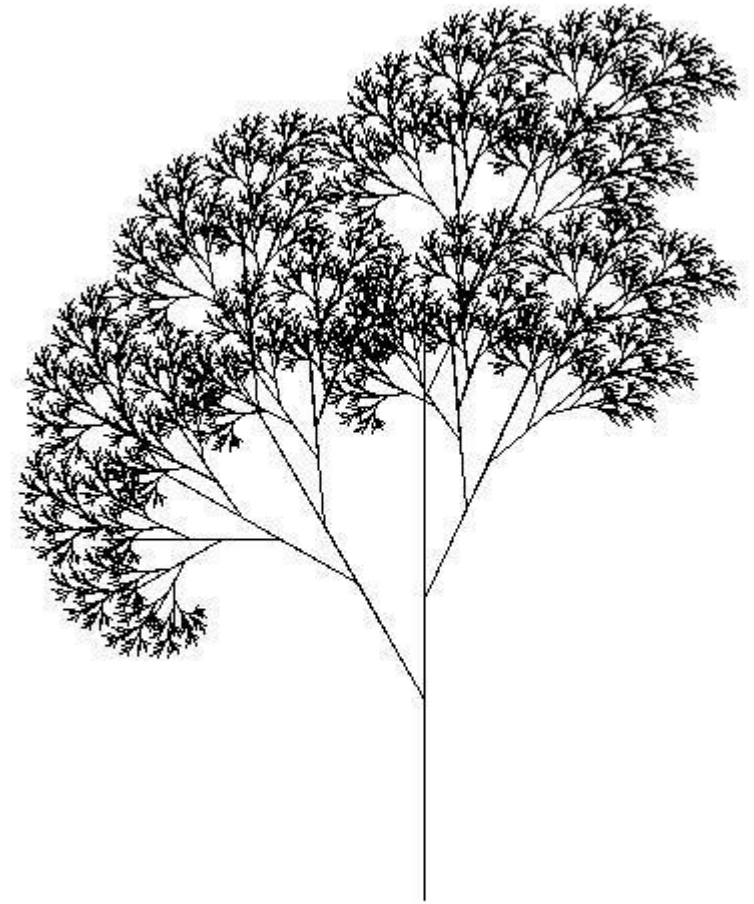
n	# Comparações	Rácio	# Atribuições	Rácio
2500	3119285		4536945	
5000	12496939	4.006	18496980	4.077
10000	49993515	4.000	74646285	4.036
20000	199969699	4.000	300555000	4.026

- Valores mais elevados do que os obtidos pela análise formal !!
- Cenário considerado é demasiado simples...

Tarefa 1

- Fazer **testes computacionais** idênticos para os **outros algoritmos**
- Ficheiros com os **dados de teste** estão disponíveis no **Moodle**

Algoritmos recursivos



[Wikipedia]

Algoritmos recursivos

- Oferecem soluções concisas e elegantes
- **MAS**, nem sempre podem ser usados – **EFICIÊNCIA**
- Podem ser um primeiro passo para o desenvolvimento de um posterior algoritmo iterativo
- **Decomposição** do problema inicial em **subproblemas mais simples** e do **mesmo tipo**
 - Desenvolvimento **Top-Down**

Estratégia de decomposição

- Identificar o(s) **caso(s) recursivo(s)**
 - Problemas do mesmo tipo
 - Diminuição da “dificuldade”
- Identificar o(s) **caso(s) de base / de paragem**
 - São atingíveis ?

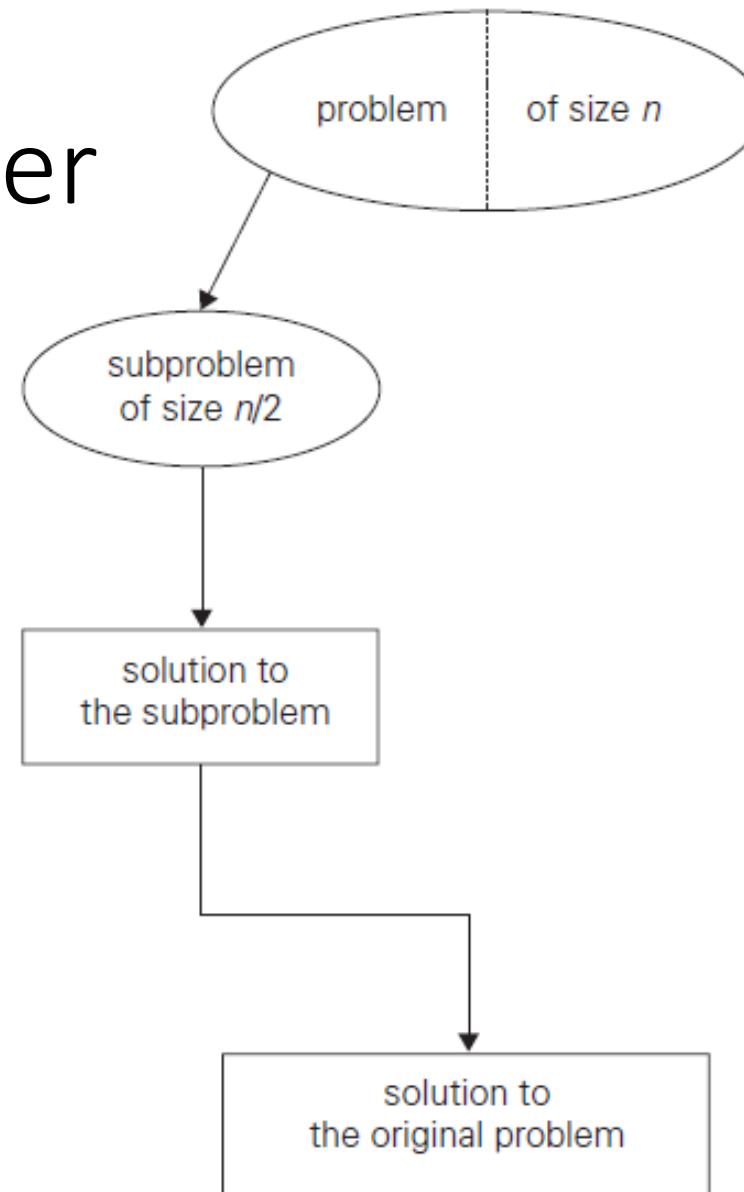
$$n! = n \times (n - 1)!$$

$$0! = 1$$

Decomposição em subproblemas

- Diminuir-para-Reinar / **Decrease-and-Conquer**
 - Resolver **1 só subproblema em cada passo** do processo recursivo
 - Lista / cadeia de chamadas recursivas
- Dividir-para-Reinar / **Divide-and-Conquer**
 - Resolver **2 ou mais subproblemas em cada passo** do processo recursivo
 - Árvore de chamadas recursivas

Decrease-and-Conquer



[Levitin]

Decrease-(by half)-and-conquer technique.

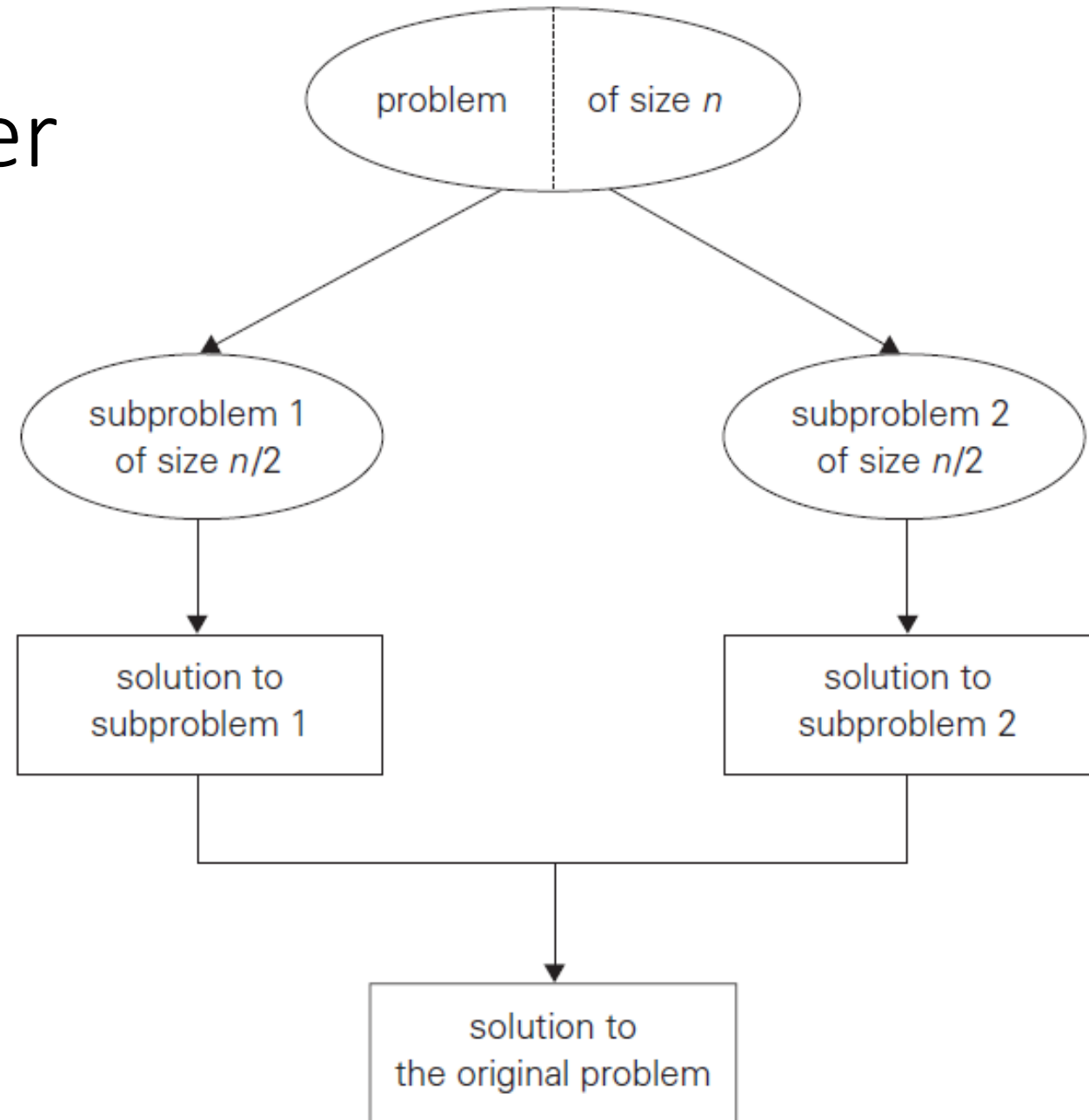
Recursividade simples

- Diminuir-para-Reinar
- Executada **1 só chamada recursiva** em cada passo
- Fatorial, mdc, travessia de um array / uma lista, procura binária, ...
- Facilmente transformável num algoritmo iterativo, usando um ciclo
- **Sugestão:** implementar alguns destes algoritmos

Tarefa 2

- Função recursiva para calcular o **mdc(a, b)**, usando o **Algoritmo de Euclides**
- Função recursiva para **procurar um valor** num array de n elementos inteiros, usando a estratégia de **Procura Sequencial**

Divide-and-Conquer



[Levitin]

Recursividade múltipla

- Dividir-para-Reinar
- Executadas 2 ou mais chamadas recursivas em cada passo
- Sucessão de Fibonacci, Combinações, ...
- Usar STACK para transformar num algoritmo iterativo
- Sugestão: implementar alguns destes algoritmos

Tarefa 3

- Função recursiva para calcular $C(n, p)$, usando a recorrência subjacente ao Triângulo de Pascal

Eficiência computacional

- **Overhead** associada a cada chamada recursiva
 - Salvaguarda do contexto
 - ...
- **MAS**, nalguns casos, também **ineficiência intrínseca**
 - Recalcular inúmeras vezes os mesmos valores
 - Repetir as mesmas operações
- A estratégia de **Programação Dinâmica** é uma **possível alternativa**, para determinados problemas

Análise Formal da Complexidade

- Identificar a **operação mais relevante**
- Obter uma **expressão recorrente** para o número de operações efetuadas
- Se possível, **desenvolver a expressão** para obter uma “fórmula fechada”
- Vamos ilustrar / aprender analisando exemplos

Calcular x^n

Calcular x^n

$$x^n = x \times x^{n-1}, n > 0$$

$$x^0 = 1$$

```
double p(double x, unsigned int n) {  
    if(n > 0) return x * p(x, n - 1);  
    return 1;  
}
```

Contar o número de multiplicações

$$M(0) = 0$$

$$M(n) = 1 + M(n - 1), n > 0$$

- Desenvolvimento telescópico – Quando parar ?

$$M(n) = 1 + M(n - 1) = 2 + M(n - 2) = \dots = k + M(n - k)$$

$$M(n) = n + M(0) = n$$

$$M(n) \in \mathcal{O}(n)$$

Tarefa 4

- Há outros algoritmos recursivos para o cálculo de potências de expoente natural
- Por exemplo:

$$x^n = x^{\left\lfloor \frac{n}{2} \right\rfloor} \times x^{\left\lceil \frac{n}{2} \right\rceil}$$

- Quais são os **casos de base** ?
- Qual é o **número de multiplicações** efetuadas ?
- **Sugestão:** implementar e comparar

Inverter a ordem dos elementos
de um array com n elementos

Inverter a ordem dos elementos

```
void inverter(int* v, int esq, int dir) {  
    if(esq < dir) {  
        trocar(&v[esq], &v[dir]);  
        inverter(v, esq + 1, dir - 1);  
    }  
}
```

Nº de trocas de elementos ?

$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = 1 + T(n - 2), n > 2$$

$$T(n) = 1 + T(n - 2) = 2 + T(n - 4) = \dots = k + T(n - 2k)$$

- Nº **par** de elementos **vs** Nº **impar** de elementos

Nº de trocas de elementos ?

$$T(n) = k + T(n - 2k)$$

- Seja o nº de elementos **par** e **maior do que 2**

$$n - 2k = 2 \Rightarrow T(n) = \frac{n - 2}{2} + T(2) = \frac{n}{2}$$

- **Tarefa:** fazer para n impar

- Verificar que **para ambos os casos:** $T(n) = \left\lfloor \frac{n}{2} \right\rfloor$

Calcular o valor de um determinante usando o Teorema de Laplace

Exemplo – Desenvolver pela 1ª coluna


$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 \times \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 4 \times \begin{vmatrix} 2 & 3 \\ 8 & 9 \end{vmatrix} + 7 \times \begin{vmatrix} 2 & 3 \\ 5 & 6 \end{vmatrix}$$

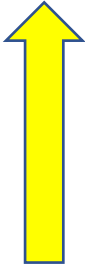
$$= 1 \times [5 \times 9 - 8 \times 6] - 4 \times [2 \times 9 - 8 \times 3] + 7 \times [2 \times 6 - 5 \times 3]$$

$$= 0$$

- **Estratégia recursiva:** decomposição em determinantes de menor dimensão

Um possível algoritmo recursivo

```
double Laplace( matriz A, unsigned int n ) {  
    ...  
    if( n == 1 ) return A[0][0];  
    sinal = -1; soma = 0;  
     for( i = 0; i < n; i++ ) {  
        aux = subMatriz(A, i, 0); // retira a 1ª coluna e a linha i  
        sinal *= -1;  
        soma += sinal * A[i][0] * Laplace(aux, n - 1);  
    }  
    return soma;  
}
```



Nº de multiplicações efetuadas ?

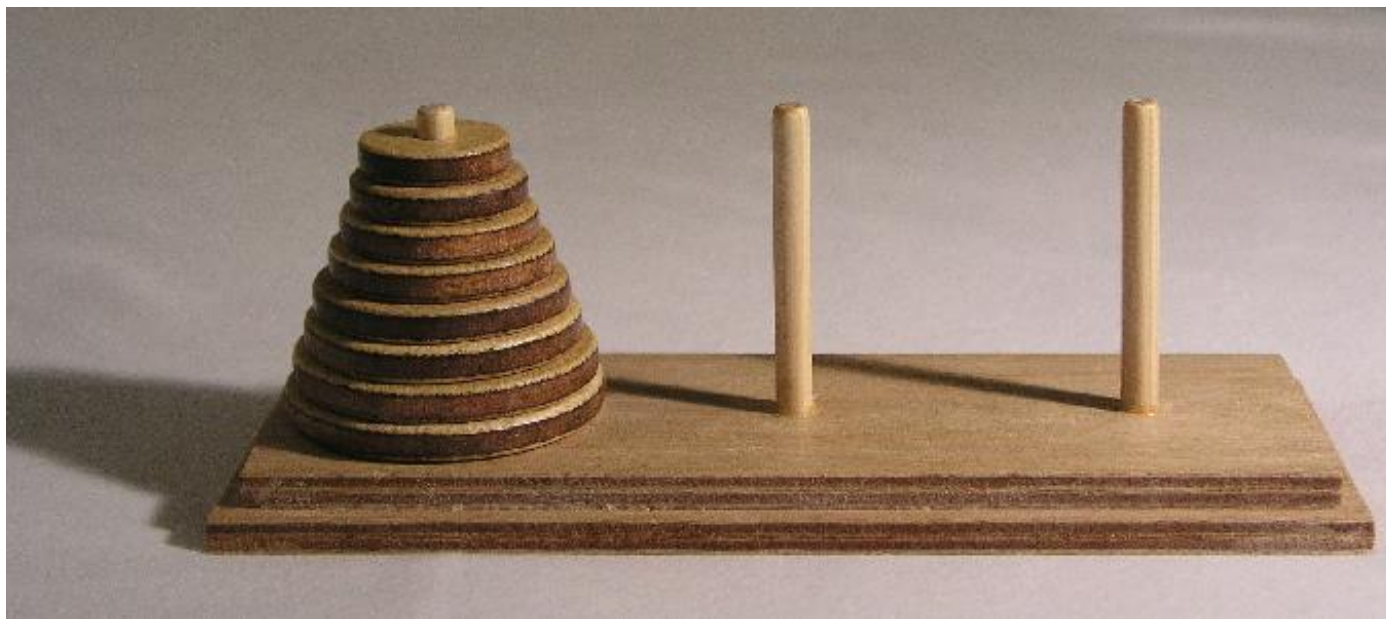
$$\bullet M(n) = \begin{cases} 0, & n = 1 \\ 2 \times n + n \times M(n - 1), & n \geq 2 \end{cases}$$

- n iterações do ciclo
- $2 \times n$ multiplicações explícitas
- n chamadas recursivas, com determinantes de menor dimensão

Nº de multiplicações efetuadas ?

$$\bullet M(n) = \begin{cases} 0, & n = 1 \\ 2 \times n + n \times M(n - 1), & n \geq 2 \end{cases}$$

- Não há uma “fórmula fechada” !!
- Verificar a rapidez com que cresce usando o **Wolfram Alpha**
- $M(n) \approx 2(e - 1)n! \Rightarrow M(n) \in O(n!)$



[Wikipedia]

As Torres de Hanói

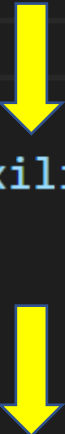
Função recursiva

```
// torresDeHanoi('A', 'B', 'C', 8);

void torresDeHanoi(char origem, char auxiliar, char destino, int n) {
    if (n == 1) {
        contadorGlobalMovs++;
        moverDisco(origem, destino); // Imprime o movimento
        return;
    }
    // Divide-and-Conquer
    torresDeHanoi(origem, destino, auxiliar, n - 1);

    contadorGlobalMovs++;
    moverDisco(origem, destino);

    torresDeHanoi(auxiliar, origem, destino, n - 1);
}
```



Tarefa – Nº de movimentos realizados ?

- $M(1) = 1$
- $M(n) = M(n-1) + 1 + M(n-1) = 1 + 2 M(n-1)$
- Fazer o desenvolvimento telescópico e obter “fórmula fechada”
- Verificar que se obtém um algoritmo EXPONENCIAL

Exercícios adicionais

Análise formal – Funções do próximo slide

- Obter uma expressão para o **resultado** de cada função
- Obter uma expressão para o nº de **chamadas recursivas efetuadas**
- Confirmar os resultados obtidos com o Wolfram Alpha

<https://www.wolframalpha.com/>

Resultado ? – Nº de chamadas recursivas ?

unsigned int

```
r1(unsigned int n) {  
    if(n == 0) return 0;  
    return 1 + r1(n - 1);  
}
```

unsigned int

```
r2(unsigned int n) {  
    if(n == 0) return 0;  
    if(n == 1) return 1;  
    return n + r2(n - 2);  
}
```

unsigned int

```
r3(unsigned int n) {  
    if(n == 0) return 0;  
    return 1 + 2 * r3(n - 1);  
}
```

unsigned int

```
r4(unsigned int n) {  
    if(n == 0) return 0;  
    return 1 + r4(n - 1) + r4(n - 1);  
}
```

Sugestões de leitura

Sugestões de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
 - Capítulo 1: secções 1.5 e 1.6
- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
 - Capítulo 2: secções 2.4 e 2.5
 - Apêndice B