

Análise da Complexidade de Algoritmos Recursivos IV

Joaquim Madeira

22/04/2021

Sumário

- Recap
- Ordenação por fusão: o Algoritmo Mergesort – Análise da Complexidade
- Ordenação por partição: o Algoritmo Quicksort
- Sugestões de leitura

Let's
RECAP

Recapitulação

The Master Theorem

- Dada uma recorrência, para $n = b^k$, $k \geq 1$

$$T(1) = c \quad \text{e} \quad T(n) = a T(n / b) + f(n)$$

com $a \geq 1$, $b \geq 2$, $c \geq 0$

- Se $f(n)$ em $\Theta(n^d)$, em que $d \geq 0$, então

$$T(n) \text{ em } \Theta(n^d), \text{ se } a < b^d$$

$$T(n) \text{ em } \Theta(n^d \log n), \text{ se } a = b^d$$

$$T(n) \text{ em } \Theta(n^{\log_b a}), \text{ se } a > b^d$$

The Smoothness Rule

- Seja $T(n)$ uma função eventualmente **não decrescente**
- Seja $g(n)$ uma função suave (“**smooth function**”)
- Se $T(n)$ em $\Theta(g(n))$ para valores de n que sejam **potências de b** , $b \geq 2$
- Então **$T(n)$ em $\Theta(f(n))$, para qualquer n**
- Resultados análogos para **$O(n)$ e $\Omega(n)$!!**
- **Boas notícias !!**

Multiplicar números inteiros muito longos

- Como fazer ?
 - E.g., mais do que 100 algarismos
- Muito maiores do que o maior inteiro representável !
- Usar o algoritmo que aprendemos na escola ?
- Divide-and-Conquer ?

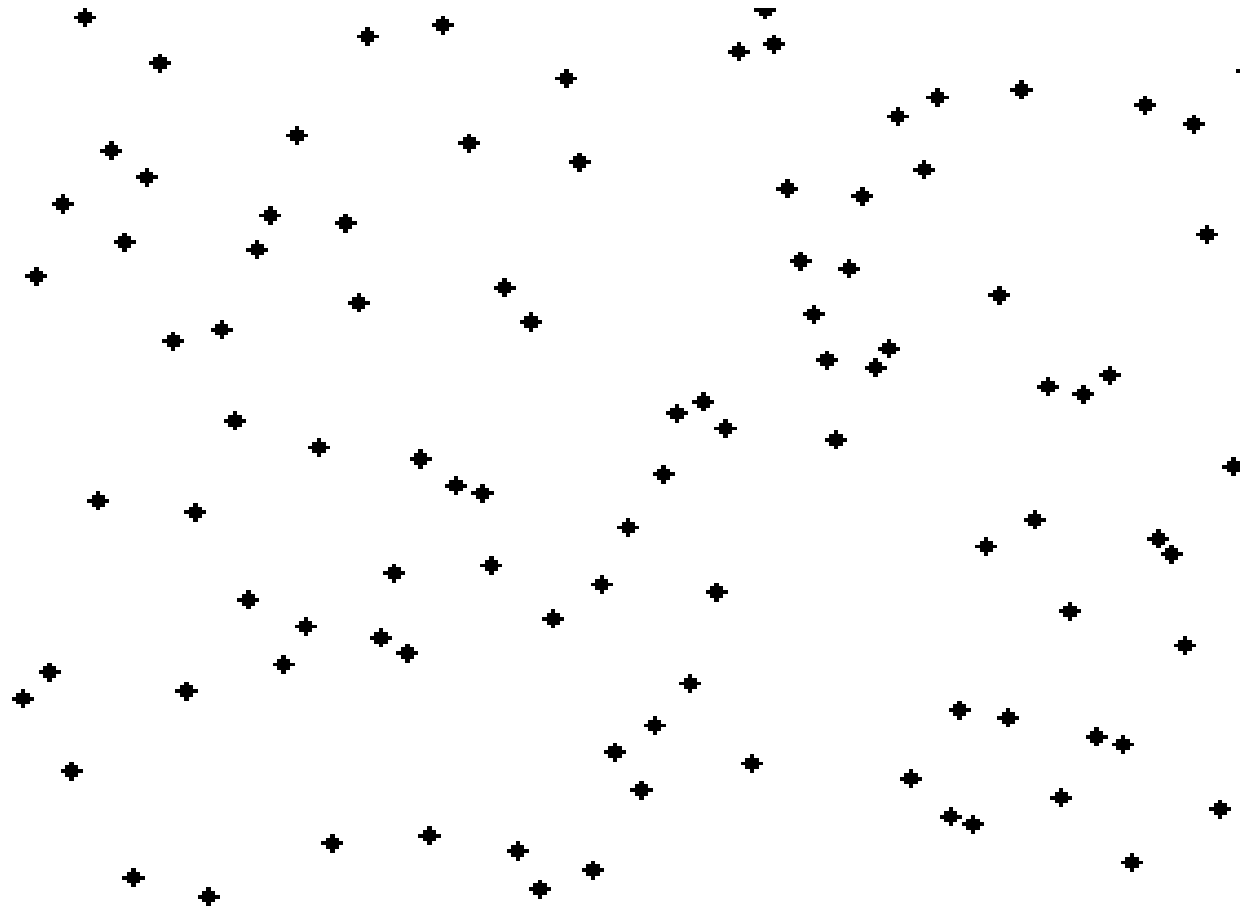
Quantas multiplicações algarismo a algarismo?

- Algoritmo da escola primária : $\Theta(n^2)$
- Multiplicação por **partição recursiva** – 1ª ideia :
 - $M(n) = 4 M(n / 2)$ $\Theta(n^2)$
- Multiplicação por **partição recursiva** – Alg. de Karatsuba :
 - $M(n) = 3 M(n / 2)$
$$\Theta(n^{\log_2 3}) = \Theta(n^{1.585}) !!$$

Mergesort

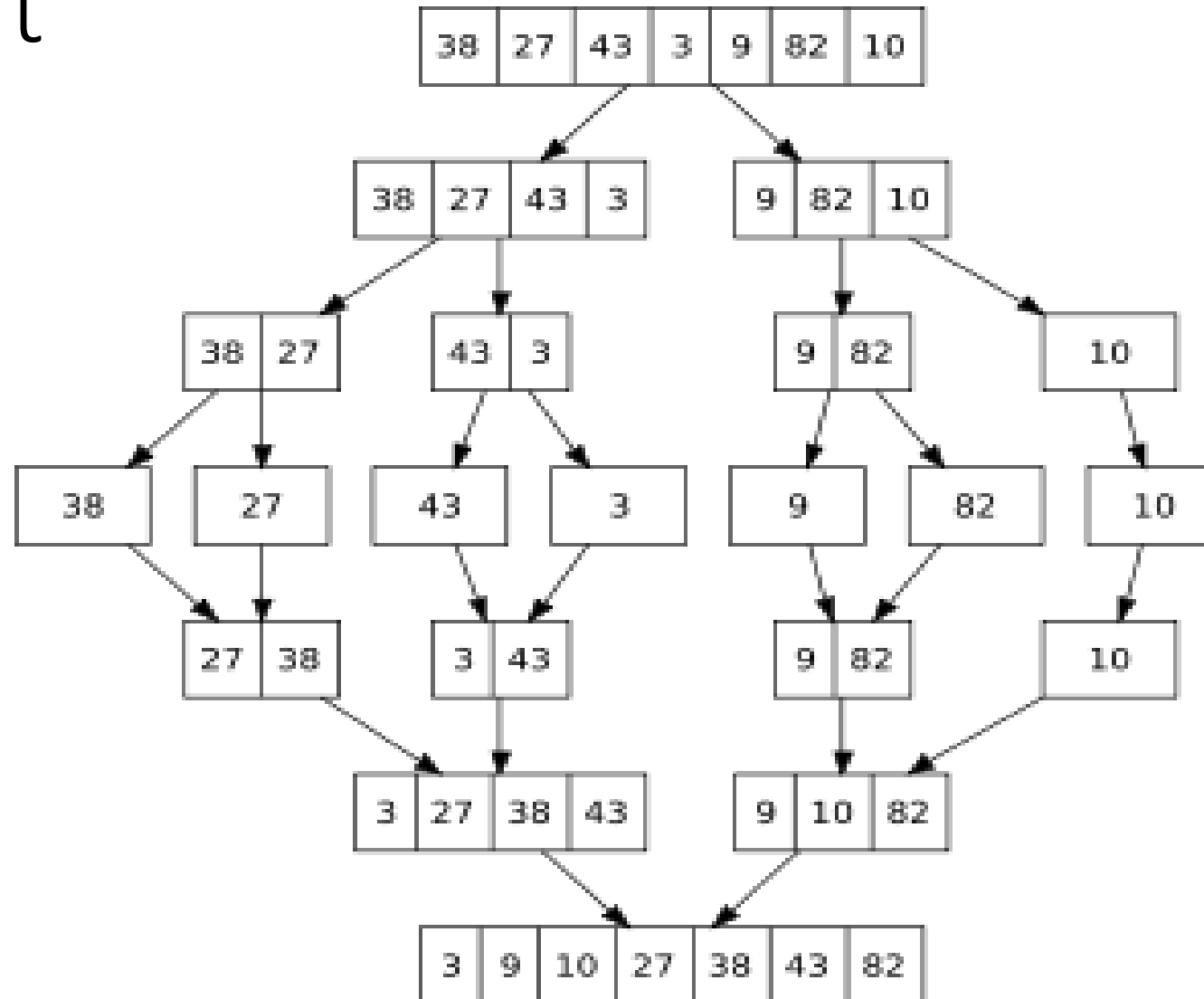
– Ordenação por Fusão

Mergesort



[Wikipedia]

Mergesort



SUBDIVISÃO

FUSÃO

Tarefa: associar a cada seta um rótulo que identifica a sequência pela qual as chamadas são executadas

[Wikipedia]


Mergesort

```
// mergeSort(A, tmpA, 0, n - 1);

void mergeSort(int* A, int* tmpA, int left, int right) {
    // Mais do que 1 elemento ?


    if (left < right) {
        int center = (left + right) / 2;

        mergeSort(A, tmpA, left, center);
        mergeSort(A, tmpA, center + 1, right);
        merge(A, tmpA, left, center + 1, right);
    }
}
```



Mergesort

```
void merge(int* A, int* tmpA, int lPos, int rPos, int rEnd) {  
    int lEnd = rPos - 1;  
    int tmpPos = lPos;  
    int nElements = rEnd - lPos + 1;  
  
    // COMPARAR O 1o ELEMENTO DE CADA METADE  
    // E COPIAR ORDENADAMENTE PARA O ARRAY TEMPORÁRIO  
  
    while (lPos <= lEnd && rPos <= rEnd) {  
        if (A[lPos] <= A[rPos])  
            tmpA[tmpPos++] = A[lPos++];  
        else  
            tmpA[tmpPos++] = A[rPos++];  
    }  
}
```



Mergesort

```
// SOBRA, PELO MENOS, 1 ELEMENTO NUMA DAS METADES

while (lPos <= lEnd) { ...
}

while (rPos < rEnd) { ...
}

// COPIAR DE VOLTA PARA O ARRAY ORIGINAL

for (int i = 0; i < nElements; i++, rEnd--) {
    A[rEnd] = tmpA[rEnd];
}
}
```

Eficiência

- Todas as **comparações** são feitas pela função de fusão
- $C_{\text{merge}}(n)$: nº de **comparações** efetuadas para fundir 2 sub-arrays ordenados, usando um **array auxiliar**
- Caso particular : $n = 2^k$

$$C(1) = 0$$

$$C(n) = 2 \times C(n / 2) + C_{\text{merge}}(n)$$

- $C_{\text{merge}}(n) = ?$

Eficiência – $C_{\text{merge}}(n)$ – Melhor Caso

- Todos elementos de um dos sub-arrays são copiados primeiro
- Apenas $n / 2$ comparações para fazer isso !!
- $C(n) = 2 \times C(n / 2) + n / 2$
- Teorema Mestre : $\Theta(n \log n)$
- Construir um exemplo !

Mergesort

0	1	2	3	4
7	6	4	3	2

Mergesort

0	1	2	3	4
7	6	4	3	2

7	6	4
----------	----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

7	6
----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

7

Mergesort

0	1	2	3	4
7	6	4	3	2

7	6
----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

7	6
----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

7
6

Mergesort

0	1	2	3	4
7	6	4	3	2

6	7
----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

6	7	4
----------	----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

6	7	4
----------	----------	----------

Mergesort

0	1	2	3	4
7	6	4	3	2

6	7
4	

Mergesort

0	1	2	3	4
7	6	4	3	2

	7
4	6

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7
---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

3	2
---	---

4	6	7
---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

3

4	6	7
---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

3	2
---	---

4	6	7
---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

3	2
---	---

4	6	7
---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7	3
			2

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7	2	3
---	---	---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7	2	3
---	---	---	---	---

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7
2		

3

Mergesort

0	1	2	3	4
7	6	4	3	2

4	6	7
2	3	

Mergesort

0	1	2	3	4
7	6	4	3	2

	6	7
2	3	4

Mergesort

0	1	2	3	4
7	6	4	3	2

		7		
2	3	4	6	

Mergesort

0	1	2	3	4
7	6	4	3	2

2	3	4	6	7
----------	----------	----------	----------	----------

Eficiência – $C_{\text{merge}}(n)$ – Pior Caso

- Os elementos de um dos sub-arrays são copiados de modo intercalado, um a um !
- Necessárias $(n - 1)$ comparações !
- $C(n) = 2 \times C(n / 2) + (n - 1)$
- Teorema Mestre : $\Theta(n \log n)$
- Construir um exemplo !



Eficiência – $C_{\text{merge}}(n)$ – Caso Médio

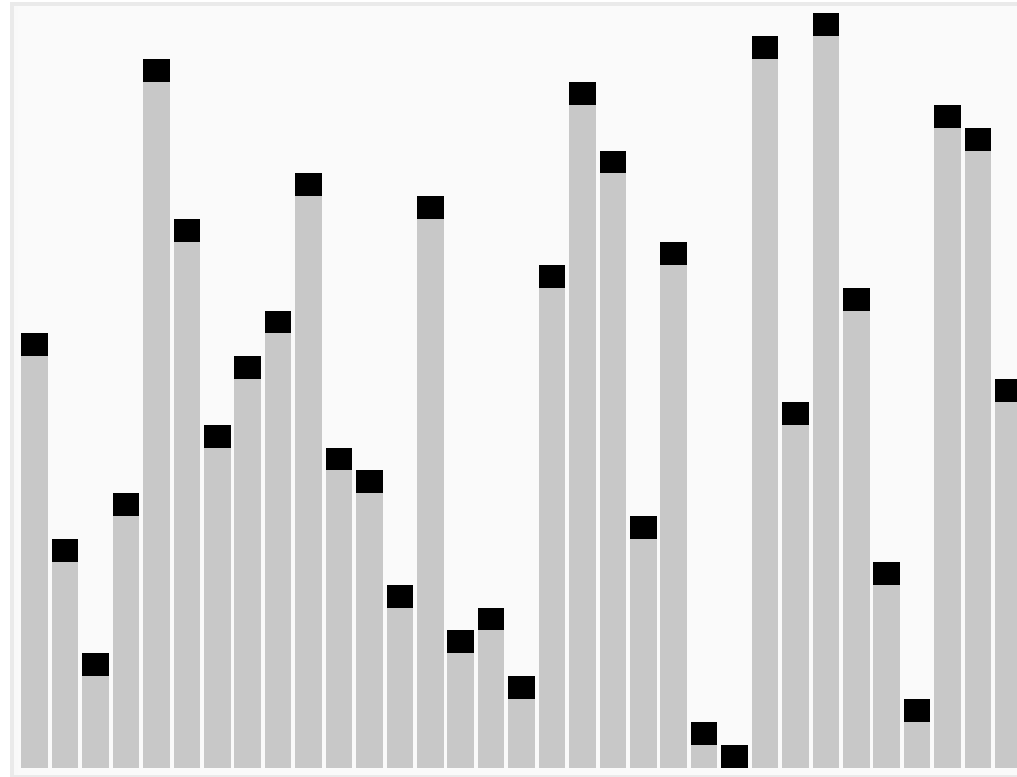
- Podemos assumir que ocorre o nº médio de comparações
- $\sim 3 \times n / 4$ comparações
- $C(n) = 2 \times C(n / 2) + 3 \times n / 4$
- Teorema Mestre : $\Theta(n \log n)$
- Construir um exemplo !



Quicksort

– Ordenação por Partição

Quicksort



[Wikipedia]

Quicksort

- Ordenar o array de modo recursivo, **sem usar memória adicional**
- **Particionar** o conjunto de elementos, **trocando de posição**, se necessário
- Com base no **valor** de um elemento **pivot**

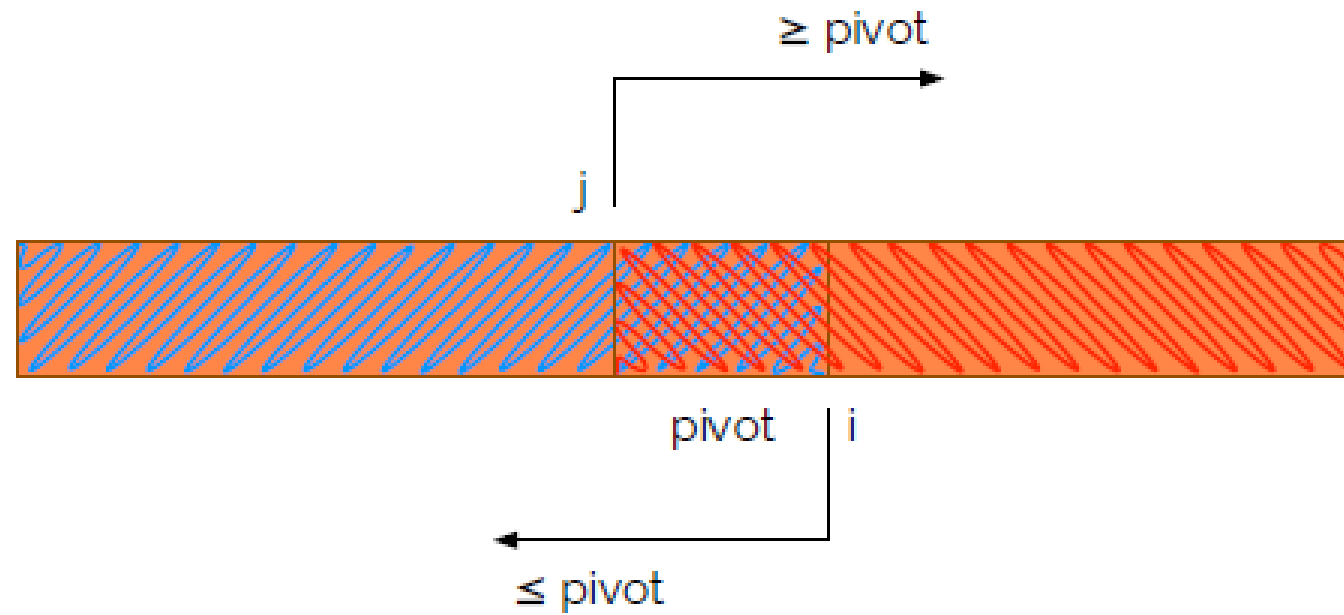
Quicksort

- Escolher o **valor** do element **pivot**
- **Particionar** o array
- Elementos da 1ª partição são **menores ou iguais** do que o pivot
- Elementos da 2ª partição são **maiores ou iguais** do que o pivot
- Ordenar de modo **recursivo** a 1ª partição e a 2ª partição

Questões

- Como **esolher o pivot** ?
 - O elemento do **meio**? O **1º** elemento? O **último** elemento?
 - O elemento **mediano** dos 3 anteriores?
 - Um elemento escolhido de modo **aleatório**?
- Como **particionar** ?
- Atenção : pode surgir uma terceira **partição central**, cujos elementos têm o valor do pivot !

Partições



[Rui Lopes]

Quicksort

0	1	2	3	4
7	2	6	4	3

Quicksort – Pivot = elemento do “meio”

0	1	2	3	4
7	2	6	4	3

7	2	6	4	3
---	---	---	---	---

Quicksort

0	1	2	3	4
7	2	6	4	3
i				
7	2	6	4	3

Quicksort

0	1	2	3	4
7	2	6	4	3
i				j
7	2	6	4	3

Quicksort

0	1	2	3	4
7	2	6	4	3
i		j		
3	2	6	4	7

Quicksort

0	1	2	3	4
7	2	6	4	3
		i	j	
3	2	6	4	7

Quicksort

0	1	2	3	4
7	2	6	4	3
		i	j	
3	2	6	4	7

Quicksort

0	1	2	3	4
7	2	6	4	3
		j	i	
3	2	4	6	7

Quicksort

0	1	2	3	4
7	2	6	4	3
			6	7
3	2	4		

Quicksort

0	1	2	3	4
7	2	6	4	3

i				6	7
	3	2	4		

Quicksort

0	1	2	3	4
7	2	6	4	3

i		j		
3	2	4	6	7

Quicksort

0	1	2	3	4
7	2	6	4	3

i	j		6	7
3	2	4		

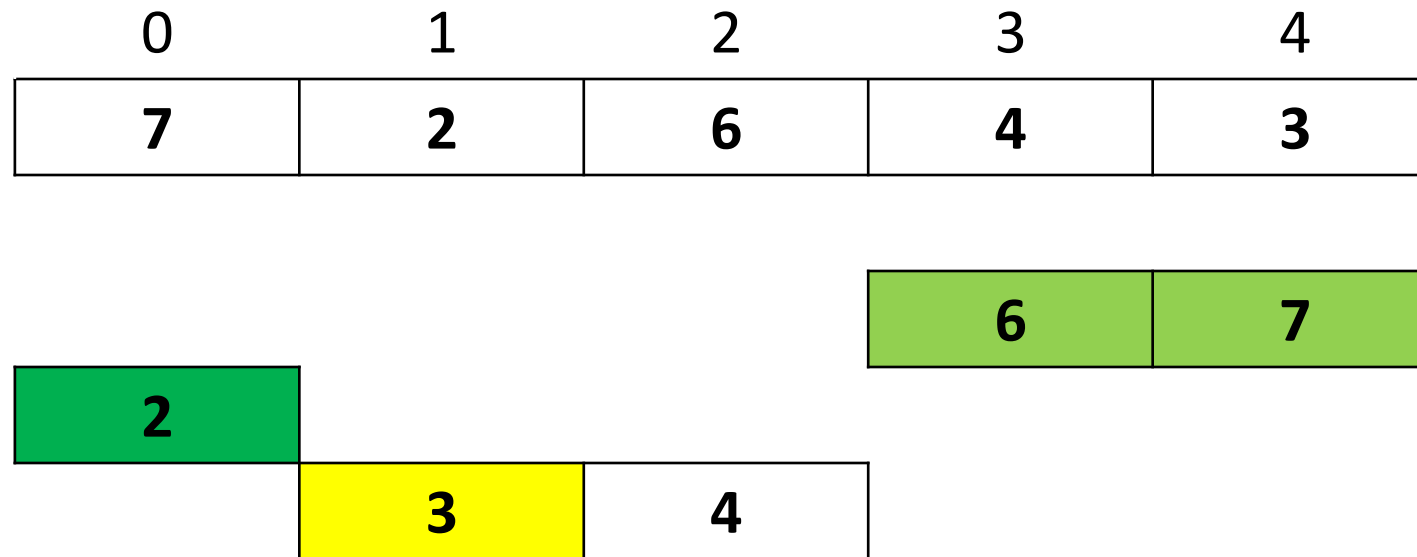
Quicksort

0	1	2	3	4
7	2	6	4	3

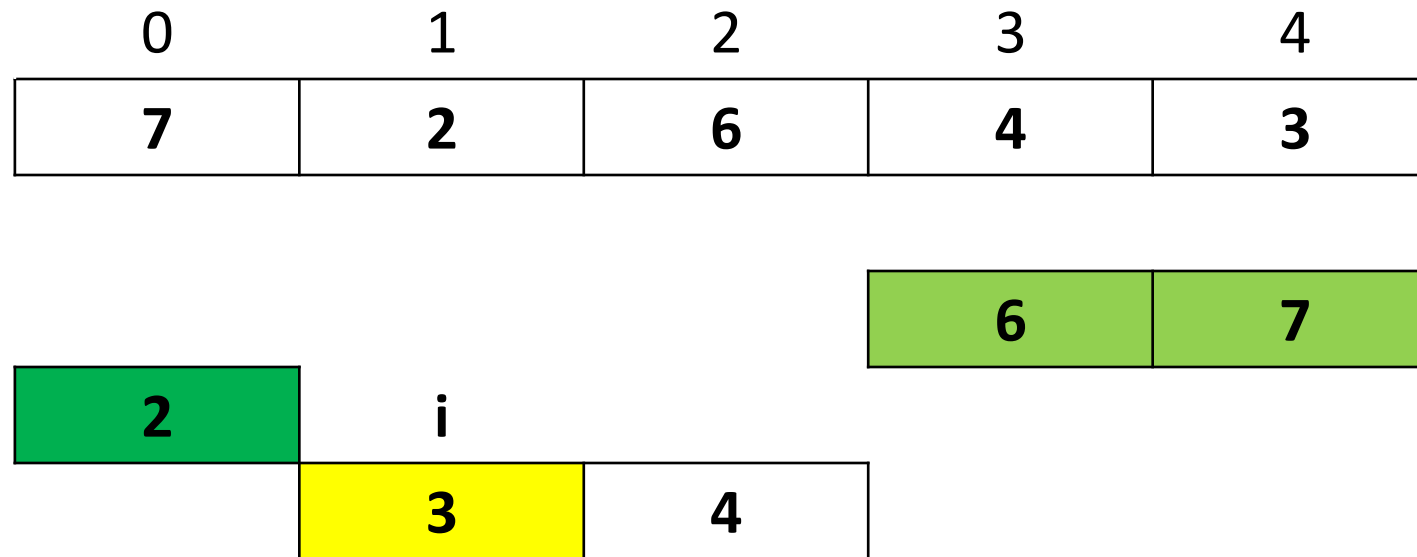
j	i		
2	3	4	

			6	7
--	--	--	---	---

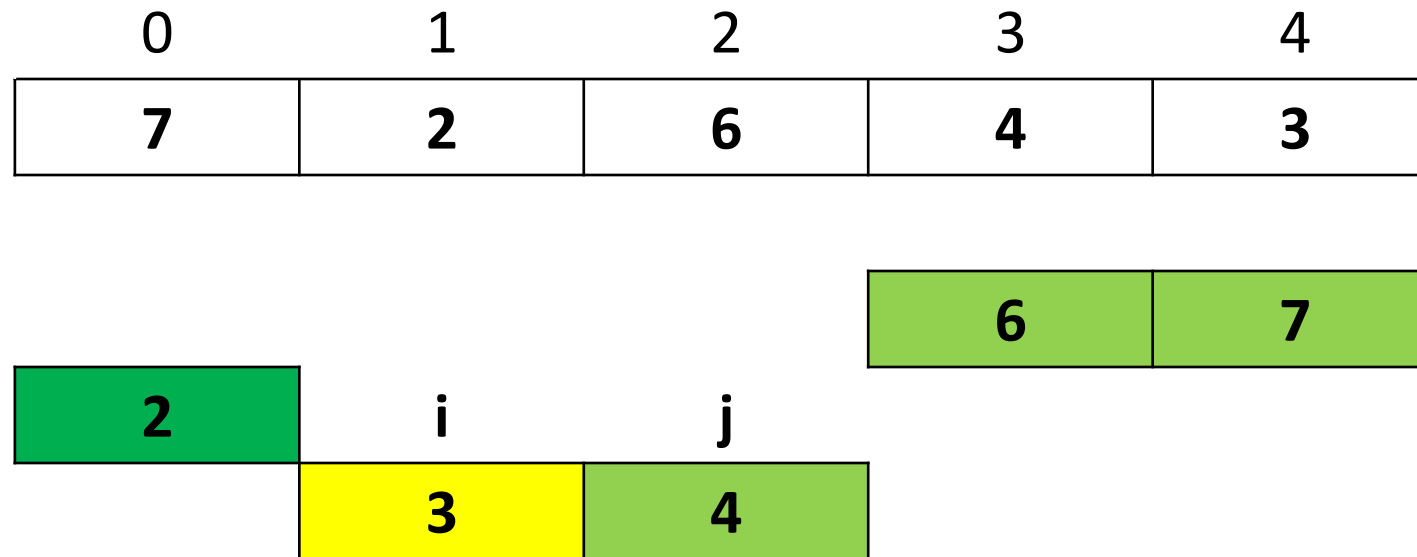
Quicksort



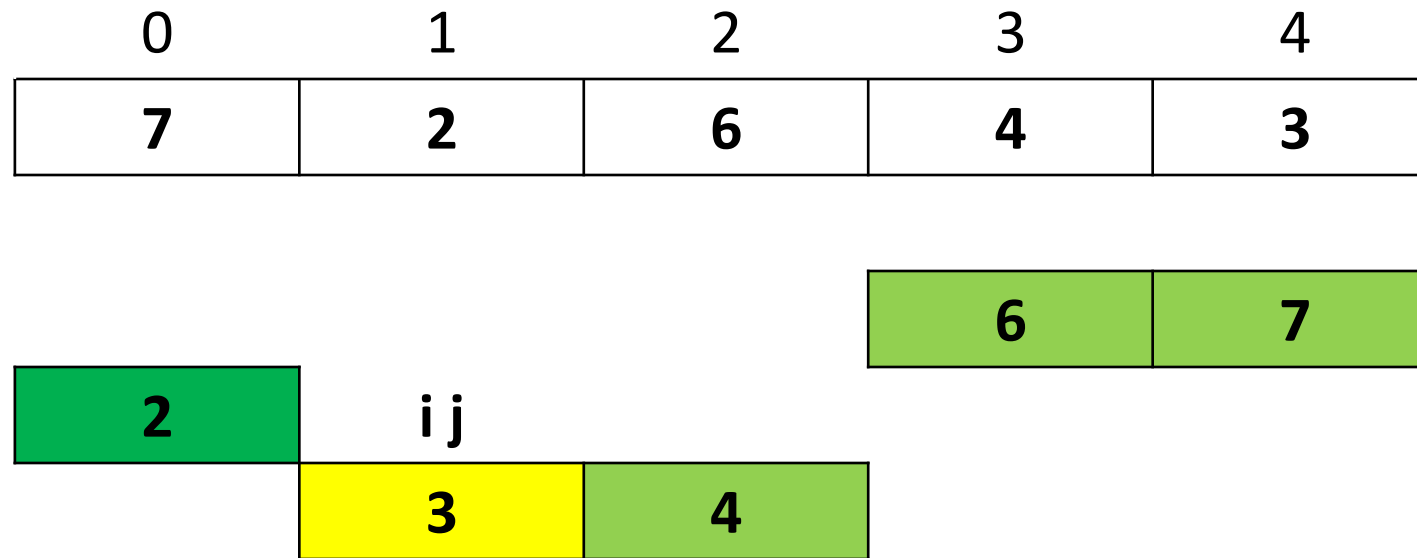
Quicksort



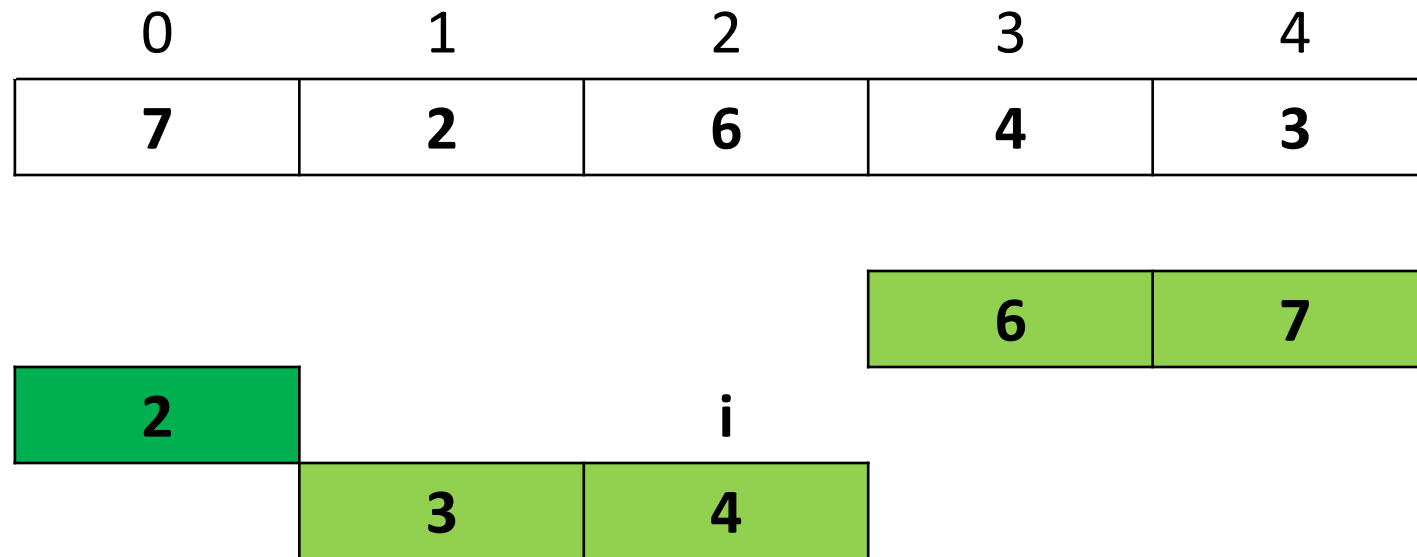
Quicksort



Quicksort



Quicksort



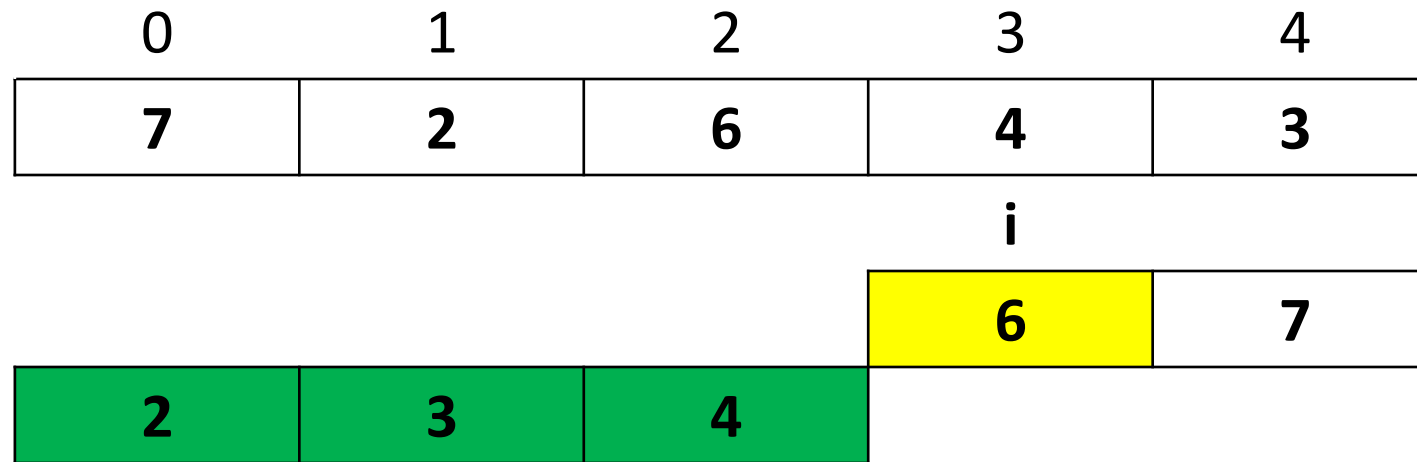
Quicksort

0	1	2	3	4
7	2	6	4	3
			6	7
2	3	4		

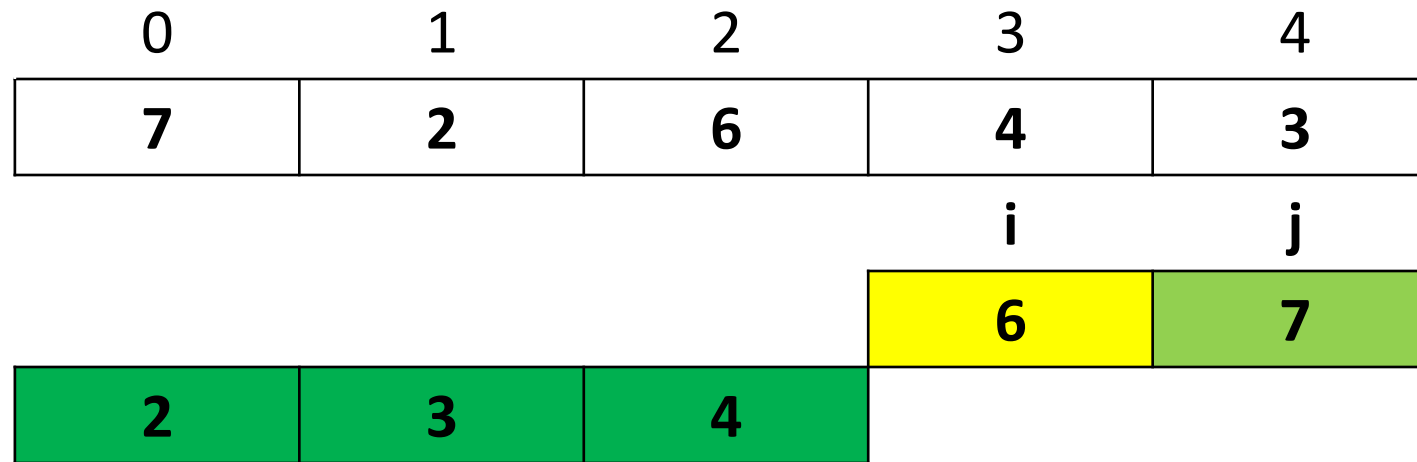
Quicksort

0	1	2	3	4
7	2	6	4	3
			6	7
2	3	4		

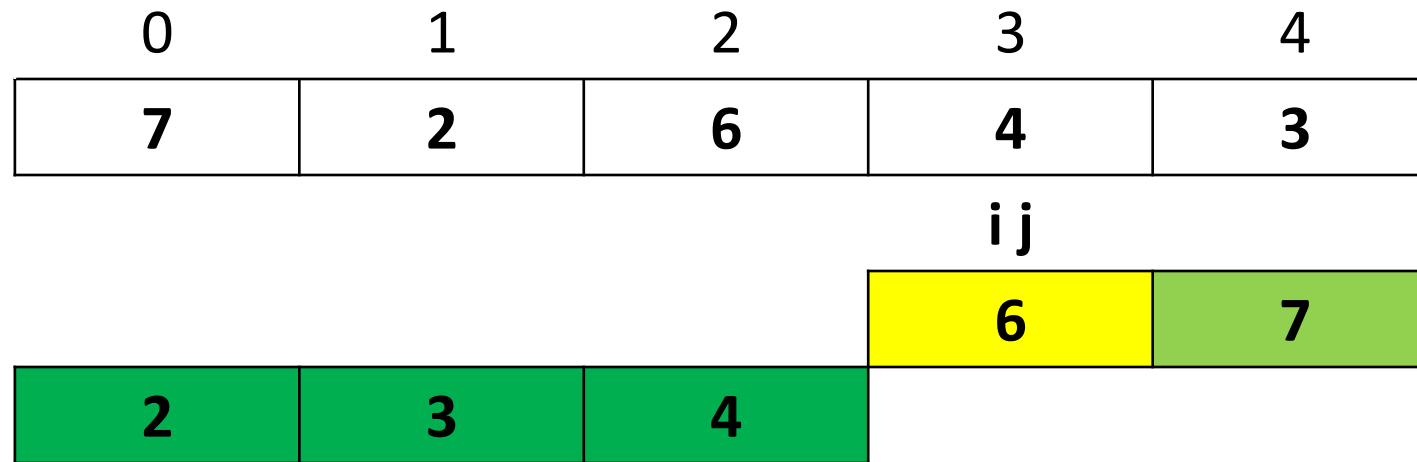
Quicksort



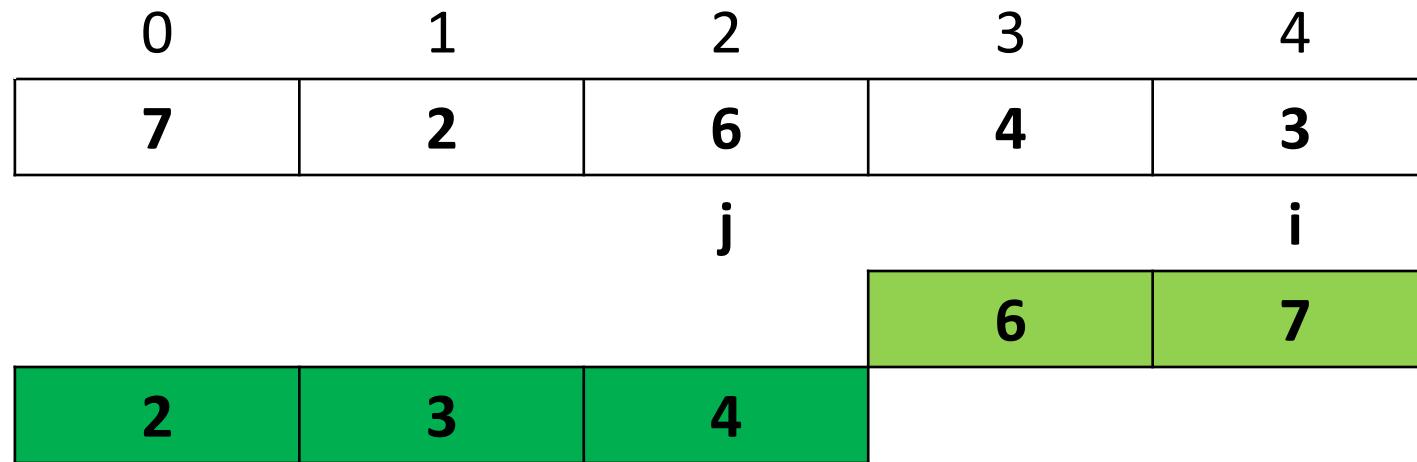
Quicksort



Quicksort



Quicksort



Quicksort

0	1	2	3	4
7	2	6	4	3
2	3	4	6	7

Quicksort

```
void quicksort(int* A, int left, int right) {  
    // Casos de base  
    if (left >= right) return;  
  
    // Caso recursivo  
    // FASE DE PARTIÇÃO  
    int pivot = (left + right) / 2;  
    int i = left;  
    int j = right;  
  
    do {  
        while (A[i] < A[pivot]) i++;  
        while (A[j] > A[pivot]) j--;  
  
        if (i <= j) {  
            trocar(&A[i], &A[j]);  
            i++;  
            j--;  
        }  
    } while (i <= j);  
  
    // Chamadas recursivas  
    quicksort(A, left, j);  
    quicksort(A, i, right);  
}
```



Tarefa 1

0	1	2	3	4	5
6	5	4	3	2	1

- Ordenar !
- Usar como pivot o elemento do “meio” !
- Qual é o valor do elemento escolhido como pivot ?
- O que tem este exemplo de particular ?

Exemplo – Pivot é o 1º elemento

- Fase de partição

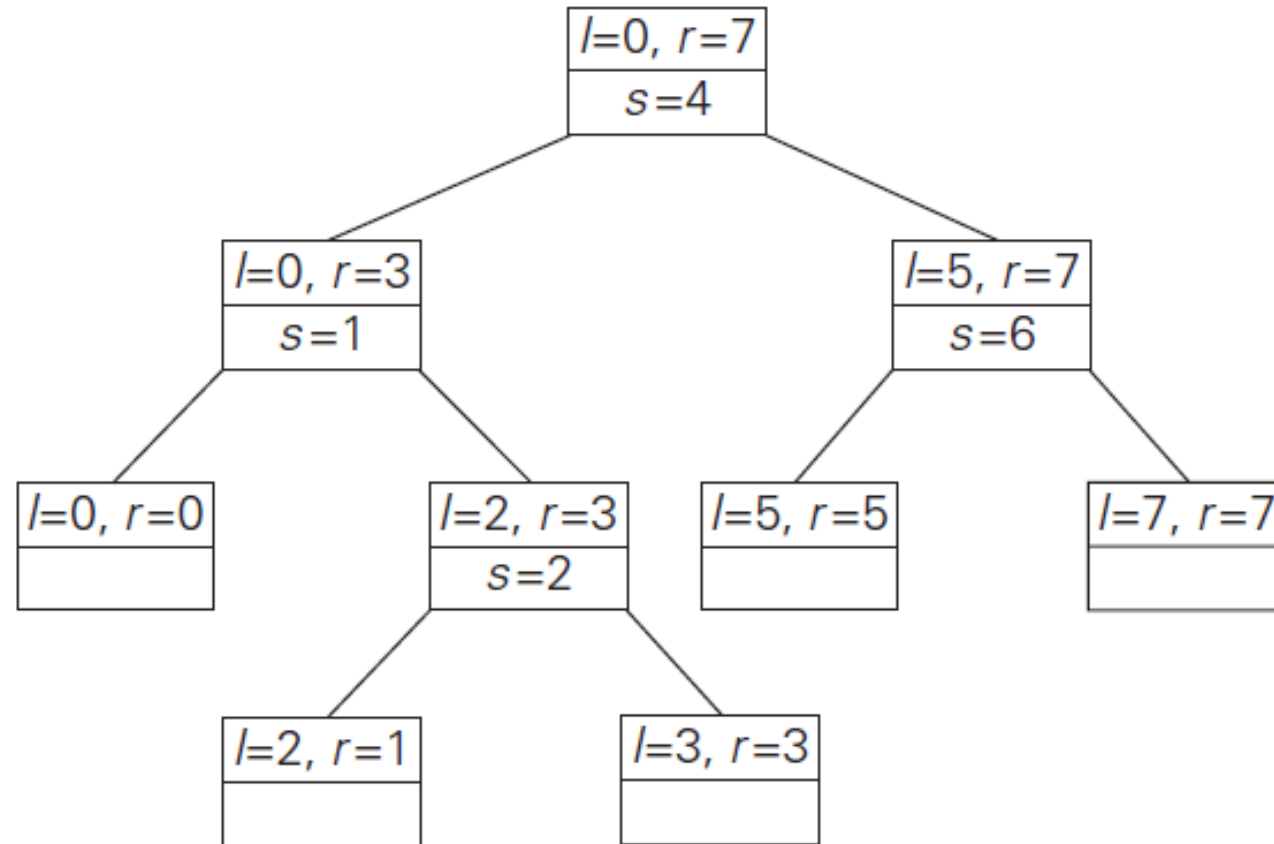
0	1	2	3	4	5	6	7
	<i>i</i>						<i>j</i>
5	3	1	9	8	2	4	7
5	3	1	9	8	2	4	7
5	3	1	4	8	2	9	7
5	3	1	4	8	2	9	7
5	3	1	4	2	8	9	7
5	3	1	4	2	8	9	7
2	3	1	4	5	8	9	7

[Levitin]

- Concluir !!

Exemplo – Pivot é o 1º elemento

- Chamadas recursivas



[Levitin]

- s é a posição do pivot após a partição

Eficiência

- Todas as **comparações** são feitas na fase de partição !!
- Qual é a **recorrência** ?
- O **caso geral** é mais difícil de desenvolver e analisar !!
- **$O(n \log n)$** para o **melhor caso** e o **caso médio**
- MAS **$O(n^2)$** para o **pior caso** !!
 - Muito raro, se escolhermos “bem” o pivot !!

Sugestões de leitura

Sugestões de leitura

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
 - Capítulo 5: secção 5.1
 - Capítulo 5: secção 5.2
 - Apêndice B