Análise da Complexidade de Algoritmos Recursivos V

Joaquim Madeira

14/04/2020

Sumário

- Guião 5 – Revisão / Análise
- Ordenação por partição: o Algoritmo Quicksort – Análise da Complexidade
- Seleção do k-ésimo elemento: o Algoritmo Quickselect
- Sugestões de leitura

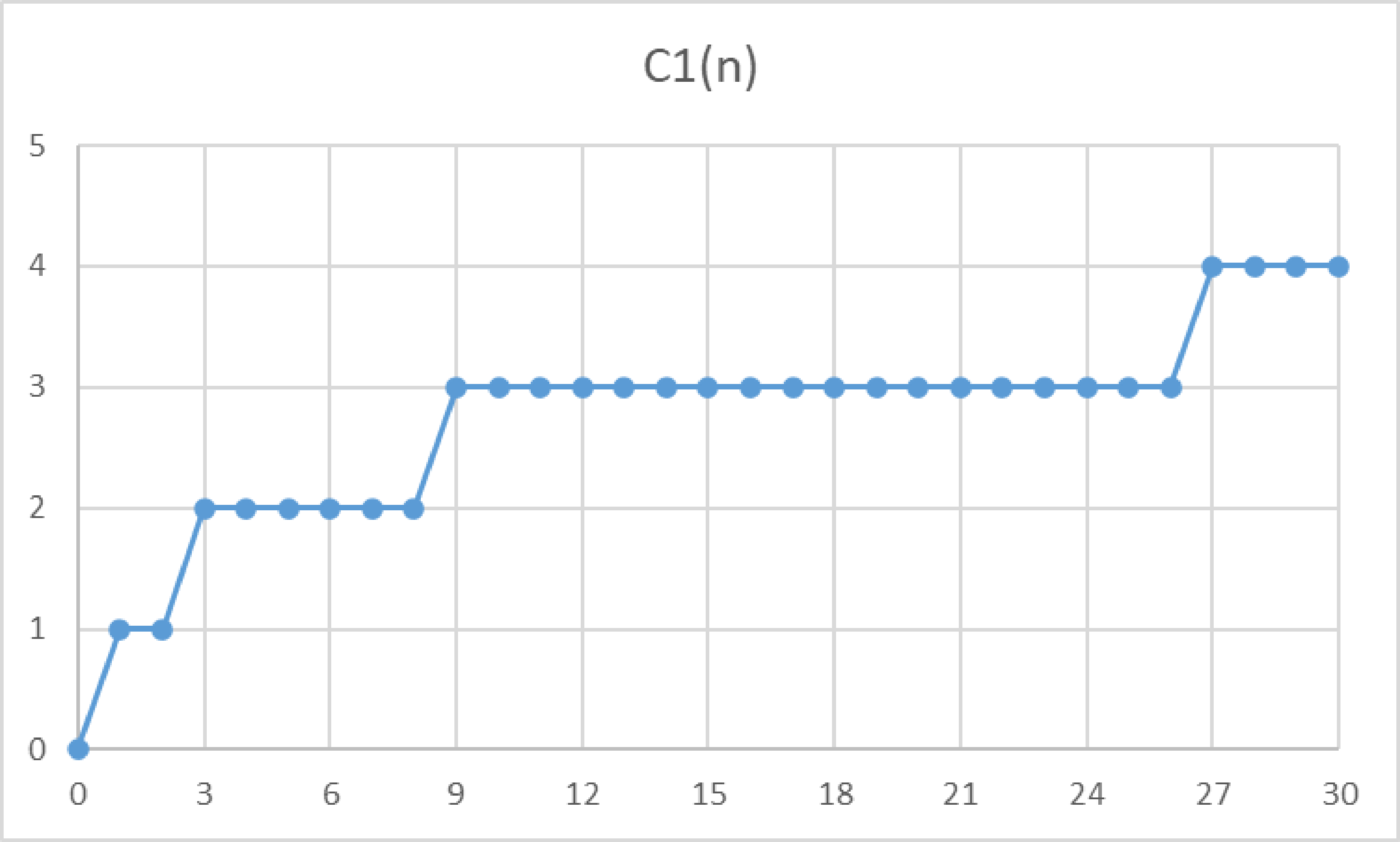
Guião 5 – Revisão / Análise

Chamadas recursivas ?

$$\mathrm{T_1(n)} = \begin{cases} 0, \text{se } n = 0 \\ \mathrm{T_1}\left(\left\lfloor \frac{\mathrm{n}}{3} \right\rfloor\right) + \mathrm{n}, \text{se } n > 0 \end{cases}$$

$$\color{red}{\mathrm{C_1}}\mathrm{(n)} = \begin{cases} 0, \text{se } n = 0 \\ \color{red}{1} + \color{red}{\mathrm{C_1}}\left(\left\lfloor \frac{\mathrm{n}}{3} \right\rfloor\right), \text{se } n > 0 \end{cases}$$

Resultados experimentais



$$C_1(n) = 1 + \lfloor \log_3 n \rfloor$$

$$C_1(n) \in \mathcal{O}(\log_3 n)$$

Desenvolvimento telescópico

$$C(n) = 1 + C\left(\left\lfloor\frac{n}{3}\right\rfloor\right) = 1 + 1 + C\left(\left\lfloor\frac{n}{9}\right\rfloor\right) = \cdots = k + C\left(\left\lfloor\frac{n}{3^k}\right\rfloor\right)$$

- Quando paramos o desenvolvimento?

$$\left\lfloor\frac{n}{3^k}\right\rfloor = 0 \text{ e } \left\lfloor\frac{n}{3^{k-1}}\right\rfloor = 1 \text{ logo } k = 1 + \lfloor\log_3 n\rfloor$$

$$C(n) = 1 + \lfloor\log_3 n\rfloor + C(0) = 1 + \lfloor\log_3 n\rfloor$$

$$C(n) \in \mathcal{O}(\log_3 n)$$

Sugestão

- Considerar n representado na base 3

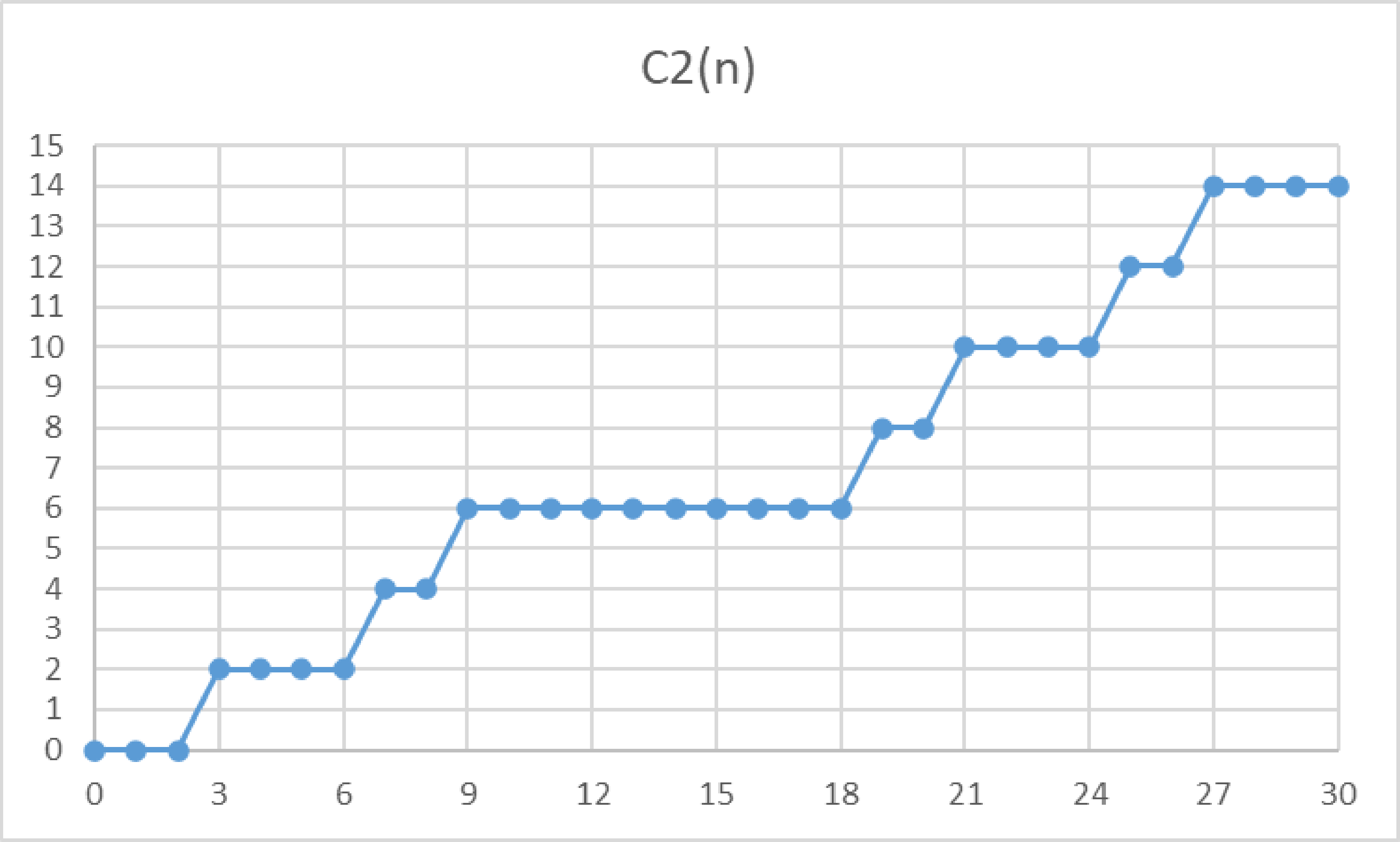
- O que acontece ao executarmos o algoritmo?

Chamadas recursivas ?

$$\mathrm{T}_2(\mathrm{n}) = \begin{cases} n, \text{se } n = 0, 1, 2 \\ \mathrm{T}_2\left(\left\lfloor \frac{\mathrm{n}}{3} \right\rfloor\right) + \mathrm{T}_2\left(\left\lceil \frac{\mathrm{n}}{3} \right\rceil\right) + \mathrm{n}, \text{se } n > 2 \end{cases}$$

$$\mathrm{C}_2(\mathrm{n}) = \begin{cases} 0, \text{se } n = 0, 1, 2 \\ 2 + \mathrm{C}_2\left(\left\lfloor \frac{\mathrm{n}}{3} \right\rfloor\right) + \mathrm{C}_2\left(\left\lceil \frac{\mathrm{n}}{3} \right\rceil\right), \text{se } n > 2 \end{cases}$$

Resultados experimentais



$$C_2(n) \in \mathcal{O}(n)$$

Caso particular: $\boldsymbol{n = 3^k}, k = \log_3 n$

- Obtém-se a recorrência simplificada:

$$C(n) = 2 + 2\,C\left(\frac{n}{3}\right)$$

- Desenvolvimento telescópico

$$C(n) = 2 + 2\,C\left(\frac{n}{3}\right) = 2 + 4 + 4\,C\left(\frac{n}{9}\right) = \cdots$$

$$= \ 2 + \cdots + \ 2^k + 2^k C\left(\frac{n}{3^k}\right)$$

$$= \sum_{i=1}^{k} 2^i + \ 2^k C(1) = \sum_{i=1}^{k} 2^i$$

Caso particular: $\boldsymbol{n = 3^k}, k = \log_3 n$

$$C(n) = 2 + 2\,C\left(\frac{n}{3}\right)$$

$$C(n) = \sum_{i=1}^{k} 2^i = 2 \times \sum_{i=0}^{k-1} 2^i = 2 \times (2^k - 1)$$

$$= 2 \times \left(2^{\log_3 n} - 1\right) = 2 \times \left({3^{\log_3 2}}^{\log_3 n} - 1\right)$$

$$= 2 \times \left(n^{\log_3 2} - 1\right)$$

Caso particular: $\boldsymbol{n = 3^k}, k = \log_3 n$

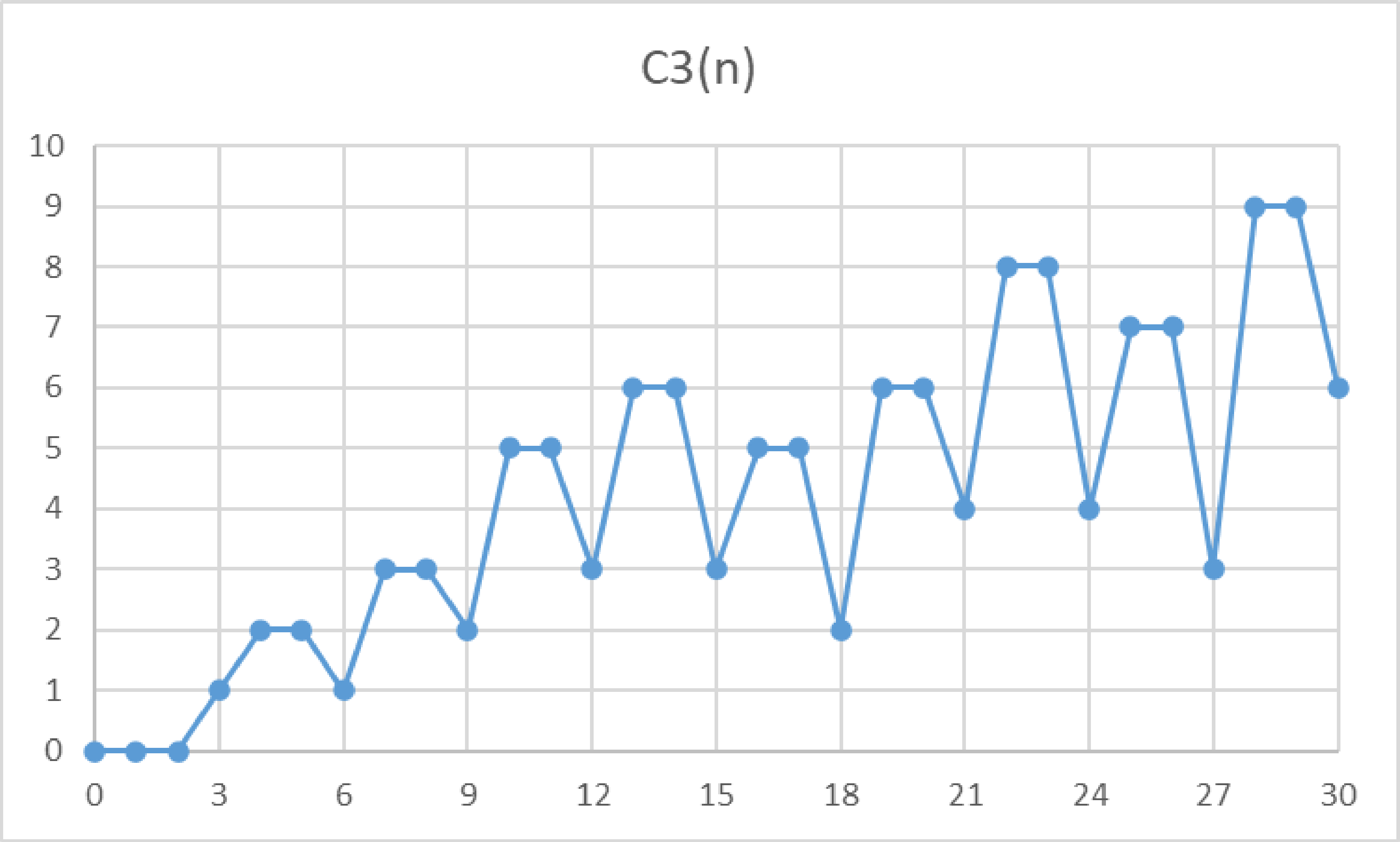
- De acordo com os valores da tabela, para estes casos particulares
- $C_2(n) \in \mathcal{O}(n^{\log_3 2})$, para todo o n (Smoothness Rule)
- O mesmo resultado se obtém usando o Teorema Mestre
  - Verifiquem !
- Não há contradição relativamente à análise experimental !
- $C_2(n) \in \mathcal{O}(n)$

Chamadas recursivas ?

$$T_3(n) = \begin{cases} n, \text{se } n = 0, 1, 2 \\ 2 \times T_3\left(\frac{n}{3}\right) + n, \text{se } n \text{ é múltiplo de } 3 \\ T_3\left(\left\lfloor\frac{n}{3}\right\rfloor\right) + T_3\left(\left\lceil\frac{n}{3}\right\rceil\right) + n, \text{caso contrário} \end{cases}$$

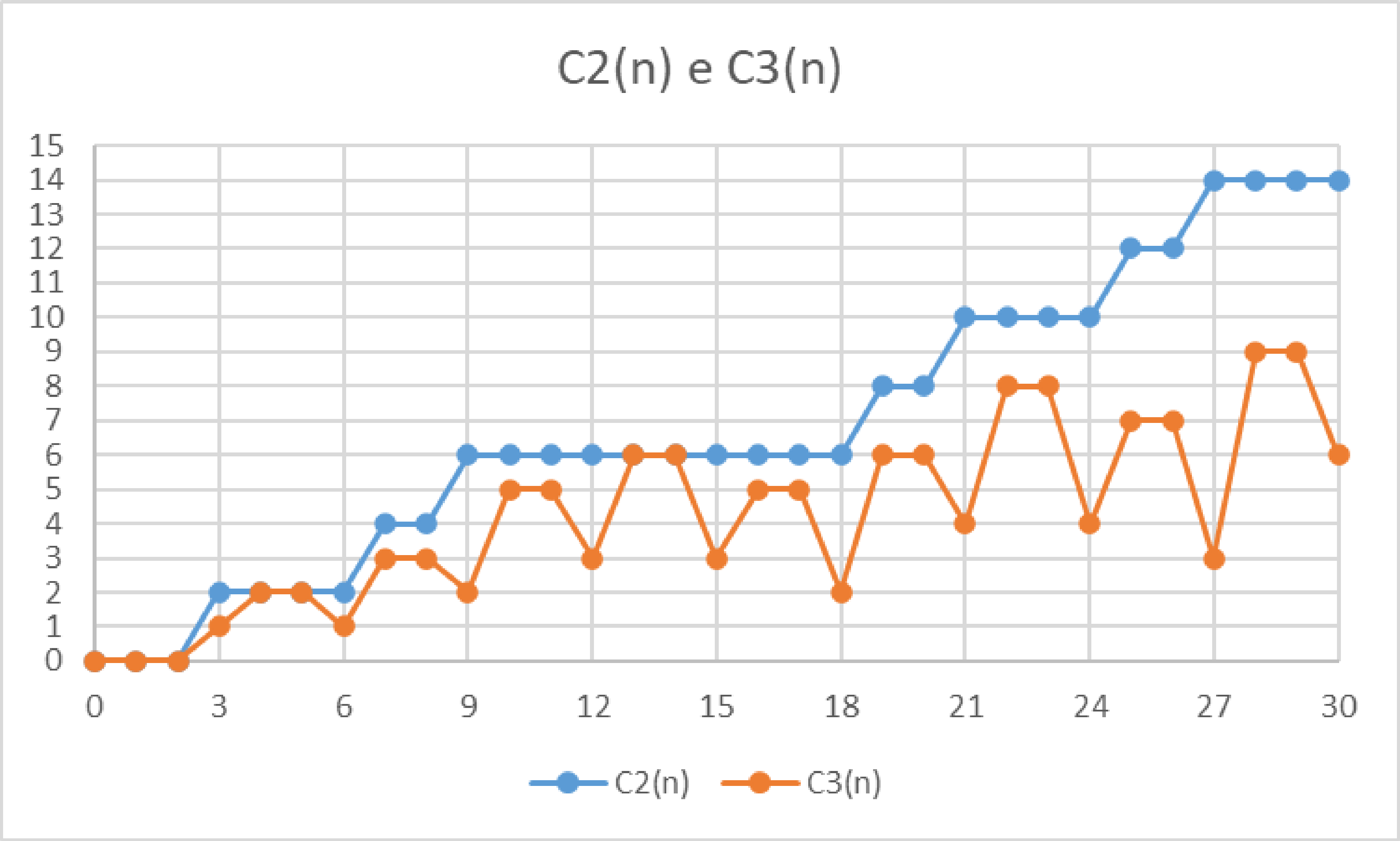
$$C_3(n) = \begin{cases} 0, \text{se } n = 0, 1, 2 \\ 1 + C_3\left(\frac{n}{3}\right), \text{se } n \text{ é múltiplo de } 3 \\ 2 + C_3\left(\left\lfloor\frac{n}{3}\right\rfloor\right) + C_3\left(\left\lceil\frac{n}{3}\right\rceil\right), \text{caso contrário} \end{cases}$$

Resultados experimentais



$$C_3(n) \in \mathcal{O}(?)$$

$T_2(n)$ e $T_3(n)$ produzem os mesmos resultados



$$C_2(n) \in \mathcal{O}(n^{\log_3 2})$$
$$C_3(n) \in \mathcal{O}(n^{\log_3 2})$$

Caso particular: $\boldsymbol{n = 3^k}, k = \log_3 n$

- Obtém-se a recorrência simplificada:

$$C(n) = 1 + C\left(\frac{n}{3}\right)$$

- Desenvolvimento telescópico

$$C(n) = 1 + C\left(\frac{n}{3}\right) = 1 + 1 + C\left(\frac{n}{9}\right) = \cdots$$

$$= k + C\left(\frac{n}{3^k}\right) = k + C(1) = k = \log_3 n$$

Caso particular: $\boldsymbol{n = 3^k}, k = \log_3 n$

- De acordo com os valores da tabela, para estes casos particulares

- **MAS**, olhando para os resultados experimentais, vemos um comportamento diferente para os outros valores de n

- Sugestão:
  - Considerar a representação na base 3
  - Identificar "melhores" e "piores" casos para o mesmo número de dígitos

"Piores" casos

Nº de dígitos	n	n na base 3	Nº de chamadas
2	7 = 2 x 3 + 1	21	3
2	8 = 2 x 3 + 2	22	3
3	22 = 3 x 7 + 1 ou 22 = 3 x 8 – 2	211	8
3	23 = 3 x 7 + 2 ou 23 = 3 x 8 – 1	212	8
4	67 = 3 x 22 + 1 ou 67 = 3 x 23 – 2	2111	18
4	68 = 3 x 22 + 2 ou 68 = 3 x 23 – 1	2112	18
5	202 = 3 x 67 + 1 ou 202 = 3 x 68 – 2	21111	38
5	203 = 3 x 67 + 2 ou 203 = 3 x 68 – 1	21112	38

“Piores” casos

- O **número de chamadas recursivas no pior caso, sendo m o número de dígitos**, define a recorrência

$$c(2) = 3$$

$$c(m) = 2 \times c(m-1) + 2$$

- A solução desta recorrência é Wolfram Alpha

$$c(m) = 5 \times 2^{m-2} - 2$$

- $c(m) \in \mathcal{O}(2^m)$

“Piores” casos

- Número de chamadas recursivas no pior caso, sendo m o número de dígitos

$$c(m) = 5 \times 2^{m-2} - 2$$

- O número de dígitos de n na base 3 é dado por $m = 1 + \lfloor \log_3 n \rfloor$

$$2^m = 2^{(1+\lfloor \log_3 n \rfloor)} = 2 \times 2^{\lfloor \log_3 n \rfloor} = 2 \times 3^{\lfloor \log_3 n \rfloor \log_3 2}$$

$$\leq 2 \times n^{\log_3 2} \leq 2 \times n^{0,631}$$

“Piores” casos

- Ordem de complexidade do número de chamadas recursivas:

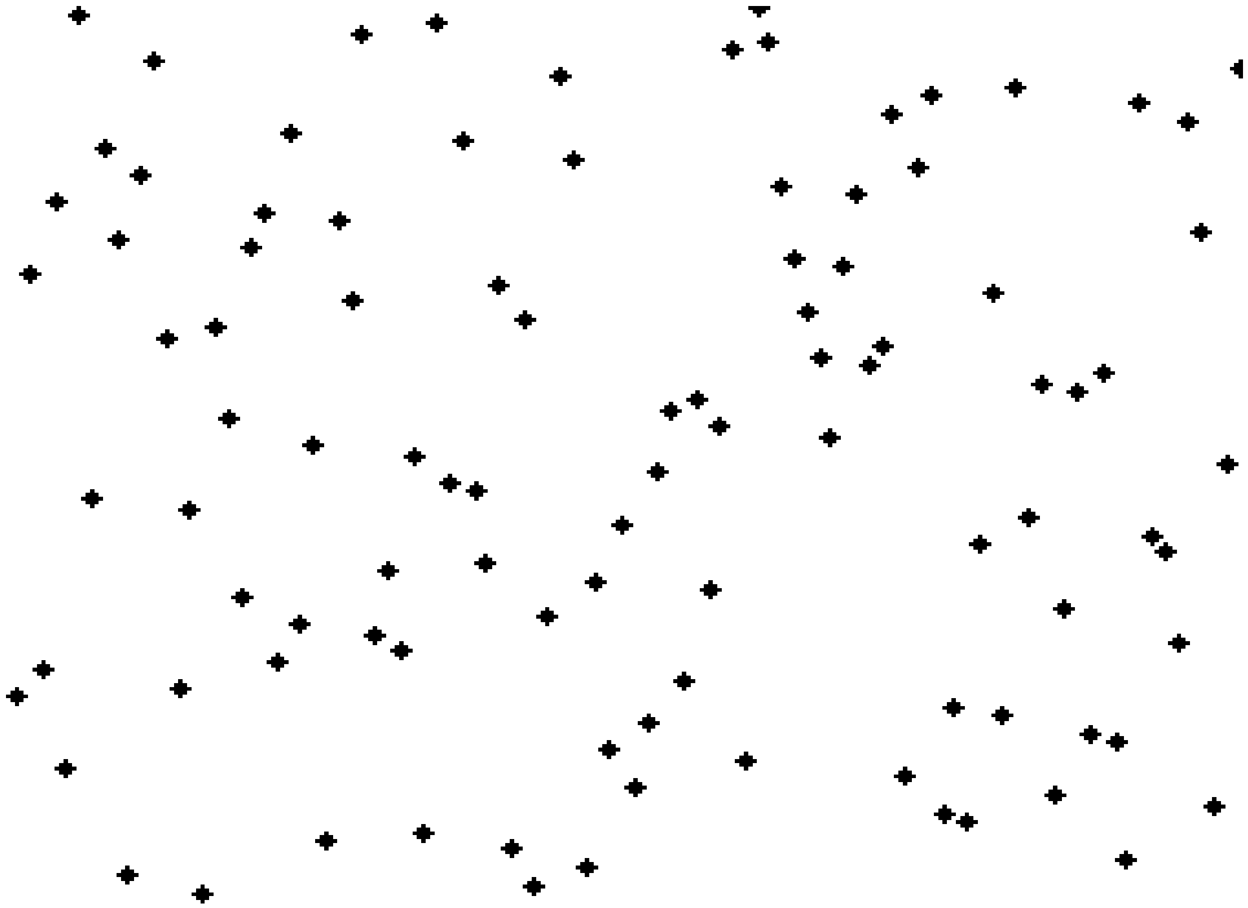
$$C_3(n) \in \mathcal{O}\left(n^{\log_3 2}\right) = \mathcal{O}(n^{0,631})$$

- **Esta ordem de complexidade é válida para qualquer n, uma vez que foram analisados os piores casos para m bits**

- **Confirma-se a análise experimental !!**

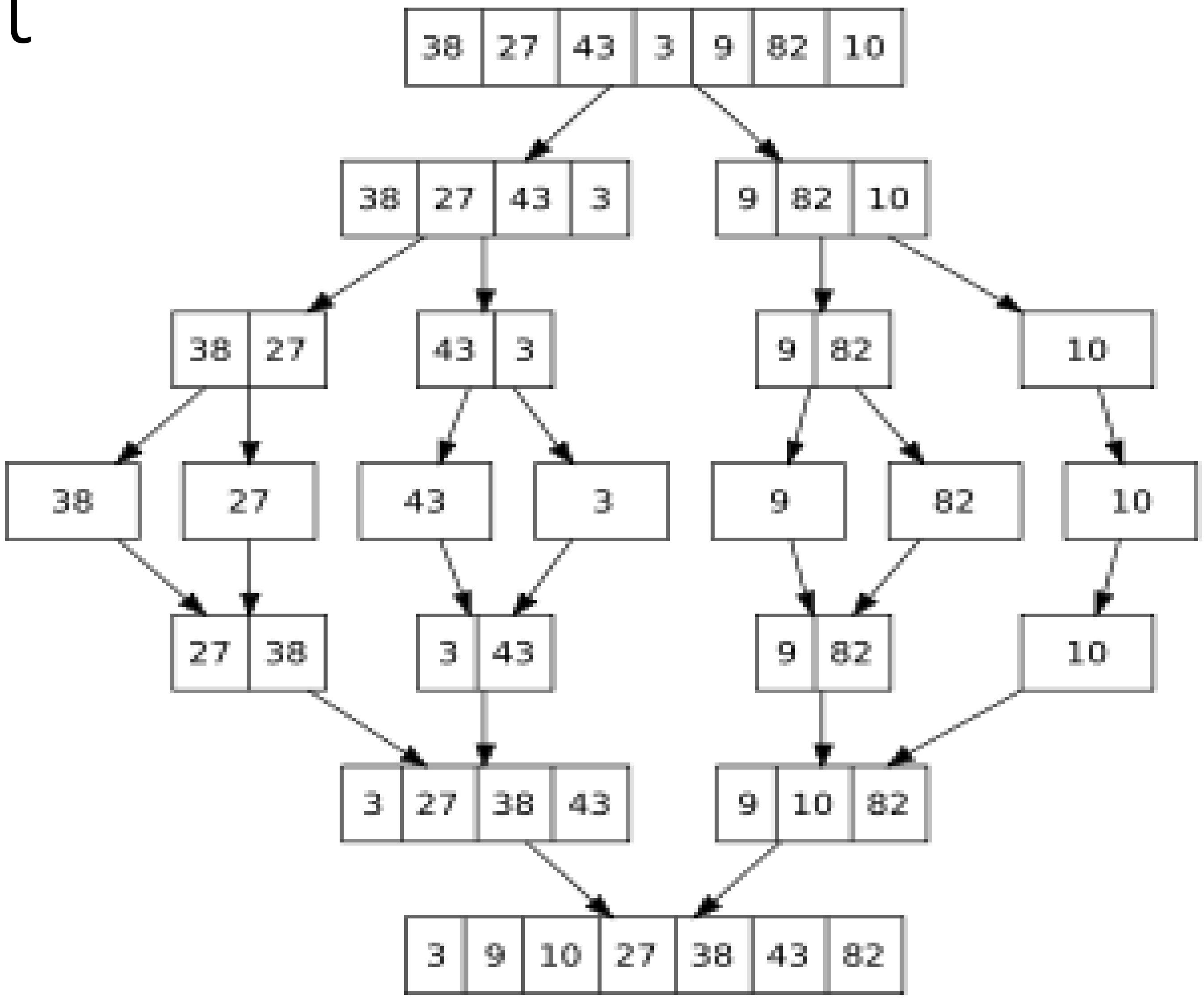
Recapitulação

Mergesort – Ordenação por Fusão



[Wikipedia]

Mergesort



SUBDIVISÃO

FUSÃO

[Wikipedia]

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

Eficiência

- Comparações são feitas pela função de fusão
  - Melhor caso / Pior caso / Caso médio

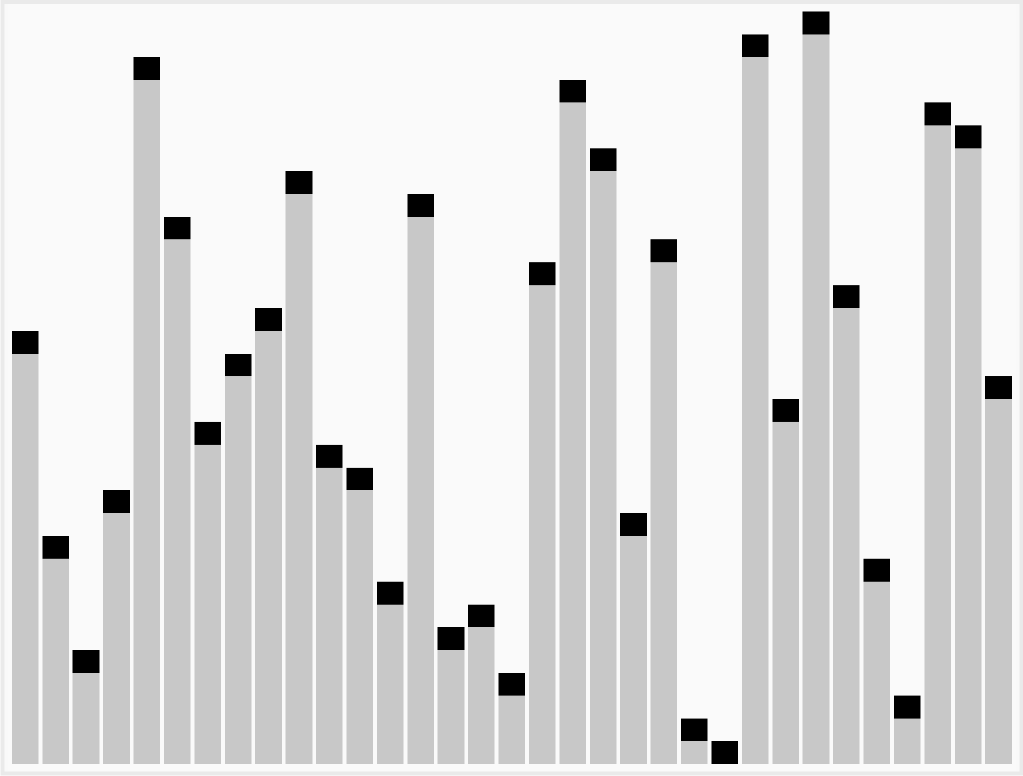
$$C_{sort}(1) = 0$$

$$C_{sort}(n) = C_{sort}(n \textbf{ div } 2) + C_{sort}((n + 1) \textbf{ div } 2) + C_{merge}(n)$$

- $C_{merge}(n) = \Theta(n)$ usando um array auxiliar
- $C_{sort}(n) \quad = \Theta(n \log n)$

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

Exemplo – Pivot é o 1º elemento

- Fase de partição

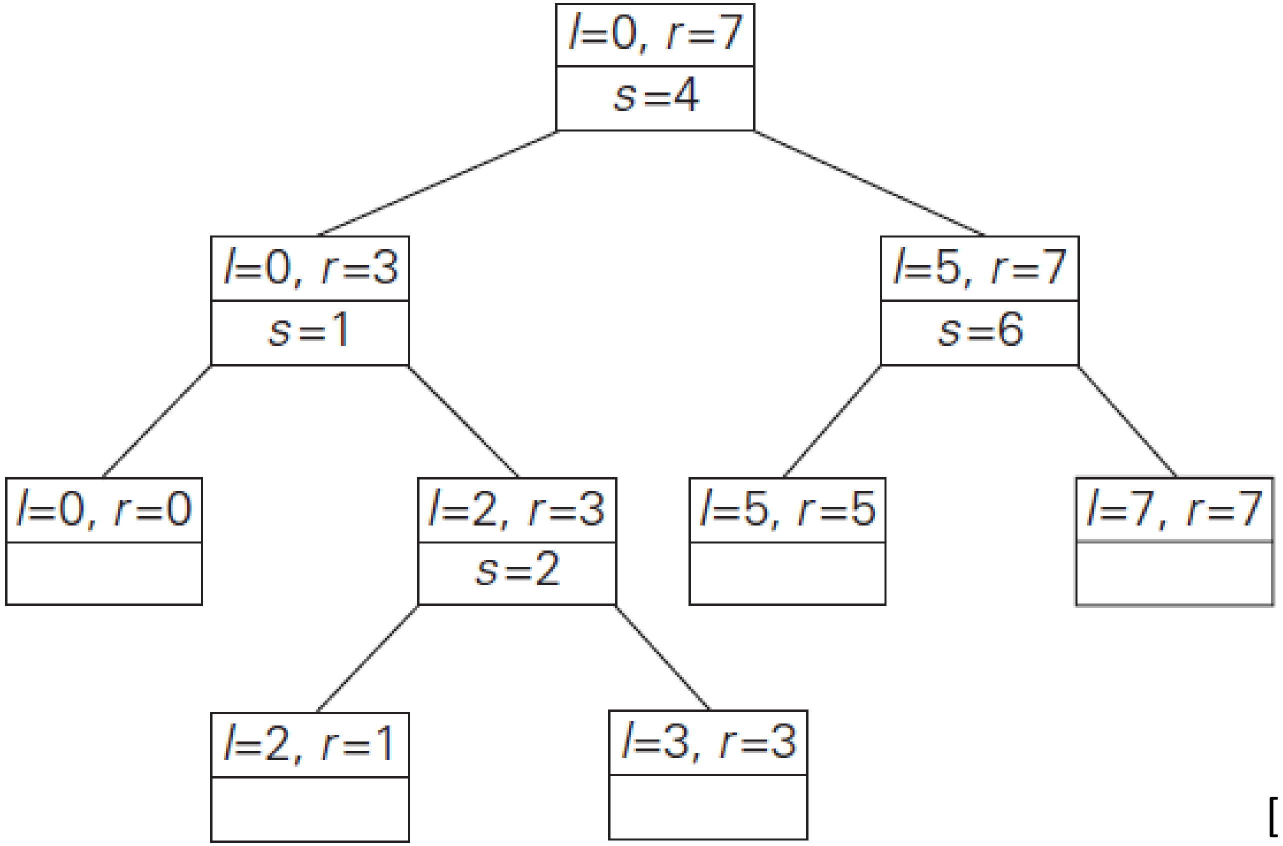
0	1	2	3	4	5	6	7
	i						*j*
5	3	1	9	8	2	4	7
			i			*j*	
5	3	1	9	8	2	4	7
			i			*j*	
5	3	1	4	8	2	9	7
				i	*j*		
5	3	1	4	8	2	9	7
				i	*j*		
5	3	1	4	2	8	9	7
				j	*i*		
5	3	1	4	2	8	9	7
2	3	1	4	**5**	8	9	7

[Levitin]

- Concluir !!

Exemplo – Pivot é o 1º elemento

- Chamadas recursivas



[Levitin]

- s é a posição do pivot após a partição

Outro exemplo

	lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
initial values				Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
random shuffle				K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
	0	5	15	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
	0	3	4	E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
	0	2	2	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	0	0	1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	1		1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	4		4	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	6	6	15	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	7	9	15	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	7	7	8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	8		8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	10	13	15	A	C	E	E	I	K	L	M	O	P	S	Q	R	T	U	X
	10	12	12	A	C	E	E	I	K	L	M	O	P	R	Q	S	T	U	X
	10	11	11	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	10		10	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	14	14	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	15		15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
result				A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

no partition for subarrays of size 1

[Sedgewick and Wayne]

Quicksort trace (array contents after each partition)

1ª versão

```c
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

- Analisar outras versões em diferentes linguagens de programação

- https://rosettacode.org/wiki/Sorting_algorithms/Quicksort

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

Melhor caso

Comp(n) ~ n log n

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
initial values			H	A	C	B	F	E	G	D	L	I	K	J	N	M	O
random shuffle			H	A	C	B	F	E	G	D	L	I	K	J	N	M	O
0	7	14	D	A	C	B	F	E	G	H	L	I	K	J	N	M	O
0	3	6	B	A	C	D	F	E	G	H	L	I	K	J	N	M	O
0	1	2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
0		0	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
2		2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
4	5	6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
4		4	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
6		6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
8	11	14	A	B	C	D	E	F	G	H	J	I	K	L	N	M	O
8	9	10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
8		8	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
10		10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
12	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12		12	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
14		14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

[Sedgewick and Wayne]

Melhor caso

- Há sempre duas partições com “metade” dos elementos

$$C_{best}(n) = 2C_{best}(n/2) + n \quad \text{for } n > 1, \quad C_{best}(1) = 0.$$

Pior caso

Comp(n) ~ n^2 / 2

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
initial values			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
random shuffle			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	0	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	2	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	3	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
4	4	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	5	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
6	6	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
7	7	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
8	8	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
9	9	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
10	10	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
11	11	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12	12	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
13	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
14		14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

[Sedgewick and Wayne]

Pior caso

- Há sempre uma partição com (n – 1) elementos

$$C_{worst}(n) = (n+1) + n + \cdots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2).$$

- Muito pouco provável !!
  - Fazer um “shuffling” inicial aos dados

Caso médio

Comp(n) ~ n log n

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
			K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
0	5	15	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	3	4	E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	2	2	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	0	1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
1		1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
4		4	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
6	6	15	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
7	9	15	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
7	7	8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
8		8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
10	13	15	A	C	E	E	I	K	L	M	O	P	S	Q	R	T	U	X
10	12	12	A	C	E	E	I	K	L	M	O	P	R	Q	S	T	U	X
10	11	11	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
10		10	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
14	14	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
15		15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
			A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

[Sedgewick and Wayne]

Caso médio

- Elementos distintos
- Tamanho equiprovável para as sucessivas partições

$$C_{avg}(n) = \frac{1}{n} \sum_{s=0}^{n-1} [(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0.$$

$$C_{avg}(n) \approx 2n \ln n \approx 1.39 n \log_2 n.$$

Caso médio

- Mais comparações do que o Mergesort !!
- MAS, na prática, é mais rápido do que o Mergesort !!
- Faz menos movimentações de elementos do array

K-Selection – Selecionar o k-ésimo elemento de um array

K-Selection

- Dado um array com n elementos : A[0,...,n – 1]
- O array não está ordenado !!

- Qual o valor do elemento na posição de índice (k – 1) ?
  - Min (k = 0) / Max (k = n – 1) / Mediano (k = n **div** 2)
  - Aplicação: **top k** elementos

- Possíveis soluções ?
- Complexidade ?

K-Selection

- Possíveis estratégias ?
- Como usar o procedimento de partição do algoritmo Quicksort ?

- Tarefa: desenvolver estratégias…

Sugestões de leitura

Sugestões de leitura

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
  - Capítulo 5: secção 5.2
  - Capítulo 4: secção 4.5