

Análise da Complexidade de Algoritmos Recursivos V

Joaquim Madeira

27/04/2021

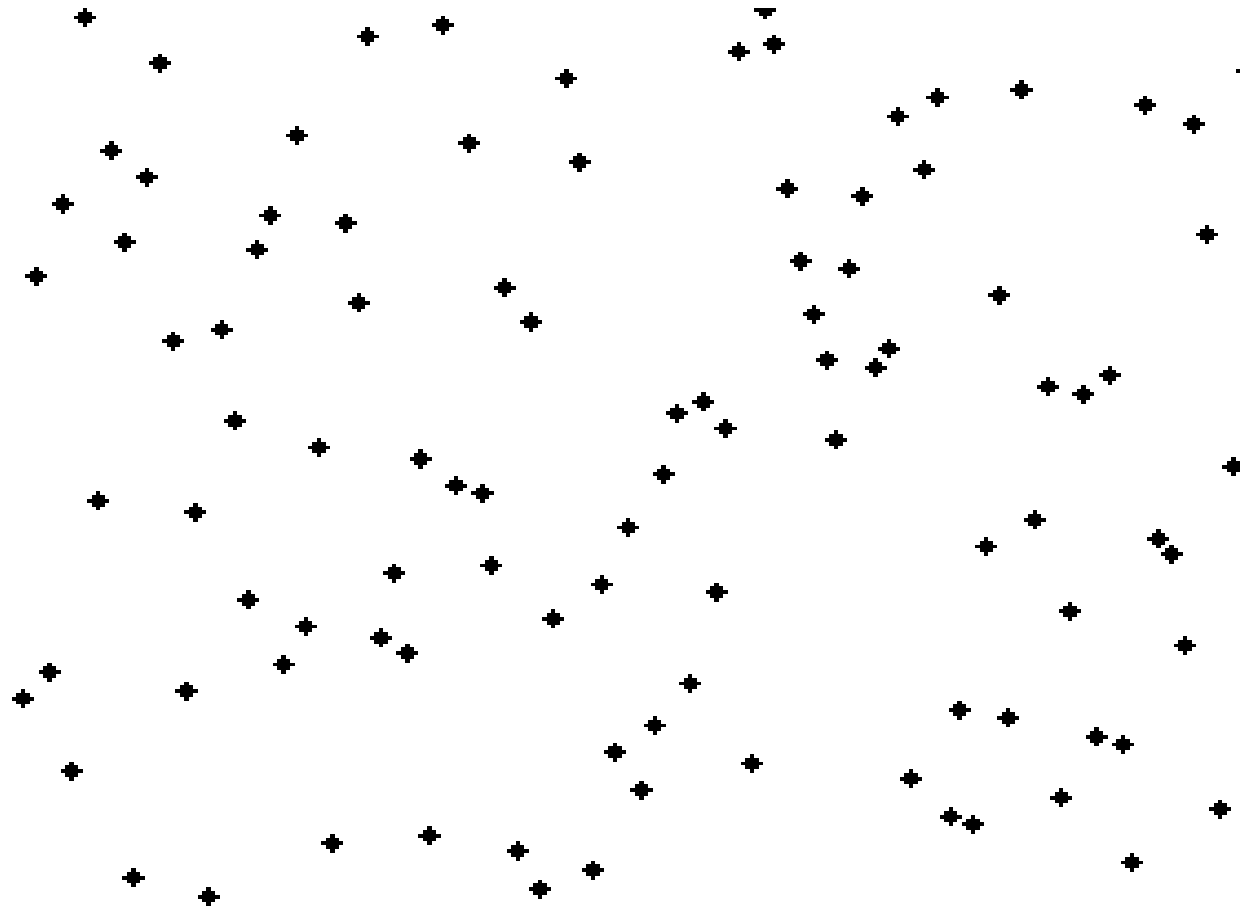
Sumário

- Recap
- Ordenação por partição: o Algoritmo Quicksort – Análise da Complexidade
- Seleção do k-ésimo elemento
- Sugestões de leitura

Let's
RECAP

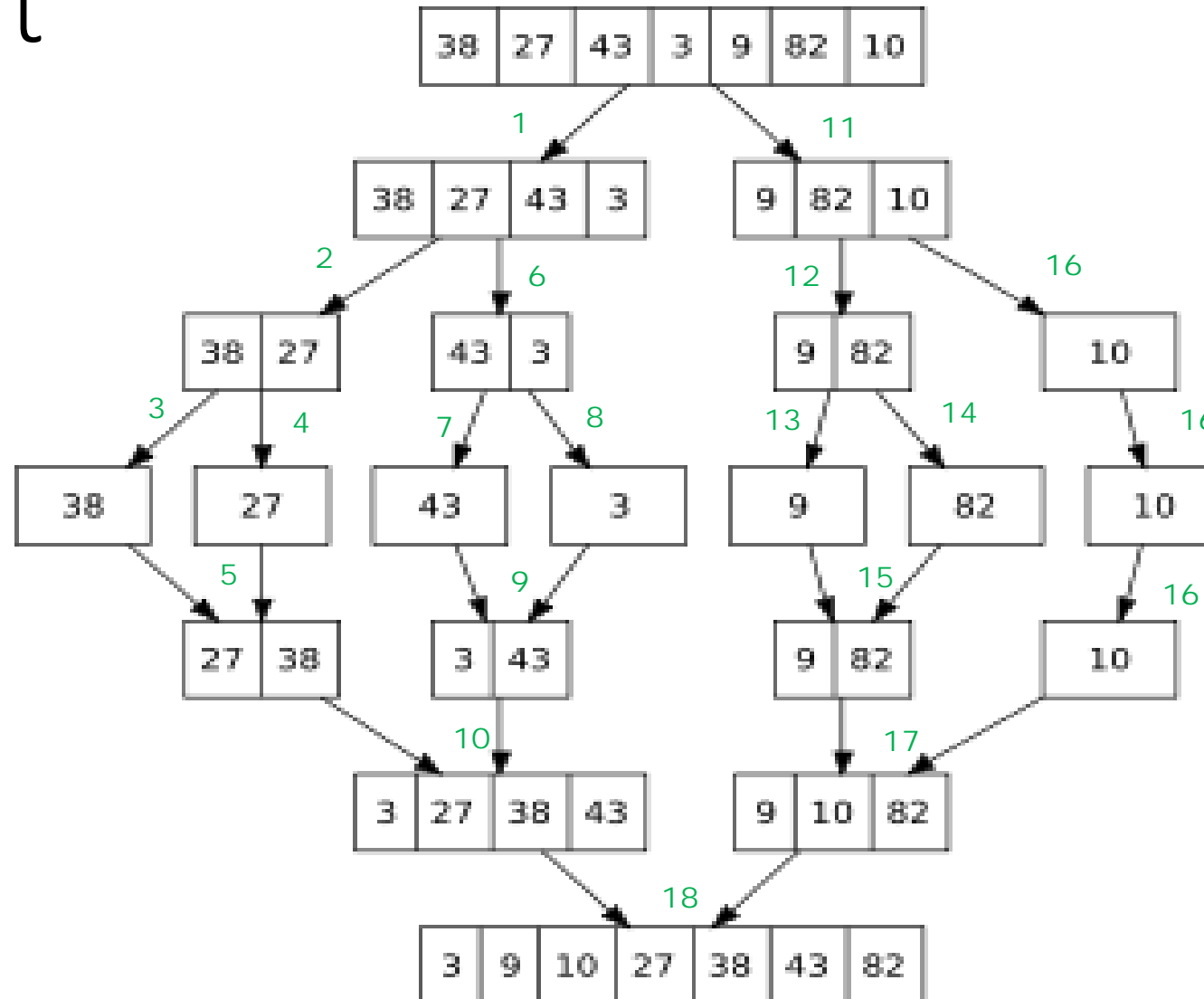
Recapitulação

Mergesort – Ordenação por Fusão



[Wikipedia]

Mergesort



SUBDIVISÃO

FUSÃO

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

[Wikipedia]

Eficiência

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

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

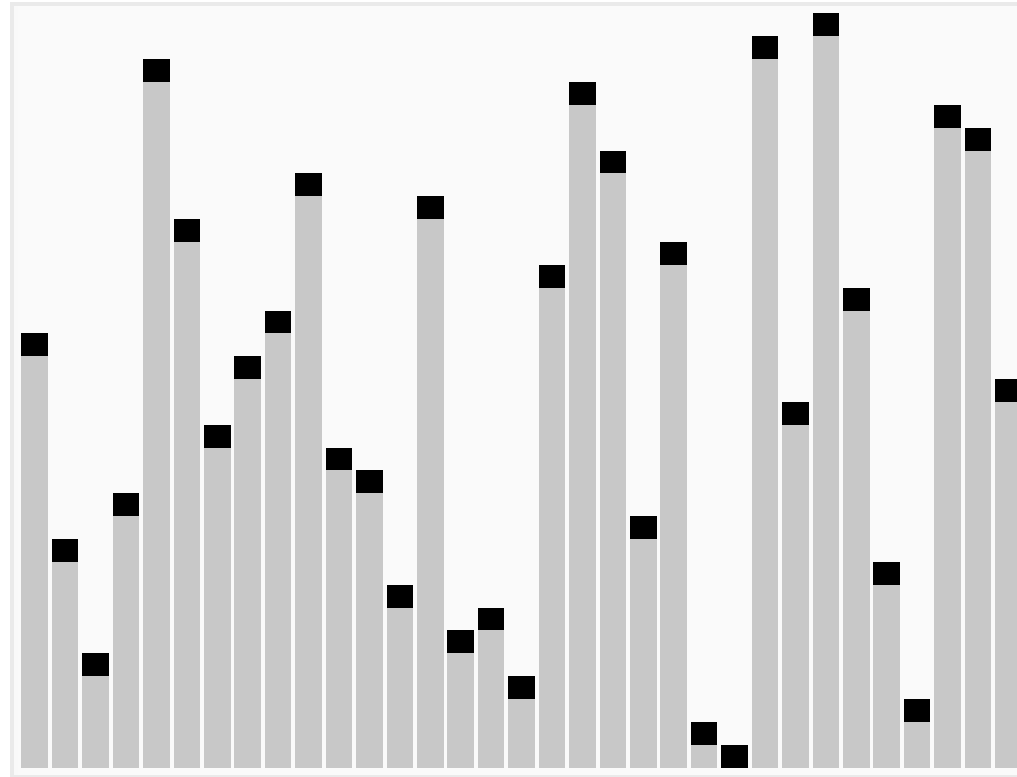
$$C_{\text{sort}}(n) = C_{\text{sort}}(n \text{ div } 2) + C_{\text{sort}}((n + 1) \text{ div } 2) + C_{\text{merge}}(n)$$

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

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

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);  
}
```



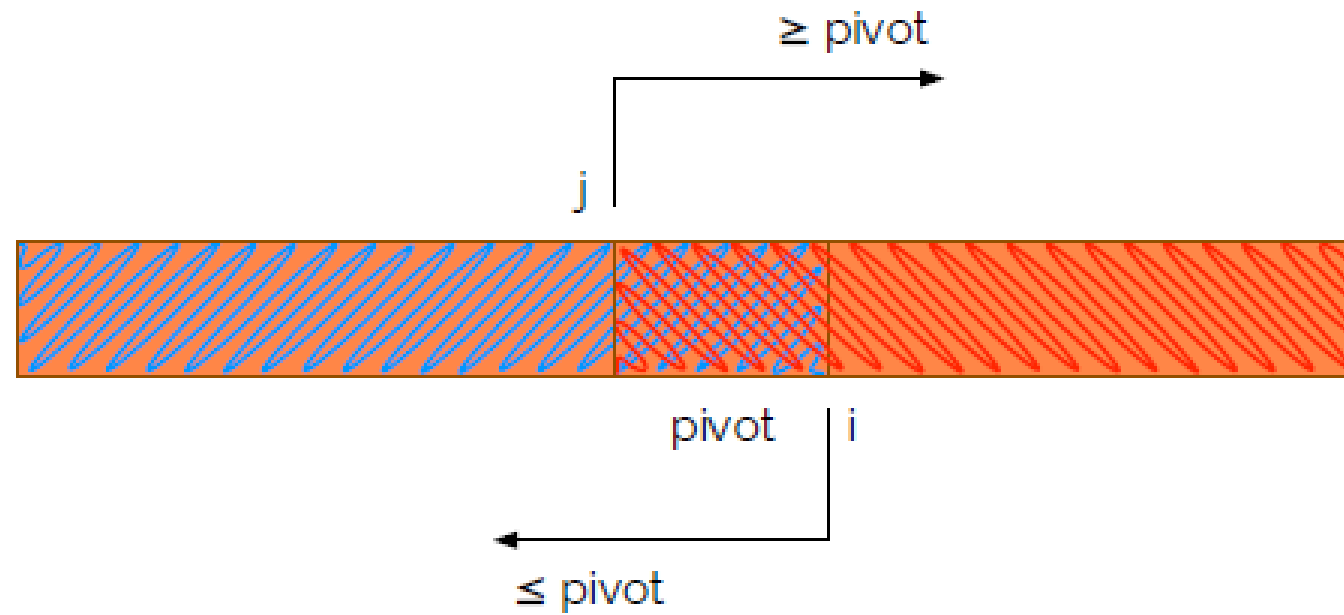
Tarefa 1

- Analisar outras **versões** em **diferentes linguagens** de programação
- https://rosettacode.org/wiki/Sorting_algorithms/Quicksort

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]

Fizeram ?

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 ?

Quicksort – Pivot = elemento do “meio”

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

6	5	4	3	2	1
---	---	---	---	---	---

Quicksort

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

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
i					j
6	5	4	3	2	1

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
i		j			
1	5	4	3	2	6

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
i			j		
1	5	4	3	2	6

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
i			j		
1	5	4	3	2	6

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
i			j		
1	5	4	3	2	6

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
		i	j		
1	2	4	3	5	6

Quicksort

0	1	2	3	4	5
6	5	4	3	2	1
		i	j		
1	2	4	3	5	6

Quicksort

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

Quicksort

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

- O array já está **ordenado** no final da fase de partição !!
- MAS, as chamadas recursivas vão **continuar** !!
- Terminar o exemplo !!

Exemplo – Pivot é o 1º elemento – Fizeram ?

- Fase de partição

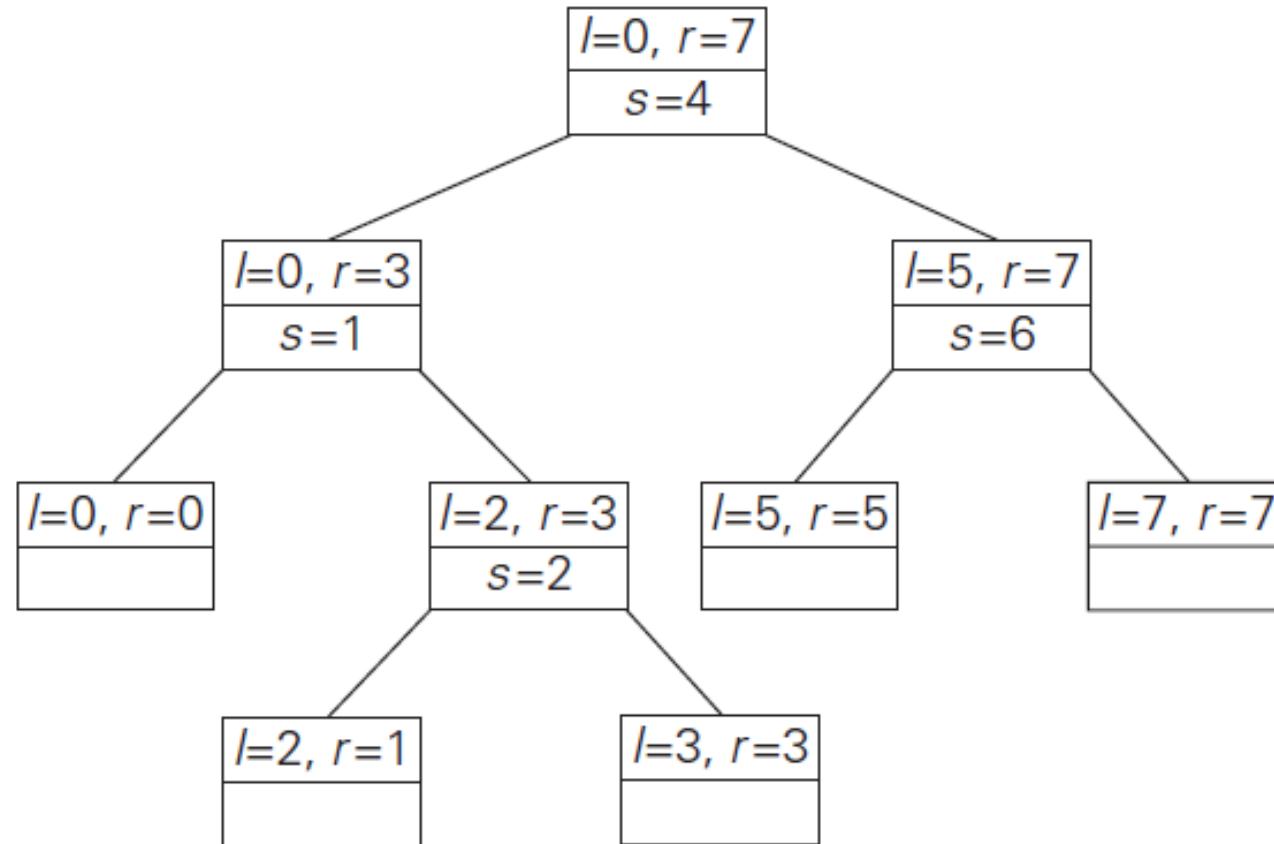
0	1	2	3	4	5	6	7
	<i>i</i>						<i>j</i>
5	3	1	9	8	2	4	7
5	3	1	<i>i</i> 9	8	2	<i>j</i> 4	7
5	3	1	<i>i</i> 4	8	2	<i>j</i> 9	7
5	3	1	4	<i>i</i> 8	<i>j</i> 2	9	7
5	3	1	4	<i>i</i> 2	<i>j</i> 8	9	7
5	3	1	4	<i>j</i> 2	<i>i</i> 8	9	7
2	3	1	4	5	8	9	7

[Levitin]

- Concluir !!

Exemplo – Pivot é o 1º elemento – Fizeram ?


- Chamadas recursivas



[Levitin]

- s é a posição do pivot após a partição

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

Quicksort trace (array contents after each partition)

[Sedgewick and Wayne]

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 !!
 - Ou se gerarmos uma **permutação aleatória** do array dado

Melhor Caso

Comp(n) ~ n log n

[Sedgewick and Wayne]

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

Melhor Caso

- Há sempre duas partições com “aprox. metade” dos elementos



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

- The Master Theorem !!

Pior Caso


$$\text{Comp}(n) \sim n^2 / 2$$

[Sedgewick and Wayne]

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

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

- Simplificar : **elementos distintos !!**
- **Tamanho equiprovável** para as sucessivas partições

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

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

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

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** : $A[0, \dots, n - 1]$
- O array **não está ordenado !!**
- **Se** o array **estivesse ordenado**
- Qual seria o valor do **elemento** na posição de **índice k** ?
 - **Min** ($k = 0$) / **Max** ($k = n - 1$) / **Mediano** ($k = n \text{ div } 2$)
 - Aplicação: **top k** elementos
- **Possíveis soluções ?**
- **Complexidade ?**

K-Selection

- Possíveis **estratégias** ?
- Como usar o **procedimento de partição** do algoritmo Quicksort ?
- Vamos começar com a **estratégia direta** !!

K-Selection – Estratégia direta

- Ordenar por ordem não decrescente os n elementos
- Consultar o elemento na posição k
- Quanto tempo ?
 - 1.000.000 elementos / 10.000.000 elementos / ...

K-Selection – Estratégia direta

- Ordenar por ordem não decrescente os n elementos
 $O(n^2)$ ou $O(n \log n)$
- Consultar o elemento na posição k
 $O(1)$
- Quanto tempo ?
 - 1.000.000 elementos / 10.000.000 elementos / ...

K-Selection – Estratégia direta

- Demasiado **esforço computacional** !!
- Se o objetivo for apenas consultar o **valor de um** ou de **alguns elementos**
- Se **k** for **muito menor** que **n**
- Como proceder de modo **mais eficiente** ?

K-Selection – Estratégia melhorada

- Copiar os primeiros $(k + 1)$ elementos para um array S
- Ordenar o array S por ordem não decrescente
- Para cada um dos restantes $(n - k - 1)$ elementos de A
 - Ignorar se maior ou igual que $S[k]$
 - Caso contrário, inserir ordenadamente em S
 - O elemento $S[k]$ é expulso do array e substituído

K-Selection – $k = 1$ – 2º elemento

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

K-Selection – $k = 1$ – 2º elemento

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

3	6
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

3	6
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

3	6
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

3	4
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

3	4
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

3	4
---	---

K-Selection – $k = 1 - 2^0$ elemento

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

i

2	3
---	---

K-Selection – $k = 1$ – 2º elemento

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

i

2	3
---	---

K-Selection – $k = 1$ – 2º elemento

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

K-Selection – $k = 1$ – 2º elemento

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

2	3
---	---

K-Selection – $k = 1$ – 2º elemento

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

1	2
---	---

K-Selection – $k = 1$ – 2º elemento

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

1	2
---	---

K-Selection – Estratégia melhorada

- O que demora mais **tempo** ?
- **Ordenar** os primeiros $(k + 1)$ elementos
 $O(k^2)$ ou $O(k \log k)$
- Para cada um dos restantes $(n - k - 1)$ elementos
 - **Ignorar** se maior ou igual que $S[k]$ $O(1)$
 - Caso contrário, **inserir ordenadamente** em S $O(k)$

K-Selection – Estratégia melhorada

- Ordem de complexidade ?

$$O(k \log k) + (n - k - 1) \times O(k) = O(n \times k)$$

- Encontrar o elemento mediano $O(n^2)$

- Quanto tempo ?

- 1.000.000 elementos / 10.000.000 elementos / ...

K-Selection – Estratégia melhorada

- Será ainda possível fazer melhor ?
- O que poderemos usar dos algoritmos e estruturas de dados que conhecemos ?
- Será necessário manter o conjunto dinâmico de $(k + 1)$ elementos completamente ordenado ?

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](#)
- M. A. Weiss, Data Structures and Algorithm Analysis in C++, 4th Edition, 2014
 - Capítulo 1: [secção 1.1](#)
 - Capítulo 7: [secção 7.7](#)