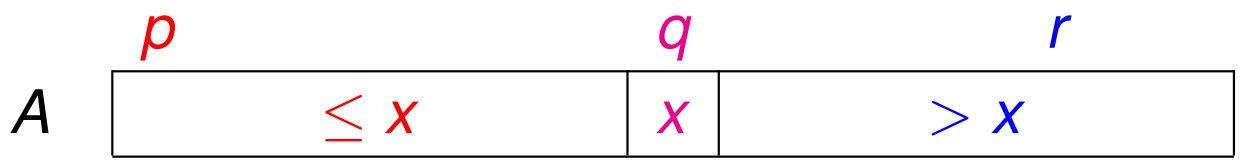
CI165 — QuickSort

André Vignatti

QuickSort

O algoritmo QUICKSORT segue o paradigma de divisão-e-conquista.

Divisão: divida o vetor em dois subvetores $A[p \ldots q-1]$ e $A[q+1 \ldots r]$ tais que



$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

Conquista: ordene os dois subvetores recursivamente usando o QUICKSORT;

Combinação: nada a fazer, o vetor está ordenado.

O passo de divisão é feito pelo procedimento PARTICIONE, que devolve um índice q que marca a posição de divisão dos subvetores.

QuickSort

Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

```
QUICKSORT(A, p, r)
1  se p < r
2      então q ← PARTICIONE(A, p, r)
3            QUICKSORT(A, p, q − 1)
4            QUICKSORT(A, q + 1, r)
```

- A chamada inicial é QUICKSORT$(A, 1, n)$.

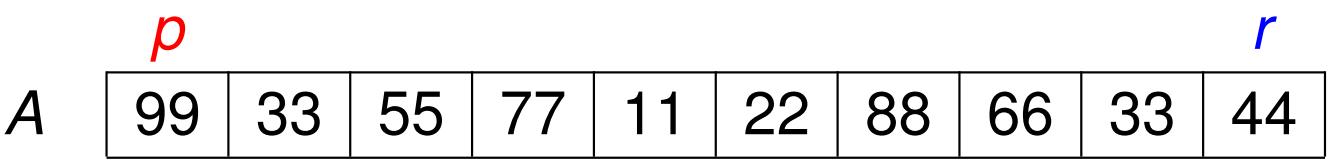
Antes de tentar entender o QUICKSORT, temos que entender o PARTICIONE.

Partição

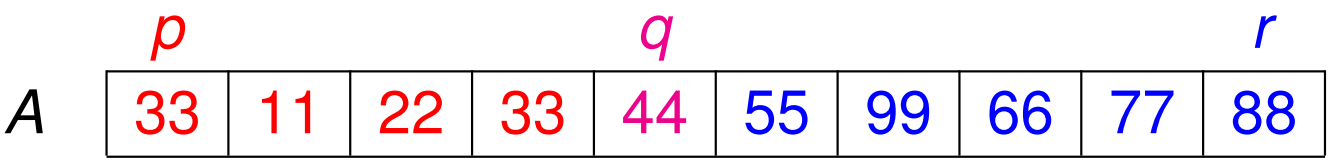
Problema: Rearranjar um dado vetor $A[p \ldots r]$ e devolver um índice q, $p \leq q \leq r$, tais que

$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

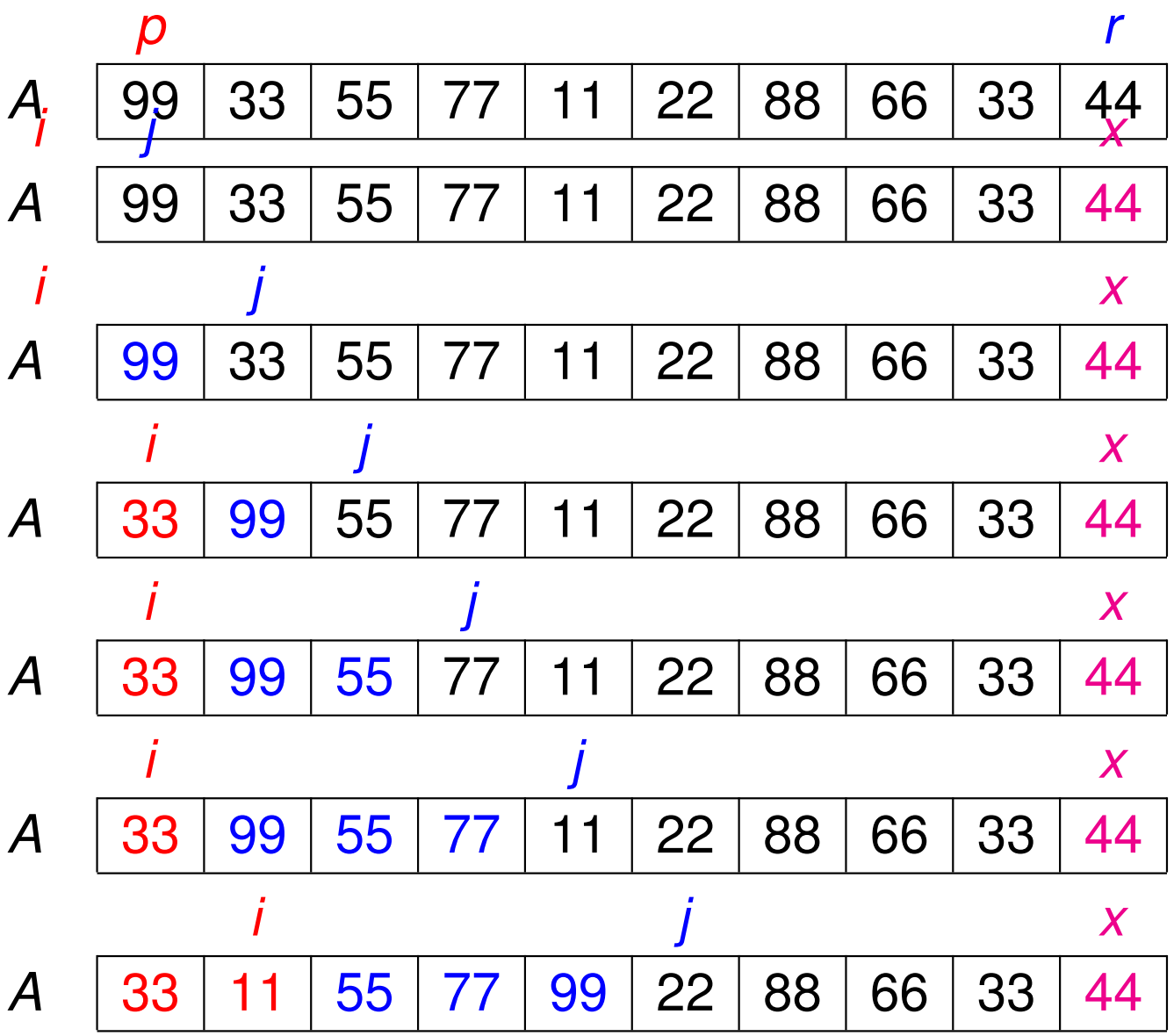
Entrada:



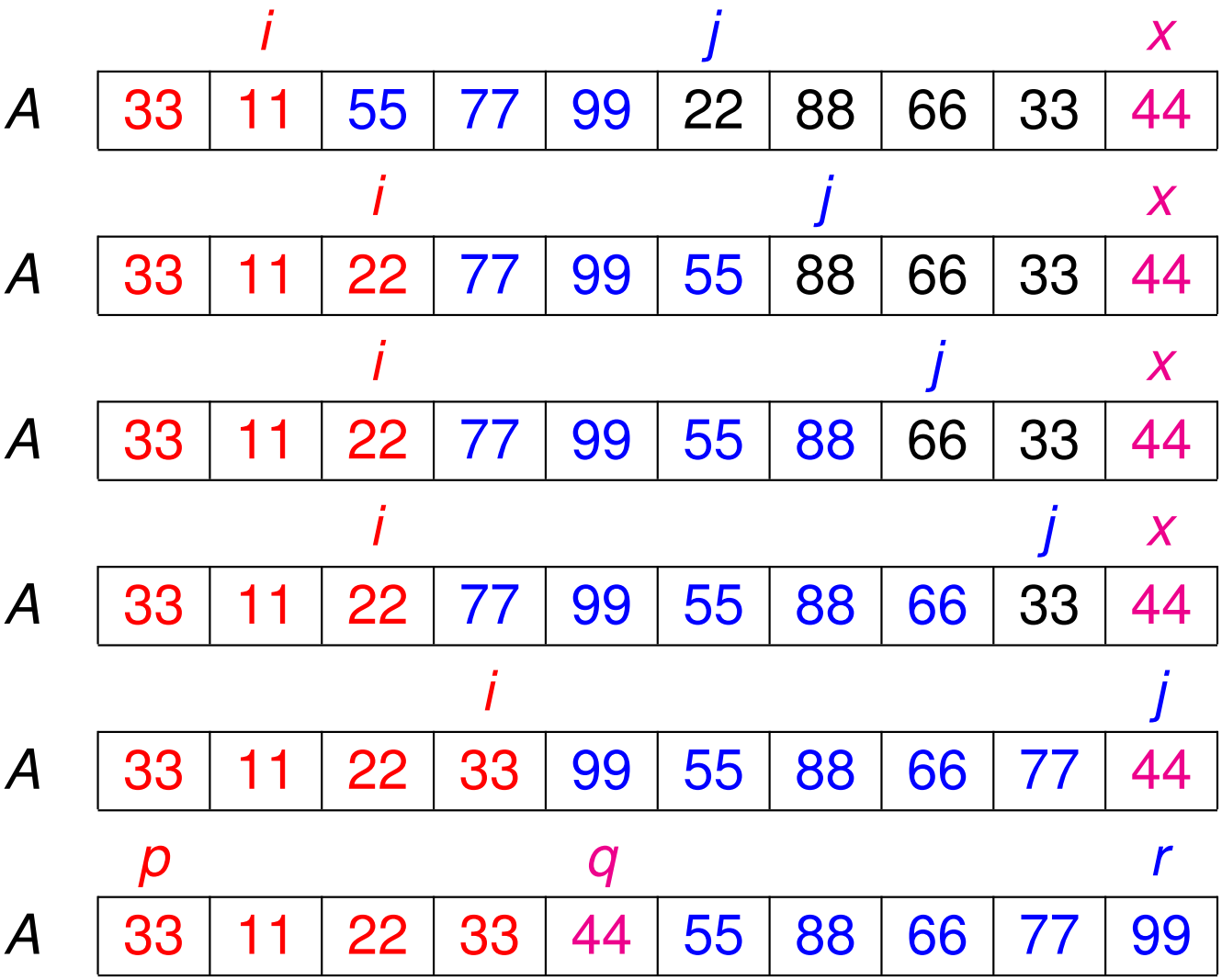
Saída:



Particione


p
r
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44

Particione


i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
A
33 11 22 33 99 55 88 66 77 44
p
q
r
A
33 11 22 33 44 55 88 66 77 99

Particione

Rearranja $A[p \ldots r]$ de modo que $p \leq q \leq r$ e
$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$

```
PARTICIONE(A, p, r)
1  x ← A[r]   ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```

Invariantes:

No começo de cada iteração da linha 3 vale que:
(1) $A[p \ldots i] \leq x$ (2) $A[i+1 \ldots j-1] > x$ (3) $A[r] = x$

Complexidade de PARTICIONE

	PARTICIONE(A, p, r)	Tempo
1	$x \leftarrow A[r]$ $\triangleright$ x é o "pivô"	?
2	$i \leftarrow p-1$	?
3	**para** $j \leftarrow p$ **até** $r-1$ **faça**	?
4	**se** $A[j] \leq x$	?
5	**então** $i \leftarrow i+1$	?
6	$A[i] \leftrightarrow A[j]$	?
7	$A[i+1] \leftrightarrow A[r]$	?
8	**devolva** $i+1$	?

$T(n)$ = complexidade de tempo no pior caso sendo

$n := r - p + 1$

Complexidade de PARTICIONE

PARTICIONE(A, p, r)		Tempo
1	$x \leftarrow A[r]$ $\triangleright$ x é o "pivô"	$\Theta(1)$
2	$i \leftarrow p-1$	$\Theta(1)$
3	**para** $j \leftarrow p$ **até** $r-1$ **faça**	$\Theta(n)$
4	**se** $A[j] \leq x$	$\Theta(n)$
5	**então** $i \leftarrow i+1$	$O(n)$
6	$A[i] \leftrightarrow A[j]$	$O(n)$
7	$A[i+1] \leftrightarrow A[r]$	$\Theta(1)$
8	**devolva** $i+1$	$\Theta(1)$

$$T(n) = \Theta(2n+4) + O(2n) = \Theta(n)$$

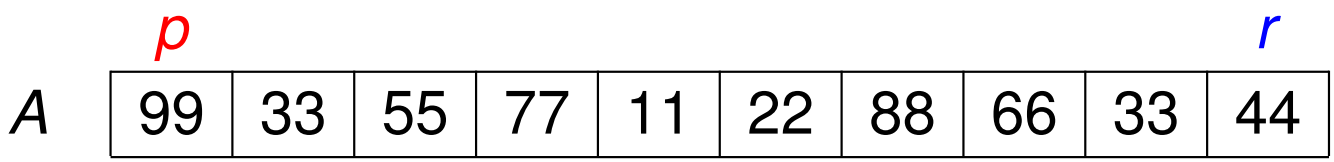
Conclusão:

A complexidade de PARTICIONE é $\Theta(n)$.

QuickSort

Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

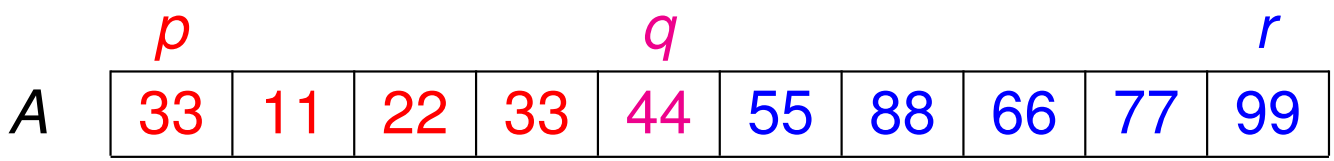
```
QUICKSORT(A, p, r)
1  se  p < r
2      então  q ← PARTICIONE(A, p, r)
3             QUICKSORT(A, p, q − 1)
4             QUICKSORT(A, q + 1, r)
```



QuickSort

Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

```
QUICKSORT(A, p, r)
1  se  p < r
2      então  q ← PARTICIONE(A, p, r)
3             QUICKSORT(A, p, q − 1)
4             QUICKSORT(A, q + 1, r)
```



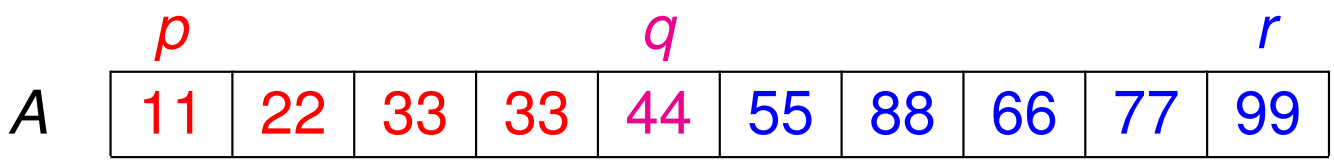
No começo da linha 3,

$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

QuickSort

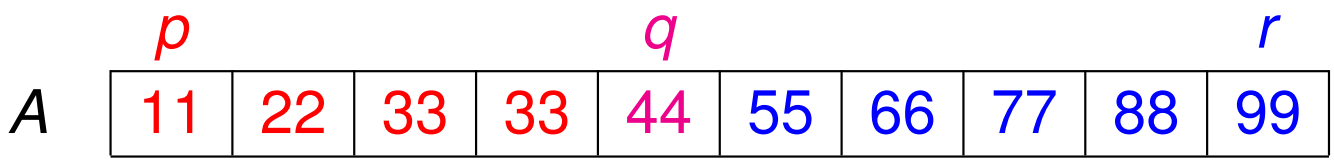
Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

```
QUICKSORT(A, p, r)
1  se p < r
2     então q ← PARTICIONE(A, p, r)
3           QUICKSORT(A, p, q − 1)
4           QUICKSORT(A, q + 1, r)
```



Rearranja um vetor $A[p \ldots r]$ em ordem crescente.

```
QUICKSORT(A, p, r)
1  se p < r
2     então q ← PARTICIONE(A, p, r)
3           QUICKSORT(A, p, q − 1)
4           QUICKSORT(A, q + 1, r)
```



Complexidade de QUICKSORT

QUICKSORT(A, p, r)			Tempo
1	**se**	$p < r$	?
2		**então** $q \leftarrow$ PARTICIONE(A, p, r)	?
3		QUICKSORT$(A, p, q-1)$	?
4		QUICKSORT$(A, q+1, r)$	?

$T(n) :=$ complexidade de tempo no pior caso sendo
$n := r - p + 1$

Complexidade de QUICKSORT

QUICKSORT(A, p, r)			Tempo
1	**se**	$p < r$	$\Theta(1)$
2		**então** $q \leftarrow$ PARTICIONE(A, p, r)	$\Theta(n)$
3		QUICKSORT$(A, p, q-1)$	$T(k)$
4		QUICKSORT$(A, q+1, r)$	$T(n-k-1)$

$$T(n) = T(k) + T(n-k-1) + \Theta(n+1)$$

$$0 \leq k := q - p \leq n - 1$$

Recorrência

$T(n) :=$ consumo de tempo do QUICKSORT.

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(k) + T(n-k-1) + \Theta(n) \text{ para } n = 2, 3, 4, \ldots$$

Um caso ruim: Quando os subvetores ficam desbalanceados: um com $n-1$ elementos e outro com 0 elementos.

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

- Ocorre quando o vetor já está ordenado (Porquê?)
- E quando o vetor está ordenado ao inverso? [Exercício]

$T(n)$ é $\Theta(n^2)$.

Recorrência de pior caso

Vamos provar que o **caso ruim**, do slide anterior, é de fato o pior caso. $T(n) :=$ complexidade de tempo no pior caso.

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n-k-1)\} + \Theta(n) \text{ para } n = 2, 3, 4, \ldots$$

$$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n-k-1)\} + bn$$

Quero mostrar que $T(n) = \Theta(n^2)$.

Demonstração – $T(n) = O(n^2)$

Usando o método da substituição, vou provar que $T(n) \leq cn^2$ para n grande.

$$
\begin{aligned}
T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + bn \\
&\leq \max_{0 \leq k \leq n-1} \left\{ ck^2 + c(n-k-1)^2 \right\} + bn \\
&= c \max_{0 \leq k \leq n-1} \left\{ k^2 + (n-k-1)^2 \right\} + bn
\end{aligned}
$$

A expressão $k^2 + (n-k-1)^2$ atinge valor máximo quando $k = 0$ ou $k = n-1$ [Exercício CLRS], provando que a divisão em subvetores de tamanhos 0 e $n-1$ é de fato o **pior caso**.

Demonstração – $T(n) = O(n^2)$ (cont.)

Então (fazendo $k = 0$ ou $k = n - 1$ na expressão $k^2 + (n - k - 1)^2$),

$$\begin{aligned} T(n) &\leq c(n-1)^2 + bn \\ &= cn^2 - 2cn + c + bn \\ &\leq cn^2 \,, \end{aligned}$$

se $c > b/2$ e $n \geq c/(2c - b)$.

- A prova que $T(n) = \Omega(n^2)$ é essencialmente a mesma, usando substituição, mas provando que $T(n) \geq dn^2$. [Exercício]

Conclusão

$T(n)$ é $\Theta(n^2)$.

A complexidade de tempo do QUICKSORT no pior caso é $\Theta(n^2)$.

A complexidade de tempo do QUICKSORT é $O(n^2)$.

QuickSort no melhor caso

Não seremos rigorosos na análise de melhor caso (só daremos a ideia).
O melhor caso ocorre quando os subvetores são bem balanceados em toda execução do PARTICIONE.

- Neste caso, cada subvetor tem $\leq n/2$ elementos. (Em uma análise cuidadosa, seriam tamanhos $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil - 1$)
- Obtemos a recorrência:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \log n). \end{aligned}$$

Que implica que o QUICKSORT é $\Omega(n \log n)$.

Mais algumas conclusões

A complexidade de tempo do QUICKSORT no melhor caso é $\Theta(n \log n)$.

A complexidade de tempo do QUICKSORT é $\Omega(n \log n)$.