

Projeto e Análise de Algoritmos*

Introdução a algoritmos aleatorizados e ordenação por
particionamento

Segundo Semestre de 2019

*Criado por C. de Souza, C. da Silva, O. Lee, F. Miyazawa et al.

A maior parte deste conjunto de slides foi inicialmente preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para cursos de Análise de Algoritmos. Além desse material, diversos conteúdos foram adicionados ou incorporados por outros professores, em especial por Orlando Lee e por Flávio Keidi Miyazawa. Os slides usados nessa disciplina são uma junção dos materiais didáticos gentilmente cedidos por esses professores e contêm algumas modificações, que podem ter introduzido erros.

O conjunto de slides de cada unidade do curso será disponibilizado como guia de estudos e deve ser usado unicamente para revisar as aulas. Para estudar e praticar, leia o livro-texto indicado e resolva os exercícios sugeridos.

Lehilton

Agradecimentos (Cid e Cândida)

- ▶ Várias pessoas contribuíram **direta ou indiretamente** com a preparação deste material.
- ▶ Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- ▶ Uma lista destes “colaboradores” (**em ordem alfabética**) é dada abaixo:
 - ▶ Célia Picinin de Mello
 - ▶ Flávio Keidi Miyazawa
 - ▶ José Coelho de Pina
 - ▶ Orlando Lee
 - ▶ Paulo Feofiloff
 - ▶ Pedro Rezende
 - ▶ Ricardo Dahab
 - ▶ Zanoni Dias

Quick Sort

QuickSort

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

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



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

Conquista: ordene os dois subvetores **recursivamente** usando o **QUICKSORT**;

Combinação: nada a fazer, o vetor está ordenado.

QuickSort

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

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



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

Conquista: ordene os dois subvetores **recursivamente** usando o QUICKSORT;

Combinação: nada a fazer, o vetor está ordenado.

QuickSort

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

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



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

Conquista: ordene os dois subvetores **recursivamente** usando o **QUICKSORT**;

Combinação: nada a fazer, o vetor está ordenado.

QuickSort

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

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



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

Conquista: ordene os dois subvetores **recursivamente** usando o QUICKSORT;

Combinação: nada a fazer, o vetor está ordenado.

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

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

Entrada:

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

PARTIÇÃO

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

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

Entrada:

	<i>p</i>									<i>r</i>
A	99	33	55	77	11	22	88	66	33	44

Saída:

	<i>p</i>			<i>q</i>						<i>r</i>
A	33	11	22	33	44	55	99	66	77	88

Partizione

	<i>p</i>								<i>r</i>	
A	99	33	55	77	11	22	88	66	33	44

<i>i</i>	<i>j</i>								<i>x</i>	
A	99	33	55	77	11	22	88	66	33	44

<i>i</i>		<i>j</i>							<i>x</i>	
A	99	33	55	77	11	22	88	66	33	44

	<i>i</i>		<i>j</i>						<i>x</i>	
A	33	99	55	77	11	22	88	66	33	44

	<i>i</i>			<i>j</i>					<i>x</i>	
A	33	99	55	77	11	22	88	66	33	44

	<i>i</i>				<i>j</i>				<i>x</i>	
A	33	99	55	77	11	22	88	66	33	44

		<i>i</i>				<i>j</i>			<i>x</i>	
A	33	11	55	77	99	22	88	66	33	44

Partizione

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
---	----	----	----	----	----	----	----	----	----	----

Partizione

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
---	----	----	----	----	----	----	----	----	----	----

Partizione

A

	<i>i</i>				<i>j</i>				<i>x</i>
33	11	55	77	99	22	88	66	33	44

A

		<i>i</i>				<i>j</i>			<i>x</i>
33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>					<i>j</i>		<i>x</i>
33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>						<i>j</i>	<i>x</i>
33	11	22	77	99	55	88	66	33	44

A

			<i>i</i>						<i>j</i>
33	11	22	33	99	55	88	66	77	44

A

<i>p</i>				<i>q</i>					<i>r</i>
33	11	22	33	44	55	88	66	77	99

Partizione

A

	<i>i</i>				<i>j</i>				<i>x</i>	
	33	11	55	77	99	22	88	66	33	44

A

		<i>i</i>				<i>j</i>			<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>					<i>j</i>		<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>						<i>j</i>	<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

			<i>i</i>						<i>j</i>	<i>x</i>
	33	11	22	33	99	55	88	66	77	44

A

	<i>p</i>			<i>q</i>					<i>r</i>	
	33	11	22	33	44	55	88	66	77	99

Partizione

A

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

A

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

A

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

A

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

A

			i						j	
	33	11	22	33	99	55	88	66	77	44

A

	p			q					r	
	33	11	22	33	44	55	88	66	77	99

Partizione

A

	<i>i</i>				<i>j</i>				<i>x</i>	
	33	11	55	77	99	22	88	66	33	44

A

		<i>i</i>				<i>j</i>			<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>					<i>j</i>		<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>						<i>j</i>	<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

			<i>i</i>						<i>j</i>	<i>x</i>
	33	11	22	33	99	55	88	66	77	44

A

	<i>p</i>			<i>q</i>					<i>r</i>	
	33	11	22	33	44	55	88	66	77	99

Partizione

A

	<i>i</i>				<i>j</i>				<i>x</i>	
	33	11	55	77	99	22	88	66	33	44

A

		<i>i</i>				<i>j</i>			<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>					<i>j</i>		<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>						<i>j</i>	<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

			<i>i</i>						<i>j</i>	<i>x</i>
	33	11	22	33	99	55	88	66	77	44

A

	<i>p</i>			<i>q</i>					<i>r</i>	
	33	11	22	33	44	55	88	66	77	99

Partizione

A

	<i>i</i>				<i>j</i>				<i>x</i>	
	33	11	55	77	99	22	88	66	33	44

A

		<i>i</i>				<i>j</i>			<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>					<i>j</i>		<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

		<i>i</i>						<i>j</i>	<i>x</i>	
	33	11	22	77	99	55	88	66	33	44

A

			<i>i</i>						<i>j</i>	<i>x</i>
	33	11	22	33	99	55	88	66	77	44

A

	<i>p</i>			<i>q</i>					<i>r</i>	
	33	11	22	33	44	55	88	66	77	99

Particione

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

```
PARTICIONE( $A, p, r$ )
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  $i+1$ 
```

Invariantes:

No começo de cada iteração da linha 3 vale que:

(1) $A[p\dots i] \leq x$ (2) $A[i+1\dots j-1] > x$ (3) $A[r] = x$

Particione

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

```
PARTICIONE( $A, p, r$ )
1   $x \leftarrow A[r]$   ▷  $x$  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  $i+1$ 
```

Invariantes:

No começo de cada iteração da linha 3 vale que:

(1) $A[p\dots i] \leq x$ (2) $A[i+1\dots j-1] > x$ (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 $i+1$?

Se $n := r - p + 1$, então a complexidade no pior caso é

$$T(n) = ?$$

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 $i+1$?

Se $n := r - p + 1$, então a complexidade no pior caso é

$$T(n) = ?$$

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 $i+1$	$\Theta(1)$

Se $n := r - p + 1$, então a complexidade no pior caso é

$$T(n) = ?$$

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 $i+1$	$\Theta(1)$

Se $n := r - p + 1$, então a complexidade no pior caso é

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

QuickSort

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

```
QUICKSORT( $A, p, r$ )  
1  se  $p < r$   
2    então  $q \leftarrow \text{PARTICIONE}(A, p, r)$   
3          QUICKSORT( $A, p, q - 1$ )  
4          QUICKSORT( $A, q + 1, r$ )
```

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

QuickSort

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

```
QUICKSORT( $A, p, r$ )
1  se  $p < r$ 
2  então  $q \leftarrow \text{PARTICIONE}(A, p, r)$ 
3  QUICKSORT( $A, p, q - 1$ )
4  QUICKSORT( $A, q + 1, r$ )
```

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

No começo da linha 3, vale

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

QuickSort

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

```
QUICKSORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \text{PARTICIONE}(A, p, r)$ 
3    QUICKSORT( $A, p, q - 1$ )
4    QUICKSORT( $A, q + 1, r$ )
```

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

QuickSort

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

```
QUICKSORT( $A, p, r$ )  
1  se  $p < r$   
2    então  $q \leftarrow \text{PARTICIONE}(A, p, r)$   
3          QUICKSORT( $A, p, q - 1$ )  
4          QUICKSORT( $A, q + 1, r$ )
```

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

Complexidade de QUICKSORT

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

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n - 1$.

$$T(n) = ?$$

Complexidade de QUICKSORT

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

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n - 1$.

$$T(n) = ?$$

Complexidade de QUICKSORT

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

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n - 1$.

$$T(n) = ?$$

Complexidade de QUICKSORT

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

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n - 1$.

$$T(n) = ?$$

Complexidade de QUICKSORT

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

Seja $n := r - p + 1$ e $k := q - p$, onde $0 \leq k \leq n - 1$.

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

Recorrência

$T(n)$:= consumo de tempo no pior caso

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

Recorrência de um caso (vetor ordenado):

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

$T(n)$ é $\Theta(?)$.

Recorrência

$T(n)$:= consumo de tempo no pior caso

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

Recorrência de um caso (vetor ordenado):

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

$T(n)$ é $\Theta(?)$.

Recorrência

$T(n)$:= consumo de tempo no pior caso

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

Recorrência de um caso (vetor ordenado):

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

$T(n)$ é $\Theta(?)$.

Recorrência

$T(n)$:= consumo de tempo no pior caso

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

Recorrência de um caso (vetor ordenado):

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

$T(n)$ é $\Theta(?)$.

Recorrência

$T(n)$:= consumo de tempo no pior caso

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

Recorrência de um caso (vetor ordenado):

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

$T(n)$ é $\Theta(n^2)$.

Recorrência para o pior caso

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

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

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

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

Recorrência para o pior caso

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

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

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

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

Recorrência para o pior caso

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

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

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

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

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

Demonstração: $T(n) = O(n^2)$

Vou mostrar $T(n) \leq cn^2$ por indução em n (grande).

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

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

Continuação: $T(n) = \Omega(n^2)$

Agora vou mostrar que $T(n) \geq dn^2$ para n grande.

$$\begin{aligned}T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + bn \\&\geq \max_{0 \leq k \leq n-1} \left\{ dk^2 + d(n-k-1)^2 \right\} + bn \\&= d \max_{0 \leq k \leq n-1} \left\{ k^2 + (n-k-1)^2 \right\} + bn \\&= d(n-1)^2 + bn \\&= dn^2 - 2dn + d + bn \\&\geq dn^2,\end{aligned}$$

se $d < b/2$ e $n \geq d/(2d-b)$.

Conclusão

$T(n)$ é $\Theta(n^2)$.

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

Conclusão

$T(n)$ é $\Theta(n^2)$.

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

QuickSort no melhor caso

$M(n)$:= complexidade de tempo no **melhor caso**

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

Mostre que, para $n \geq 1$,

$$M(n) \geq \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

Isto implica que **no melhor caso** o QUICKSORT é $\Omega(n \lg n)$.

QuickSort no melhor caso

$M(n)$:= complexidade de tempo no **melhor caso**

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

Mostre que, para $n \geq 1$,

$$M(n) \geq \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

Isto implica que **no melhor caso** o QUICKSORT é $\Omega(n \lg n)$.

QuickSort no melhor caso

$M(n)$:= complexidade de tempo no **melhor caso**

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

Mostre que, para $n \geq 1$,

$$M(n) \geq \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

Isto implica que **no melhor caso** o QUICKSORT é $\Omega(n \lg n)$.

QuickSort no melhor caso

No melhor caso k é aproximadamente $(n - 1)/2$.

$$R(n) = R\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + R\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$.

Humm, lembra a recorrência do MERGE-SORT...

Mais algumas conclusões

$M(n)$ é $\Theta(n \lg n)$.

O consumo de tempo do QUICKSORT no melhor caso é $\Omega(n \log n)$.

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QUICKSORT no caso médio é mais próximo do melhor caso do que do pior caso.

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QUICKSORT no caso médio é mais próximo do melhor caso do que do pior caso.

Por quê??

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QUICKSORT no caso médio é mais próximo do melhor caso do que do pior caso.

Por quê??

Suponha que (por sorte) o algoritmo PARTICIONE sempre divide o vetor na proporção $\frac{1}{9}$ para $\frac{9}{10}$. Então

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QUICKSORT no caso médio é mais próximo do melhor caso do que do pior caso.

Por quê??

Suponha que (por sorte) o algoritmo PARTICIONE sempre divide o vetor na proporção $\frac{1}{9}$ para $\frac{9}{10}$. Então

$$T(n) = T\left(\left\lfloor \frac{n-1}{9} \right\rfloor\right) + T\left(\left\lceil \frac{9(n-1)}{10} \right\rceil\right) + \Theta(n)$$

Caso médio

Apesar da complexidade de tempo do QUICKSORT no pior caso ser $\Theta(n^2)$, na prática ele é o algoritmo mais eficiente.

Mais precisamente, a complexidade de tempo do QUICKSORT no caso médio é mais próximo do melhor caso do que do pior caso.

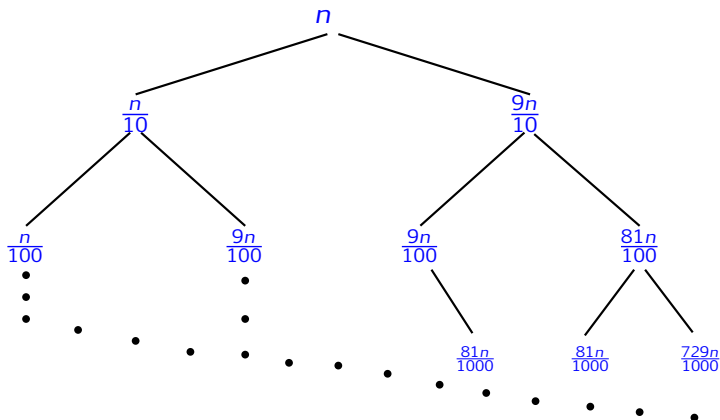
Por quê??

Suponha que (por sorte) o algoritmo PARTICIONE sempre divide o vetor na proporção $\frac{1}{9}$ para $\frac{9}{10}$. Então

$$T(n) = T\left(\left\lfloor \frac{n-1}{9} \right\rfloor\right) + T\left(\left\lceil \frac{9(n-1)}{10} \right\rceil\right) + \Theta(n)$$

Solução: $T(n)$ é $\Theta(n \lg n)$.

Árvore de recorrência

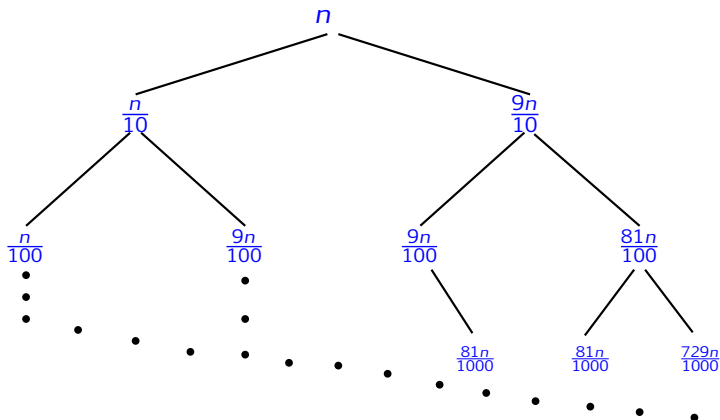


Número de níveis $\leq \log_{10/9} n$.

Em cada nível o custo é $\leq n$.

Custo total é $O(n \log n)$.

Árvore de recorrência

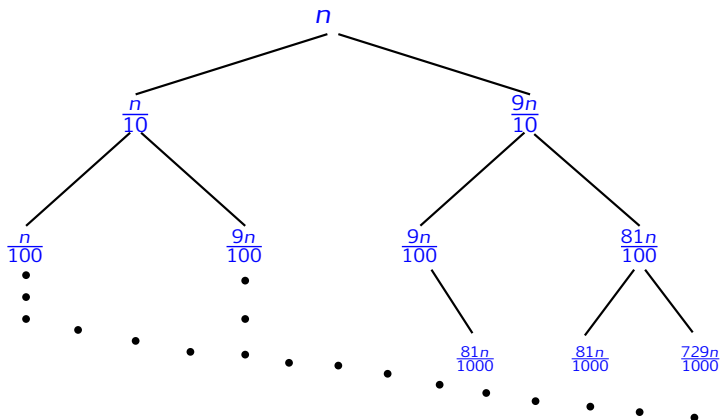


Número de níveis $\leq \log_{10/9} n$.

Em cada nível o custo é $\leq n$.

Custo total é $O(n \log n)$.

Árvore de recorrência

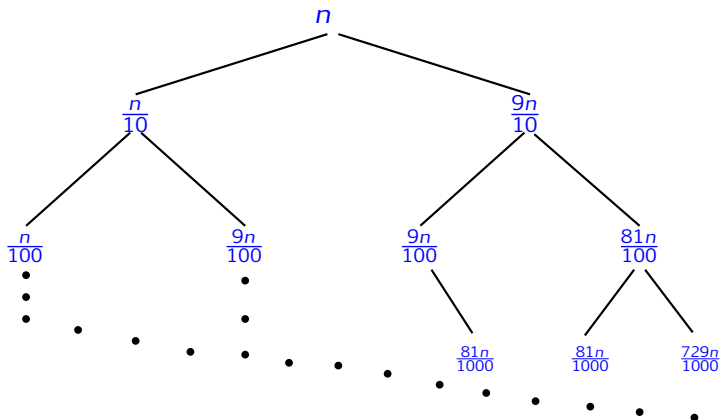


Número de níveis $\leq \log_{10/9} n$.

Em cada nível o custo é $\leq n$.

Custo total é $O(n \log n)$.

Árvore de recorrência



Número de níveis $\leq \log_{10/9} n$.

Em cada nível o custo é $\leq n$.

Custo total é $O(n \log n)$.

QuickSort Aleatório

O **pio**r caso do QUICKSORT ocorre devido a uma escolha infeliz do pivô.

Um modo de minimizar este problema é usar aleatoriedade.

```
PARTICIONE-ALEATÓRIO( $A, p, r$ )
```

```
1   $i \leftarrow \text{RANDOM}(p, r)$ 
```

```
2   $A[i] \leftrightarrow A[r]$ 
```

```
3  devolva PARTICIONE( $A, p, r$ )
```

```
QUICKSORT-ALEATÓRIO( $A, p, r$ )
```

```
1  se  $p < r$ 
```

```
2      então  $q \leftarrow \text{PARTICIONE-ALEATÓRIO}(A, p, r)$ 
```

```
3          QUICKSORT-ALEATÓRIO( $A, p, q - 1$ )
```

```
4          QUICKSORT-ALEATÓRIO( $A, q + 1, r$ )
```

QuickSort Aleatório

O **pior caso** do QUICKSORT ocorre devido a uma escolha infeliz do pivô.

Um modo de minimizar este problema é usar aleatoriedade.

```
PARTICIONE-ALEATÓRIO( $A, p, r$ )
```

```
1   $i \leftarrow \text{RANDOM}(p, r)$ 
```

```
2   $A[i] \leftrightarrow A[r]$ 
```

```
3  devolva PARTICIONE( $A, p, r$ )
```

```
QUICKSORT-ALEATÓRIO( $A, p, r$ )
```

```
1  se  $p < r$ 
```

```
2      então  $q \leftarrow \text{PARTICIONE-ALEATÓRIO}(A, p, r)$ 
```

```
3          QUICKSORT-ALEATÓRIO( $A, p, q - 1$ )
```

```
4          QUICKSORT-ALEATÓRIO( $A, q + 1, r$ )
```

QuickSort Aleatório

O **pior caso** do QUICKSORT ocorre devido a uma escolha infeliz do pivô.

Um modo de minimizar este problema é usar aleatoriedade.

```
PARTICIONE-ALEATÓRIO( $A, p, r$ )
```

```
1  $i \leftarrow \text{RANDOM}(p, r)$ 
```

```
2  $A[i] \leftrightarrow A[r]$ 
```

```
3 devolva PARTICIONE( $A, p, r$ )
```

```
QUICKSORT-ALEATÓRIO( $A, p, r$ )
```

```
1 se  $p < r$ 
```

```
2 então  $q \leftarrow \text{PARTICIONE-ALEATÓRIO}(A, p, r)$ 
```

```
3     QUICKSORT-ALEATÓRIO( $A, p, q - 1$ )
```

```
4     QUICKSORT-ALEATÓRIO( $A, q + 1, r$ )
```

QuickSort Aleatório

O **pior caso** do QUICKSORT ocorre devido a uma escolha infeliz do pivô.

Um modo de minimizar este problema é usar aleatoriedade.

```
PARTICIONE-ALEATÓRIO( $A, p, r$ )
```

```
1   $i \leftarrow \text{RANDOM}(p, r)$ 
```

```
2   $A[i] \leftrightarrow A[r]$ 
```

```
3  devolva PARTICIONE( $A, p, r$ )
```

```
QUICKSORT-ALEATÓRIO( $A, p, r$ )
```

```
1  se  $p < r$ 
```

```
2      então  $q \leftarrow \text{PARTICIONE-ALEATÓRIO}(A, p, r)$ 
```

```
3          QUICKSORT-ALEATÓRIO( $A, p, q - 1$ )
```

```
4          QUICKSORT-ALEATÓRIO( $A, q + 1, r$ )
```

Tempo Esperado do QuickSort

O número esperado de comparações do QuickSort é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são *distintos*. Sejam $z_1 < z_2 < \dots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ e X_{ij} *variável aleatória que indica que z_i foi comparado com z_j* :

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o número total de comparações X é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

O número esperado de comparações do QuickSort é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são **distintos**. Sejam $z_1 < z_2 < \dots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ e X_{ij} **variável aleatória que indica que z_i foi comparado com z_j** :

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o número total de comparações X é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

O número esperado de comparações do QuickSort é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são **distintos**. Sejam $z_1 < z_2 < \dots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ e X_{ij} **variável aleatória que indica que z_i foi comparado com z_j** :

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o número total de comparações X é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

O número esperado de comparações do QuickSort é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são **distintos**. Sejam $z_1 < z_2 < \dots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ e X_{ij} **variável aleatória que indica que z_i foi comparado com z_j** :

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o número total de comparações X é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

O número esperado de comparações do QuickSort é $O(n \log n)$.

Vamos supor que todos os elementos do vetor são **distintos**. Sejam $z_1 < z_2 < \dots < z_n$ os elementos ordenados.

Para $i < j$ seja $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ e X_{ij} **variável aleatória que indica que z_i foi comparado com z_j** :

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ é comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o **número total de comparações X** é

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

e o número esperado de comparações é $E[X]$.

Tempo Esperado do QuickSort

Pela linearidade da esperança:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}]$$

Como X_{ij} é uma variável aleatória binária,

$$\begin{aligned} E[X_{ij}] &= 0 \cdot \Pr\{z_i \text{ não ser comparado com } z_j\} + \\ &\quad 1 \cdot \Pr\{z_i \text{ ser comparado com } z_j\} \\ &= \Pr\{z_i \text{ ser comparado com } z_j\} \end{aligned}$$

Tempo Esperado do QuickSort

Pela linearidade da esperança:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}]$$

Como X_{ij} é uma variável aleatória binária,

$$\begin{aligned} E[X_{ij}] &= 0 \cdot \Pr\{z_i \text{ não ser comparado com } z_j\} + \\ &\quad 1 \cdot \Pr\{z_i \text{ ser comparado com } z_j\} \\ &= \Pr\{z_i \text{ ser comparado com } z_j\} \end{aligned}$$

Tempo Esperado do QuickSort

Considere a escolha do pivô e a comparação entre z_i e z_j :

$$\underbrace{z_1, z_2, \dots, z_{i-1}, z_i}_{\text{posterga}}, \underbrace{z_{i+1}, \dots, z_{j-1}, z_j}_{\text{não comp.}}, \underbrace{z_{j+1}, \dots, z_n}_{\text{posterga}}$$

Assim,

$$\begin{aligned} & \Pr\{z_i \text{ ser comparado com } z_j\} \\ &= \Pr\{z_i \text{ ou } z_j \text{ é escolhido como pivô primeiro em } Z_{ij}\} \\ &= \Pr\{z_i \text{ é escolhido como pivô primeiro em } Z_{ij}\} + \\ & \quad \Pr\{z_j \text{ é escolhido como pivô primeiro em } Z_{ij}\} \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1}. \end{aligned}$$

Tempo Esperado do QuickSort

Portanto,

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\ &< \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} \\ &= \sum_{i=1}^{n-1} O(\lg n) \\ &= O(n \lg n) \end{aligned}$$

Conclusão

O consumo de tempo esperado pelo QUICKSORT-ALEATÓRIO para **itens distintos** é $O(n \lg n)$.

Exercício Mostre que $T(n) = \Omega(n \lg n)$.

Conclusão:

O consumo de tempo esperado pelo QUICKSORT-ALEATÓRIO para **itens distintos** é $\Theta(n \lg n)$.

Exercício Analise o tempo esperado do **QUICKSORT-ALEATÓRIO** quando todos os elementos são iguais.

Exercício Adapte o algoritmo **QUICKSORT-ALEATÓRIO** para executar em tempo esperado $O(n \lg n)$ quando há elementos iguais.

Dica: Faça uma variante do **PARTICIONE-ALEATÓRIO** que considera elementos iguais, dividindo o vetor em três partes, de acordo com o pivô (partes com elementos menores, iguais e maiores que o pivô).