

Análise de Algoritmos

CLRS 7

QuickSort

Rearranja $A[p..r]$ 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( $A, q + 1, r$ )
```

Partição

Problema: Rearranjar um dado vetor $A[p..r]$ e devolver um índice q tal que $p \leq q \leq r$ e

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

Entra:

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

Partição

Problema: Rearranjar um dado vetor $A[p..r]$ e devolver um índice q tal que $p \leq q \leq r$ e

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

Entra:

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

Sai:

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

Partizione

p *r*

A	99	33	55	77	11	22	88	66	33	44
---	----	----	----	----	----	----	----	----	----	----

Partizione

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

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

A	99	33	55	77	11	22	88	66	33	44
---	----	----	----	----	----	----	----	----	----	----

	Partizione									
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

Partizione

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

Partizione

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

Partizione

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	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

33	99	55	77	11	22	88	66	33	44
----	----	----	----	----	----	----	----	----	----

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

Particione

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

Particione

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

Particione

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	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*

A	33	11	22	33	99	55	88	66	77	44
---	----	----	----	----	----	----	----	----	----	----

Partizione

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	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*

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

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 de 3–6,

(i0) $A[p..i] \leq x$ (i1) $A[i+1..j-1] > x$ (i2) $A[r] = x$

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1-2	$= 2\Theta(1)$
3	$= \Theta(n)$
4	$= \Theta(n)$
5-6	$= 2O(n)$
7-8	$= 2\Theta(1)$
total	$= \Theta(2n + 4) + O(2n) = \Theta(n)$

Conclusão:

O algoritmo **PARTICIONE** consome tempo $\Theta(n)$.

QuickSort

Rearranja $A[p..r]$ 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 $A[p..r]$ 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,

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

QuickSort

Rearranja $A[p..r]$ 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 $A[p..r]$ 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

QuickSort

Rearranja $A[p..r]$ 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 ( $A, q + 1, r$ )
```

No começo da linha 3,

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

Consumo de tempo?

QuickSort

Rearranja $A[p..r]$ 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 ( $A, q + 1, r$ )
```

No começo da linha 3,

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

Consumo de tempo?

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

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha	
1	=	?
2	=	?
3	=	?
4	=	?
total	=	????

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha		consumo de todas as execuções da linha
1	=	$\Theta(1)$
2	=	$\Theta(n)$
3	=	$T(k)$
4	=	$T(n - k - 1)$

$$\text{total} = T(k) + T(n - k - 1) + \Theta(n + 1)$$

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

Recorrência

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

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

Recorrência

$T(n)$:= consumo de tempo máximo quando $n = r - p + 1$

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

Recorrência grosseira:

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

$T(n)$ é $\Theta(???)$.

Recorrência

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

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

Recorrência grosseira:

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

$T(n)$ é $\Theta(n^2)$.

Demonstração: ... Exercício!

Recorrência cuidadosa

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

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

Recorrência cuidadosa

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

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

Versão simplificada:

$$T(0) = 1$$

$$T(1) = 1$$

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

n	0	1	2	3	4	5
$T(n)$	1	1	$2 + 2$	$5 + 3$	$9 + 4$	$14 + 5$

Recorrência cuidadosa

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

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

Versão simplificada:

$$T(0) = 1$$

$$T(1) = 1$$

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

n	0	1	2	3	4	5
$T(n)$	1	1	2 + 2	5 + 3	9 + 4	14 + 5

Vamos mostrar que $T(n) \leq n^2 + 1$ para $n \geq 0$.

Demonstração

Prova: Trivial para $n \leq 1$. Se $n \geq 2$ então

$$\begin{aligned} T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + n \\ &\stackrel{\text{hi}}{\leq} \max_{0 \leq k \leq n-1} \left\{ k^2 + 1 + (n-k-1)^2 + 1 \right\} + n \\ &= \dots \\ &= n^2 - n + 3 \\ &\leq n^2 + 1. \end{aligned}$$

Prove que $T(n) \geq \frac{1}{2}n^2$ para $n \geq 0$.

Algumas conclusões

$$T(n) \text{ é } \Theta(n^2).$$

O consumo de tempo do QUICKSORT no pior caso é $O(n^2)$.

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

QuickSort no melhor caso

$M(n) :=$ consumo de tempo **mínimo** quando $n = r - p + 1$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n - k - 1)\} + \Theta(n)$$

QuickSort no melhor caso

$M(n) :=$ consumo de tempo **mínimo** quando $n = r - p + 1$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n - k - 1)\} + \Theta(n)$$

Versão simplificada:

$$M(0) = 1$$

$$M(1) = 1$$

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

QuickSort no melhor caso

$M(n) :=$ consumo de tempo **mínimo** quando $n = r - p + 1$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n - k - 1)\} + \Theta(n)$$

Versão simplificada:

$$M(0) = 1$$

$$M(1) = 1$$

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

Mostre que $M(n) \geq \frac{(n+1)}{2} \lg(n+1)$ para $n \geq 0$.

QuickSort no melhor caso

$M(n) :=$ consumo de tempo **mínimo** quando $n = r - p + 1$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n - k - 1)\} + \Theta(n)$$

Versão simplificada:

$$M(0) = 1$$

$$M(1) = 1$$

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

Mostre que $M(n) \geq \frac{(n+1)}{2} \lg(n+1)$ para $n \geq 0$.

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

QuickSort no melhor caso

$M(n) :=$ consumo de tempo **mínimo** quando $n = r - p + 1$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n - k - 1)\} + \Theta(n)$$

Versão simplificada:

$$M(0) = 1$$

$$M(1) = 1$$

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

Mostre que $M(n) \geq \frac{(n+1)}{2} \lg(n+1)$ para $n \geq 0$.

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

Mais algumas conclusões

$M(n)$ é $\Theta(n \lg n)$.

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

Na verdade ...

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

Apesar do consumo de tempo de pior caso do **QUICKSORT** ser $\Theta(n^2)$, sua performance na prática é comparável (e em geral melhor) a de outros algoritmos cujo consumo de tempo no pior caso é $O(n \lg n)$. Tanto que é usado na prática:

Apesar do consumo de tempo de pior caso do **QUICKSORT** ser $\Theta(n^2)$, sua performance na prática é comparável (e em geral melhor) a de outros algoritmos cujo consumo de tempo no pior caso é $O(n \lg n)$. Tanto que é usado na prática:

NAME

`qsort, qsort_r` - sort an array

SYNOPSIS

```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *))
```

Apesar do consumo de tempo de pior caso do **QUICKSORT** ser $\Theta(n^2)$, sua performance na prática é comparável (e em geral melhor) a de outros algoritmos cujo consumo de tempo no pior caso é $O(n \lg n)$. Tanto que é usado na prática:

NAME

```
qsort, qsort_r - sort an array
```

SYNOPSIS

```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *))
```

Por que isso acontece?

QuickSort é bom na média!

Apesar do consumo de tempo de pior caso do **QUICKSORT** ser $\Theta(n^2)$, sua performance na prática é comparável (e em geral melhor) a de outros algoritmos cujo consumo de tempo no pior caso é $O(n \lg n)$. Tanto que é usado na prática:

NAME

`qsort, qsort_r` - sort an array

SYNOPSIS

```
#include <stdlib.h>
```

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *))
```

Por que isso acontece?

Exercício

Considere a recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n$$

para $n = 2, 3, 4, \dots$

Solução assintótica: $T(n)$ é $O(???)$, $T(n)$ é $\Theta(???)$

Exercício

Considere a recorrência

$$T(1) = 1$$

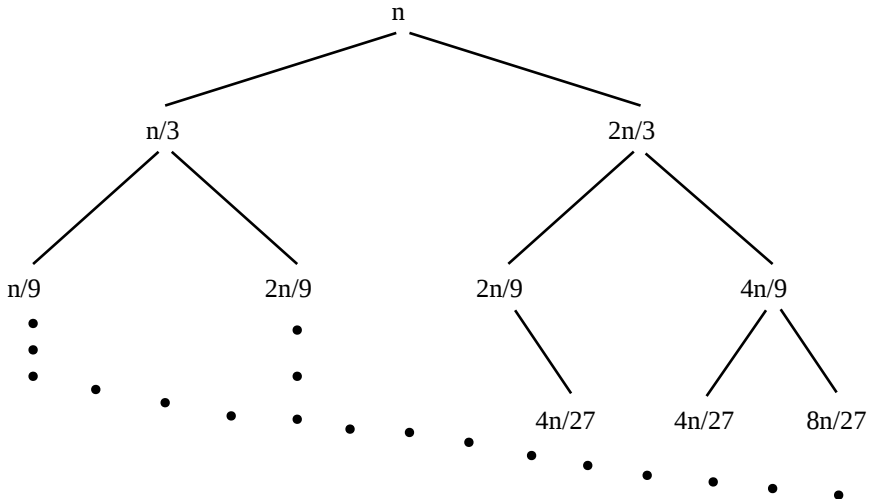
$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n$$

para $n = 2, 3, 4, \dots$

Solução assintótica: $T(n)$ é $O(???)$, $T(n)$ é $\Theta(???)$

Vamos olhar a **árvore da recorrência**.

Árvore da recorrência



total de níveis $\leq \log_{3/2} n$

Árvore da recorrência

soma em cada horizontal $\leq n$

número de “níveis” $\leq \log_{3/2} n$

$T(n)$ = a soma de tudo

$$T(n) \leq n \log_{3/2} n + \underbrace{1 + \dots + 1}_{\log_{3/2} n}$$

$T(n)$ é $O(n \lg n)$.

De volta à recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n \text{ para } n = 2, 3, 4, \dots$$

n	$T(n)$
1	1
2	$1 + 1 + 2 = 4$
3	$1 + 4 + 3 = 8$
4	$4 + 4 + 4 = 12$

De volta à recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n \text{ para } n = 2, 3, 4, \dots$$

n	$T(n)$
1	1
2	$1 + 1 + 2 = 4$
3	$1 + 4 + 3 = 8$
4	$4 + 4 + 4 = 12$

Vamos mostrar que $T(n) \leq 20n \lg n$ para $n = 2, 3, 4, 5, 6, \dots$

De volta à recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n \text{ para } n = 2, 3, 4, \dots$$

n	$T(n)$
1	1
2	$1 + 1 + 2 = 4$
3	$1 + 4 + 3 = 8$
4	$4 + 4 + 4 = 12$

Vamos mostrar que $T(n) \leq 20n \lg n$ para $n = 2, 3, 4, 5, 6, \dots$

Para $n = 2$ temos $T(2) = 4 < 20 \cdot 2 \cdot \lg 2$.

Para $n = 3$ temos $T(3) = 8 < 20 \cdot 3 \cdot \lg 3$.

Suponha agora que $n > 3$. Então...

Continuação da prova

$$\begin{aligned}T(n) &= T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + n \\&\stackrel{\text{hi}}{\leq} 20\left\lceil \frac{n}{3} \right\rceil \lg \left\lceil \frac{n}{3} \right\rceil + 20\left\lfloor \frac{2n}{3} \right\rfloor \lg \left\lfloor \frac{2n}{3} \right\rfloor + n \\&\leq 20\frac{n+2}{3} \left\lceil \lg \frac{n}{3} \right\rceil + 20\frac{2n}{3} \lg \frac{2n}{3} + n \\&< 20\frac{n+2}{3} \left(\lg \frac{n}{3} + 1\right) + 20\frac{2n}{3} \lg \frac{2n}{3} + n \\&= 20\frac{n+2}{3} \lg \frac{2n}{3} + 20\frac{2n}{3} \lg \frac{2n}{3} + n \\&= 20\frac{n}{3} \lg \frac{2n}{3} + 20\frac{2}{3} \lg \frac{2n}{3} + 20\frac{2n}{3} \lg \frac{2n}{3} + n\end{aligned}$$

Continuação da continuação da prova

$$< 20n \lg \frac{2n}{3} + 14 \lg \frac{2n}{3} + n$$

$$= 20n \lg n + 20n \lg \frac{2}{3} + 14 \lg n + 14 \lg \frac{2}{3} + n$$

$$< 20n \lg n + 20n(-0.58) + 14 \lg n + 14(-0.58) + n$$

$$< 20n \lg n - 11n + 14 \lg n - 8 + n$$

$$= 20n \lg n - 10n + 14 \lg n - 8$$

$$< 20n \lg n - 10n + 7n - 8$$

$$< 20n \lg n$$

liiéééééssss!

De volta à intuição

Certifique-se que a conclusão seria a mesma qualquer que fosse a proporção fixa que tomássemos. Por exemplo, resolva o seguinte...

De volta à intuição

Certifique-se que a conclusão seria a mesma qualquer que fosse a proporção fixa que tomássemos. Por exemplo, resolva o seguinte...

Exercício: Considere a recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/10 \rceil) + T(\lfloor 9n/10 \rfloor) + n$$

para $n = 2, 3, 4, \dots$ e mostre que $T(n)$ é $O(n \lg n)$.

De volta à intuição

Certifique-se que a conclusão seria a mesma qualquer que fosse a proporção fixa que tomássemos. Por exemplo, resolva o seguinte...

Exercício: Considere a recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/10 \rceil) + T(\lfloor 9n/10 \rfloor) + n$$

para $n = 2, 3, 4, \dots$ e mostre que $T(n)$ é $O(n \lg n)$.

Note que, se o **QUICKSORT** fizer uma “boa” partição a cada, digamos, 5 níveis da recursão, o efeito geral é o mesmo, assintoticamente, que ter feito uma boa partição em todos os níveis.

Próxima aula

Análise probabilística

CLRS 5.1, 5.2, C.1 a C.3, 7.1 e 7.2