AULA 11

Limites inferiores

CLRS 8.1

Máximo: limite inferior

Problema: Encontrar o maior elemento de um vetor $A[1\,..\,n]$.

Existe um algoritmo que faz o serviço com $n-1$ comparações.

Máximo: limite inferior

Problema: Encontrar o maior elemento de um vetor $A[1\,..\,n]$.

Existe um algoritmo que faz o serviço com $n-1$ comparações.

Existe um algoritmo que faz menos comparações?

Máximo: limite inferior

Problema: Encontrar o maior elemento de um vetor $A[1\,..\,n]$.

Existe um algoritmo que faz o serviço com $n-1$ comparações.

Existe um algoritmo que faz menos comparações?

Não, se o algoritmo é baseado em comparações:

> dados dois número $A[i]$ e $A[j]$ podemos apenas compará-los a fim de encontrar o maior elemento.

Suponha dado um algoritmo baseado em comparações que resolve o problema.

Algoritmo baseado em comparações

O algoritmo consiste, no fundo, na determinação de uma coleção $\mathcal{A}$ de pares ou arcos $\langle i, j \rangle$ de elementos distintos em $\{1, \ldots, n\}$ tais que $A[i] < A[j]$ e existe um "sorvedouro".

Eis o paradigma de todo algoritmo baseado em comparações:

MAX (A, n)
    $\mathcal{A} \leftarrow \emptyset$
    enquanto $\mathcal{A}$ "não possui sorvedouro" **faça**
        escolha índice i e j em $\{1, \ldots, n\}$
        se $A[i] < A[j]$
            então $\mathcal{A} \leftarrow \mathcal{A} \cup \langle i, j \rangle$
            senão $\mathcal{A} \leftarrow \mathcal{A} \cup \langle j, i \rangle$
    devolva $\mathcal{A}$

Conclusão

Qualquer conjunto $\mathcal{A}$ devolvido pelo método contém uma "árvore enraizada" e portanto contém pelo menos $n-1$ arcos.

Qualquer algoritmo baseado em comparações que encontra o maior elemento de um vetor $A[1\,..\,n]$ faz **pelo menos $n-1$** comparações.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

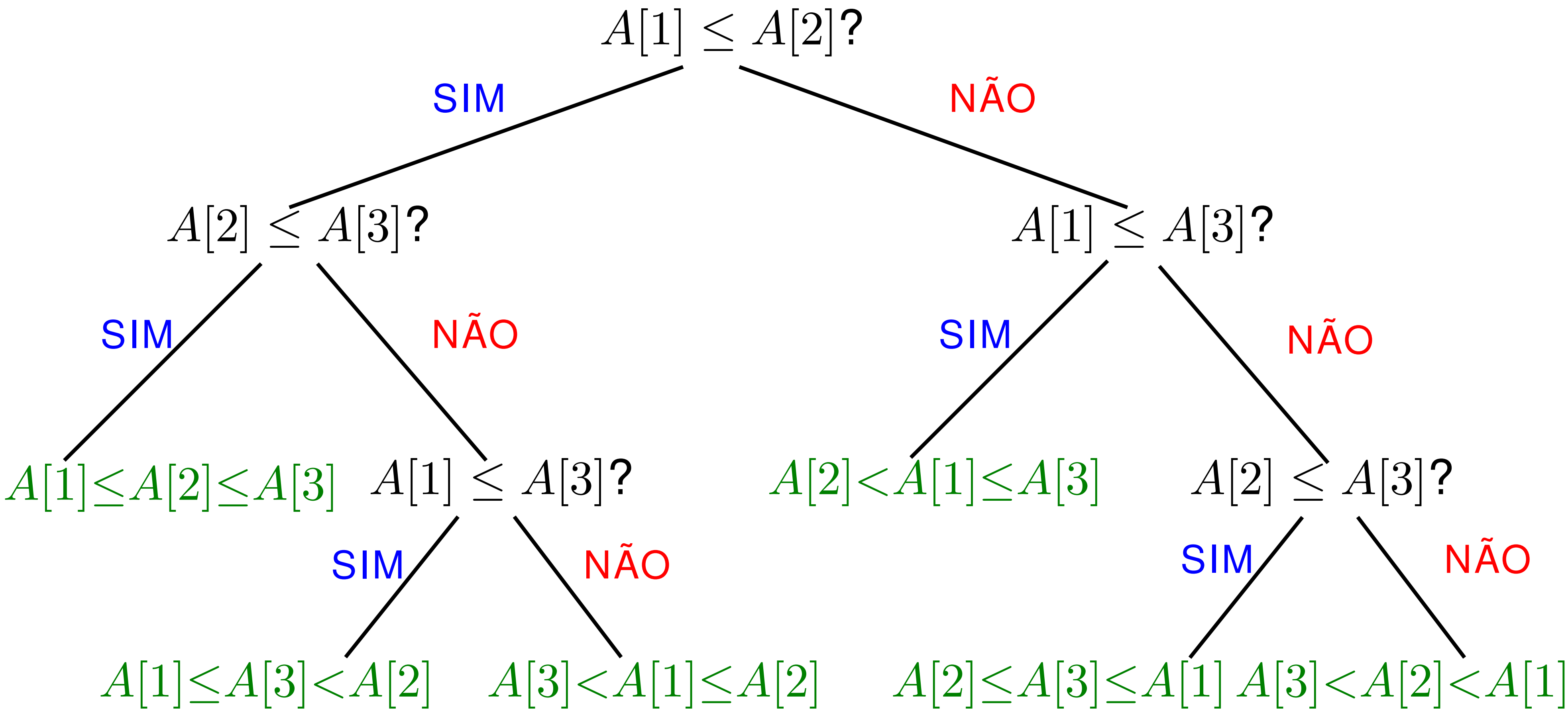
NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações é uma "árvore de decisão".

Exemplo

ORDENA-POR-INSERÇÃO $(A[1\,.\,.\,3])$:



Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Qual é o número de comparações, no pior caso?

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1\,.\,.\,n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: …

Limite inferior

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h.

A afirmação vale para $h = 0$

Suponha que a afirmação vale para toda árvore binária de altura menor que h, $h \geq 1$.

O número de folhas de uma árvore de altura h é a soma do número de folhas de suas sub-árvores; que têm altura $\leq h - 1$.

Logo, o número de folhas de uma árvore de altura h é não superior a

$$2 \times 2^{h-1} = 2^h.$$

Limite inferior

Logo, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 \ = \ \prod_{i=0}^{n-1} (n-i)(i+1) \ \geq \ \prod_{i=1}^{n} n \ = \ n^n$$

Portanto,

$$h \ \geq \ \lg(n!) \ \geq \ \lg n^{\frac{n}{2}} \ = \ \frac{1}{2}\, n \lg n.$$

Conclusão

Todo algoritmo de ordenação baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Exercícios

Exercício 16.A
Desenhe a árvore de decisão para o SELECTION-SORT aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 16.B [CLRS 8.1-1]
Qual o menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

Exercício 16.C [CLRS 8.1-2]
Mostre que $\lg(n!) = \Omega(n \lg n)$ sem usar a fórmula de Stirling. Sugestão: Calcule $\sum_{k=n/2}^{n} \lg k$. Use as técnicas de CLRS A.2.

Primos de O

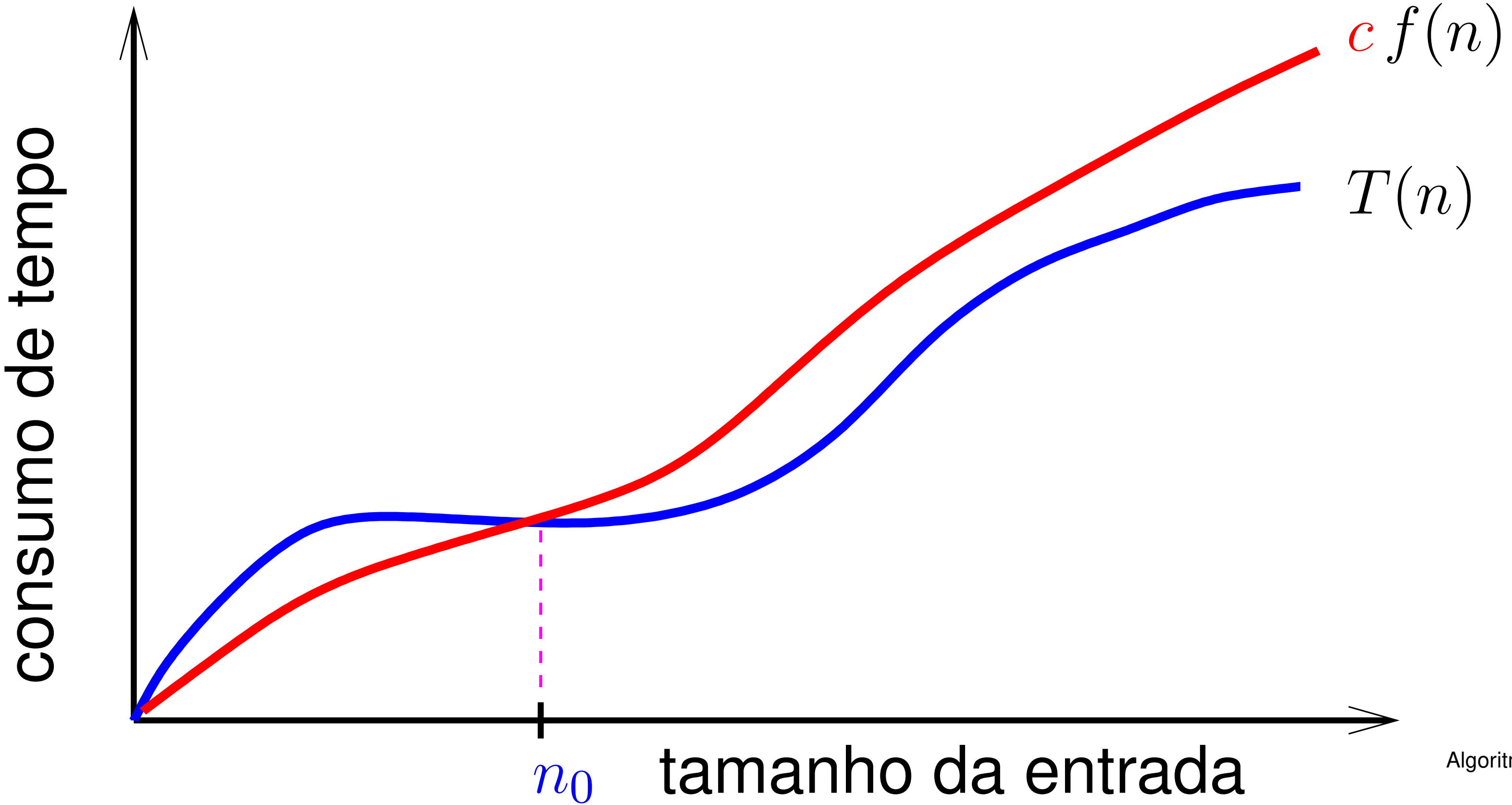
CLRS 3.1

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros nos reais.
Dizemos que $T(n)$ é $\mathrm{o}(f(n))$ se **PARA TODA** constante positiva c **EXISTE** uma constante inteira n_0 tal que

$$T(n) < c\,f(n)$$

para todo $n \geq n_0$.



Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\mathrm{o}(n^3)$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\mathrm{o}(n^3)$.

Prova: Seja $c > 0$ uma constante.

Tome $n_0 := \lceil (12 + 6 + 1)/c \rceil$.

Temos que

$$12n^2 + 6n < 12n^2 + 6n^2 = (12 + 6)n^2 < c\, n^3$$

para todo $n \geq n_0$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\mathrm{o}(n^2)$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 2
$12n^2 + 6n$ não é $\mathrm{o}(n^2)$.

Prova: Para $c := 12$ temos que

$$12n^2 + 6n \geq 12n^2 = c\,n^2$$

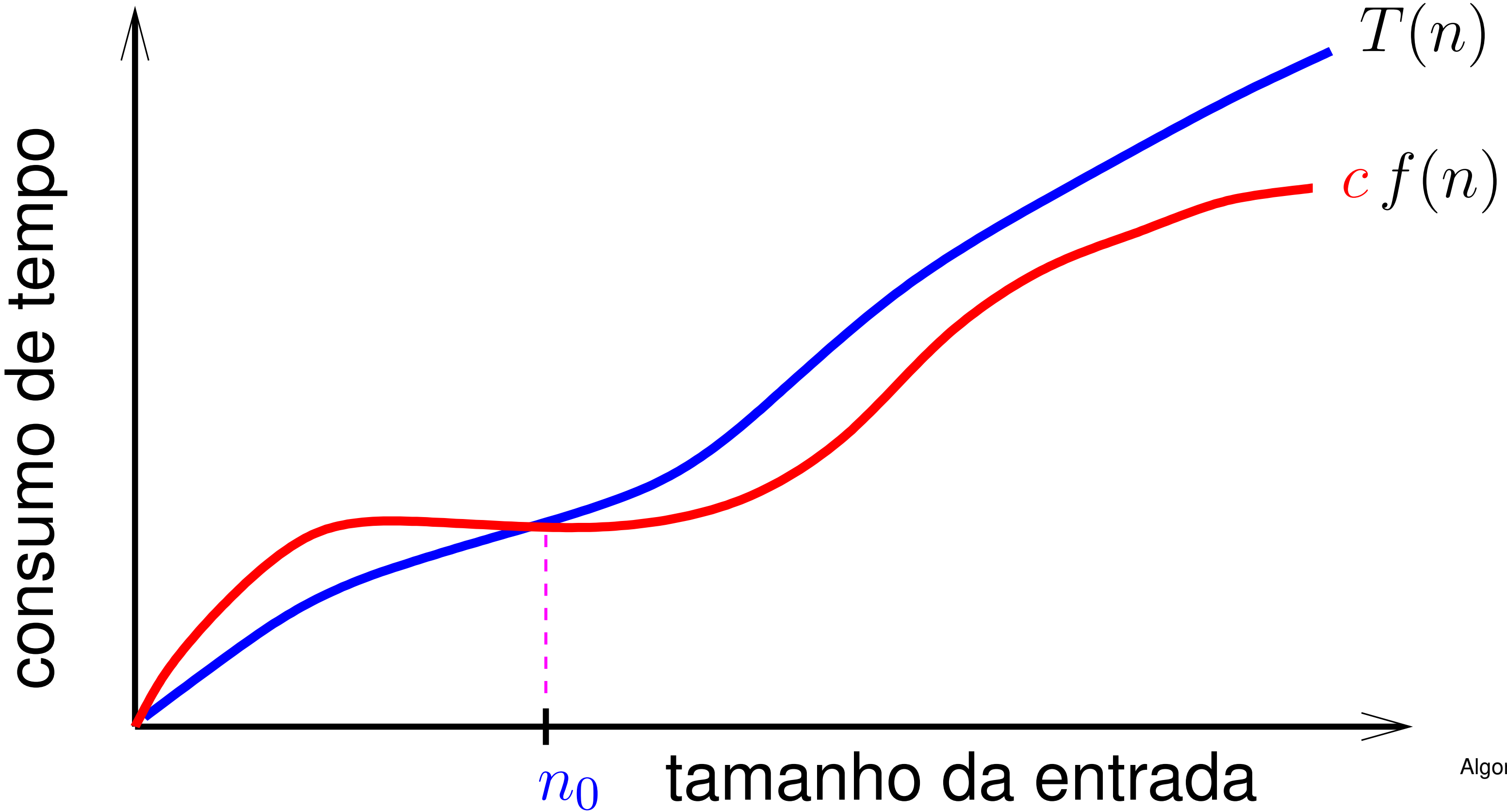
para todo $n \geq 0$.

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ é $\omega(f(n))$ se para **PARA TODA** constante positiva c **EXISTE** uma constante inteira n_0 tal que

$$T(n) \; > \; c\,f(n)$$

para todo $n \geq n_0$.



Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\omega(n)$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\omega(n)$.

Prova: Seja $c > 0$ uma constante.

Tome $n_0 := \lceil c/12 \rceil$.

Temos que

$$12n^2 + 6n > 12n^2 \geq c\,n$$

para todo $n \geq n_0$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\omega(n^2)$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\omega(n^2)$.

Prova: Para $c := 18$ temos que

$$12n^2 + 6n \leq 18\, n^2 = c\, n^2$$

para todo $n \geq 0$.

Intuitivamente

Comparação assintótica, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$
$T(n) < f(n)$	$T(n)$ é $\mathrm{o}(f(n))$
$T(n) > f(n)$	$T(n)$ é $\omega(f(n))$

Limite inferior para ordenação

Não existe algoritmo de ordenação baseado em comparações que faça

$$o(n \lg n)$$

comparações.

Não existe algoritmo de ordenação baseado em comparações que consuma tempo

$$o(n \lg n).$$

Ordenação em tempo linear

CLRS 8.2–8.3

Ordenação por contagem

Recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ os elementos de $A[1\,..\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

Entra:

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
A & 2 & 5 & 3 & 0 & 2 & 3 & 0 & 5 & 3 & 0 \\
\hline
\end{array}
$$

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
B & & & & & & & & & & \\
\hline
\end{array}
$$

Ordenação por contagem

Recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ os elementos de $A[1\,..\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

Entra:

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

Sai:

	1	2	3	4	5	6	7	8	9	10
B	0	0	0	2	2	3	3	3	5	5

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C						

Ordenação por contagem

	1	2	3	4	5	6	7	8	9	10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	0	0	0	0

Ordenação por contagem

	j									
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	1	0	0	0

Ordenação por contagem

		j								
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	1	0	0	1

Ordenação por contagem

			j							
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	0	0	1	1	0	1

Ordenação por contagem

			j						

A

2	5	3	0	2	3	0	5	3	0

B

1	2	3	4	5	6	7	8	9	10

C

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

Ordenação por contagem

					j					
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	1	0	2	1	0	1

Ordenação por contagem

						j				
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	1	0	2	2	0	1

Ordenação por contagem

							j			
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	2	0	1

Ordenação por contagem

								j		
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	2	0	2

Ordenação por contagem

									j	
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	2	0	2	3	0	2

Ordenação por contagem

										j
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	0	2	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	0	2	3	0	2

Ordenação por contagem

A

1									10
2	5	3	0	2	3	0	5	3	0

B

1	2	3	4	5	6	7	8	9	10

C

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

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	3	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	0	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	2

Ordenação por contagem

	1									10
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	10

Ordenação por contagem

										j
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C	3	3	5	8	8	10

Ordenação por contagem

									j	
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B			0							

	0	1	2	3	4	5
C	2	3	5	8	8	10

Ordenação por contagem

								j		
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B			0					3		

	0	1	2	3	4	5
C	2	3	5	7	8	10

Ordenação por contagem

							j			
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B			0					3		5

	0	1	2	3	4	5
C	2	3	5	7	8	9

Ordenação por contagem

						j				
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0					3		5

	0	1	2	3	4	5
C	1	3	5	7	8	9

Ordenação por contagem

					j					
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0				3	3		5

	0	1	2	3	4	5
C	1	3	5	6	8	9

Ordenação por contagem

				j						
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B		0	0		2		3	3		5

	0	1	2	3	4	5
C	1	3	4	6	8	9

Ordenação por contagem

			j							
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2		3	3		5

	0	1	2	3	4	5
C	0	3	4	6	8	9

Ordenação por contagem

		j								
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2	3	3	3		5

	0	1	2	3	4	5
C	0	3	4	5	8	9

Ordenação por contagem

	j									
A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0		2	3	3	3	5	5

	0	1	2	3	4	5
C	0	3	4	5	8	8

Ordenação por contagem

j

A	2	5	3	0	2	3	0	5	3	0

	1	2	3	4	5	6	7	8	9	10
B	0	0	0	2	2	3	3	3	5	5

	0	1	2	3	4	5
C	0	3	3	5	8	8

Ordenação por contagem

COUNTING-SORT (A, B, n, k)

para $i \leftarrow 0$ **até** k **faça**
    $C[i] \leftarrow 0$

para $j \leftarrow 1$ **até** n **faça**
    $C[A[j]] \leftarrow C[A[j]] + 1$
  $\triangleright$ $C[i]$ é o número de js tais que $A[j] = i$

para $i \leftarrow 1$ **até** k **faça**
    $C[i] \leftarrow C[i] + C[i-1]$
  $\triangleright$ $C[i]$ é o número de js tais que $A[j] \leq i$

para $j \leftarrow n$ **decrescendo até** 1 **faça**
    $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Consumo de tempo

linha	consumo na linha
1–2	$\Theta(k)$
3–4	$\Theta(n)$
5–6	$\Theta(k)$
7–9	$\Theta(n)$

Consumo total: $\Theta(n+k)$

Conclusões

O consumo de tempo do COUNTING-SORT é $\Theta(n+k)$.

- se $k \leq n$ então consumo é $\Theta(n)$
- se $k \leq 10n$ então consumo é $\Theta(n)$
- se $k = \mathrm{O}(n)$ então consumo é $\Theta(n)$
- se $k \geq n^2$ então consumo é $\Theta(k)$
- se $k = \Omega(n)$ então consumo é $\Theta(k)$

Estabilidade

A propósito: COUNTING-SORT é estável:

na saída, chaves com mesmo valor estão na mesma ordem que apareciam na entrada.

A	2	5	3	0	2	3	0	5	3	0
	1	2	3	4	5	6	7	8	9	10
B	0	0	0	2	2	3	3	3	5	5

Ordenação digital (=radix sort)

Exemplo:

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Cada $A[j]$ têm d dígitos decimais:
$A[j] = a_d\,10^{d-1} + \cdots + a_2\,10^1 + a_1\,10^0$

Exemplo com $d = 3$: $\quad 3 \cdot 10^2 + 2 \cdot 10 + 9$

Ordenação digital

```
RADIX-SORT (A, n, d)
1    para i ← 1 até d faça
2        ▷ 1 até d e não o contrário!
3        ordene A[1 . . n] pelo dígito i
```

Linha 3:

- faz ordenação $A[j_1 \,..\, j_n]$ de $A[1 \,..\, n]$ tal que

$$A[j_1]_i \leq \cdots \leq A[j_n]_i;$$

- ordenação deve ser estável; e
- use COUNTING-SORT.

Exemplos

- dígitos decimais: $\Theta(dn)$
- dígitos em $0\,..\,k$: $\Theta(d\,(n+k))$.

Exemplo com $d = 5$ e $k = 128$:

$$a_5 128^4 + a_4 128^3 + a_3 128^2 + a_2 128 + a_1$$

sendo $0 \leq a_i \leq 127$

Conclusão

Dados n números com b bits e um inteiro $r \leq b$, RADIX-SORT ordena esses números em tempo

$$\Theta\Big(\frac{b}{r}(n + 2^r)\Big).$$

Mais experimentação

Os programas foram executados na

SunOS rebutosa 5.7 Generic_106541-04 sun4d sparc.

Os códigos foram compilados com o

gcc version 2.95.2 19991024 (release)

e opção de compilação

-Wall -ansi -pedantic -O2.

Algoritmos implementados:

`merger`	MERGE-SORT recursivo
`mergei`	MERGE-SORT iterativo
`heap`	HEAPSORT
`quick`	QUICKSORT recursivo.
`qCLR`	QUICKSORT do CLR.
`radix`	RADIX-SORT

Resultados

n	merger	mergei	heap	quick	qCLR	radix
4096	0.02	0.03	0.01	0.01	0.01	0.01
8192	0.05	0.06	0.03	0.02	0.02	0.03
16384	0.12	0.13	0.07	0.06	0.06	0.06
32768	0.24	0.27	0.15	0.11	0.11	0.11
65536	0.52	0.58	0.33	0.25	0.23	0.22
131072	1.10	1.25	0.75	0.58	0.48	0.49
262144	2.37	2.64	1.71	1.39	0.98	0.98
524288	5.10	5.57	5.07	3.91	1.93	2.06
1048576	10.86	11.77	14.07	10.12	4.32	4.44
2097152	22.71	24.45	37.48	26.61	9.05	10.61
4194304	47.63	52.23	90.99	75.76	19.19	23.98
8388608	99.90	108.73	214.78	231.30	39.91	44.70

Código

```
#define BITSWORD    32
#define BITSBYTE    8
#define K           256
#define BYTESWORD   4
#define R           (1 << BITSBYTE)
#define digit(k,d)
     (((k)>>(BITSWORD-(d)*BITSBYTE))&(R-1))
void
radix_sort(int *v, int l, int r)
{
  int i; /* digito */

  for (i = BYTEWORD-1; i >= 0; i--)
    {
      count_sort(v, l, r+1, i);
    }
}
```

Exercícios

Exercício 17.A

O seguinte algoritmo promete rearranjar o vetor $A[1\,.\,.\,n]$ em ordem crescente supondo que cada $A[i]$ está em $\{0,\ldots,k\}$. O algoritmo está correto?

C-Sort (A, n, k)

para $i \leftarrow 0$ **até** k **faça**
    $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
    $C[A[j]] \leftarrow C[A[j]] + 1$
$j \leftarrow 1$
para $i \leftarrow 0$ **até** k **faça**
    enquanto $C[i] > 0$ **faça**
        $A[j] \leftarrow i$
        $j \leftarrow j + 1$
        $C[i] \leftarrow C[i] - 1$

Qual o consumo de tempo do algoritmo?

Mais exercícios

Exercício 17.B

O seguinte algoritmo promete rearranjar o vetor $A[1 \mathinner{.\,.} n]$ em ordem crescente supondo que cada $A[j]$ está em $\{1, \ldots, k\}$. O algoritmo está correto? Estime, em notação O, o consumo de tempo do algoritmo.

VITO-SORT (A, n, k)

$i \leftarrow 1$
para $a \leftarrow 1$ **até** $k - 1$ **faça**
    para $j \leftarrow i$ **até** n **faça**
        se $A[j] = a$
            então $A[j] \leftrightarrow A[i]$ $\quad\triangleright$ troca
                $i \leftarrow i + 1$

Exercício 17.C

Suponha que os components do vetor $A[1 \mathinner{.\,.} n]$ estão todos em $\{0, 1\}$. Prove que $n - 1$ comparações são suficientes para rearranjar o vetor em ordem crescente.

Exercício 17.D

Qual a principal invariante do algoritmo RADIX-SORT?