Ordenação em tempo linear

CLRS cap 8

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?

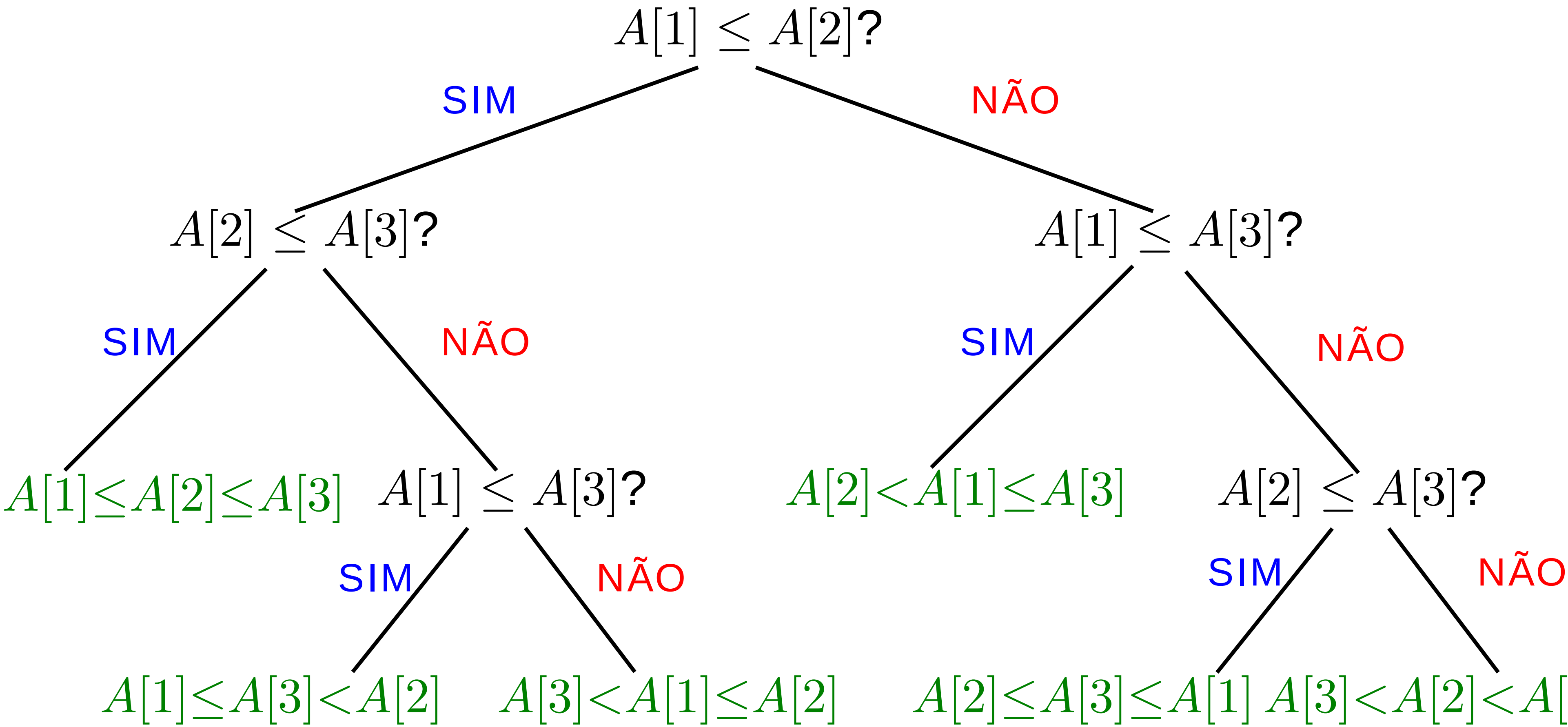
NÃO, se o algoritmo é baseado em comparações.

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 $A[1\,.\,.\,n]$.

Número de comparações, no pior caso?

Limite inferior

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

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1 \mathinner{..} n]$.

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 $A[1 \, . \, . \, n]$.

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.

Limite inferior

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

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: Por indução em h. A afirmação vale para $h = 0$.
Suponha que a afirmação vale para toda árvore binária de altura menor que h, para $h \geq 1$.
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

Assim, 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,

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

Alternativamente, a fórmula de Stirling diz que

$$n! \;=\; \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right).$$

Disso, temos que $h \;\geq\; \lg(n!) \;\geq\; \lg\left(\frac{n}{e}\right)^n \;=\; n(\lg n - \lg e)$.

Conclusão

Todo algoritmo de ordenação baseado em comparações faz

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

comparações no pior caso

Counting Sort

Recebe inteiros n e k, e um vetor $A[1 \,.\,.\, n]$ onde cada elemento é um inteiro entre 1 e k.

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Devolve um vetor $B[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Devolve um vetor $B[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

COUNTINGSORT(A, n)

para $i \leftarrow 1$ **até** k **faça**
  $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $C[A[j]] \leftarrow C[A[j]] + 1$
para $i \leftarrow 2$ **até** k **faça**
  $C[i] \leftarrow C[i] + C[i-1]$
para $j \leftarrow n$ **decrescendo até** 1 **faça**
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
devolva B

Consumo de tempo

linha	consumo na linha
1	$\Theta(k)$
2	$O(k)$
3	$\Theta(n)$
4	$O(n)$
5	$\Theta(k)$
6	$O(k)$
7	$\Theta(n)$
8	$O(n)$
9	$O(n)$
10	$\Theta(1)$
total	????

Consumo de tempo

linha	consumo na linha
1	$\Theta(k)$
2	$\mathrm{O}(k)$
3	$\Theta(n)$
4	$\mathrm{O}(n)$
5	$\Theta(k)$
6	$\mathrm{O}(k)$
7	$\Theta(n)$
8	$\mathrm{O}(n)$
9	$\mathrm{O}(n)$
10	$\Theta(1)$
total	$\Theta(k + n)$

Counting Sort

COUNTINGSORT(A, n)

para $i \leftarrow 1$ **até** k **faça**
  $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $C[A[j]] \leftarrow C[A[j]] + 1$
para $i \leftarrow 2$ **até** k **faça**
  $C[i] \leftarrow C[i] + C[i-1]$
para $j \leftarrow n$ **decrescendo até** 1 **faça**
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
devolva B

Consumo de tempo: $\Theta(k+n)$

Se $k = \mathrm{O}(n)$, o consumo de tempo é $\Theta(n)$.

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

RADIXSORT(A, n, d)
1 **para** $i \leftarrow 1$ **até** d **faça**
2 ORDENE(A, n, i)

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

RADIXSORT(A, n, d)
1 **para** $i \leftarrow 1$ **até** d **faça**
2 ORDENE(A, n, i)

ORDENE(A, n, i): ordena $A[1\,..\,n]$ pelo i-ésimo dígito dos números em A por meio de um algoritmo estável.

Consumo de tempo

Depende do algoritmo ORDENE.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Se d é limitado por uma constante (ou seja, se $d = \mathrm{O}(1)$)
e $k = \mathrm{O}(n)$, então o consumo de tempo é $\Theta(n)$.

Bucket Sort

Recebe um inteiro n e um vetor $A[1 \,..\, n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

Bucket Sort

Recebe um inteiro n e um vetor $A[1 \,.\,.\, n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1 \,.\,.\, n]$ com
os elementos de $A[1 \,.\,.\, n]$ em ordem crescente.

BUCKETSORT(A, n)
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**

    $B[i] \leftarrow$ NIL

para $i \leftarrow 1$ **até** n **faça**

    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$

para $i \leftarrow 0$ **até** $n - 1$ **faça**

    ORDENELISTA$(B[i])$

$C \leftarrow$ CONCATENE(B, n)

devolva C

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**

  $B[i] \leftarrow \textsc{nil}$

para $i \leftarrow 1$ **até** n **faça**

  INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$

para $i \leftarrow 0$ **até** $n - 1$ **faça**

  ORDENELISTA$(B[i])$

$C \leftarrow$ CONCATENE(B, n)

devolva C

INSIRA(p, x): insere x na lista apontada por p

ORDENELISTA(p): ordena a lista apontada por p

CONCATENE(B, n): devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Se os números em $A[1 \mathinner{..} n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o consumo de tempo esperado é linear em n.

Bucket Sort

Se os números em $A[1\,..\,n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o número esperado de elementos de $A[1\,..\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Bucket Sort

Se os números em $A[1\,..\,n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o número esperado de elementos de $A[1\,..\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Logo, o consumo de tempo esperado para ordenar cada uma das listas $B[i]$ é linear $\Theta(1)$.

Assim, o consumo de tempo esperado do algoritmo BUCKETSORT neste caso é $\Theta(n)$.

Bucket Sort

Se os números em $A[1\,..\,n]$ forem uniformemente distribuídos no intervalo $[0,1)$, então o número esperado de elementos de $A[1\,..\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Logo, o consumo de tempo esperado para ordenar cada uma das listas $B[i]$ é linear $\Theta(1)$.

Assim, o consumo de tempo esperado do algoritmo BUCKETSORT neste caso é $\Theta(n)$.

Exercícios

Exercício 10.A

Desenhe a árvore de decisão para o SELECTIONSORT aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 10.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 10.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.

Exercícios

Exercício 10.D [CLRS 8.2-1]
Simule a execução do COUNTINGSORT usando como entrada o vetor $A[1\,..\,11] = \langle 7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3 \rangle$.

Exercício 10.E [CLRS 8.2-2]
Mostre que o COUNTINGSORT é estável.

Exercício 10.F [CLRS 8.2-3]
Suponha que o **para** da linha 7 do COUNTINGSORT é substituído por

7 **para** $j \leftarrow 1$ **até** n **faça**

Mostre que o ainda funciona. O algoritmo resultante continua estável?