Aula 12: Algoritmos de ordenação com complexidade linear

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

25 de março de 2015

Download me from `http://DavidDeharbe.github.io`.

Plano da aula

Introdução

Limite teórico dos algoritmos de ordenação

Ordenação por contadores

Ordenação por algarismos

Ordenação por lotes

Síntese

PPgSC

Introdução

Ref: Cormen et al. Capítulo 9.

- Limite inferior do pior caso dos algoritmos de ordenação
- Algoritmos de complexidade linear
  - Ordenação por contadores (*counting sort*)
  - Ordenação por algarismos (*radix sort*)
  - Ordenação por lotes (*bucket sort*)

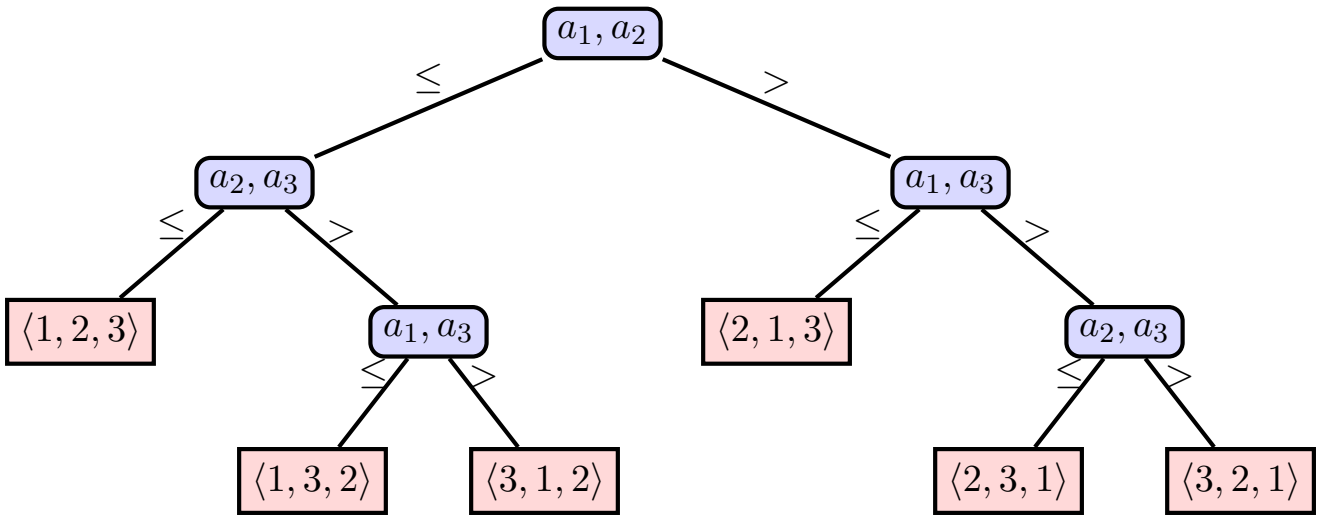
PPgSC

Limite inferior da ordenação

- Número mínimo de comparações necessárias para ordenar um arranjo de tamanho n.
- Hipóteses:
  - todos os elementos são diferentes;
  - a comparação é $\leq$.
- Modelo de árvores de decisão.

Árvores de decisão

Ordenação por inserção com três elementos

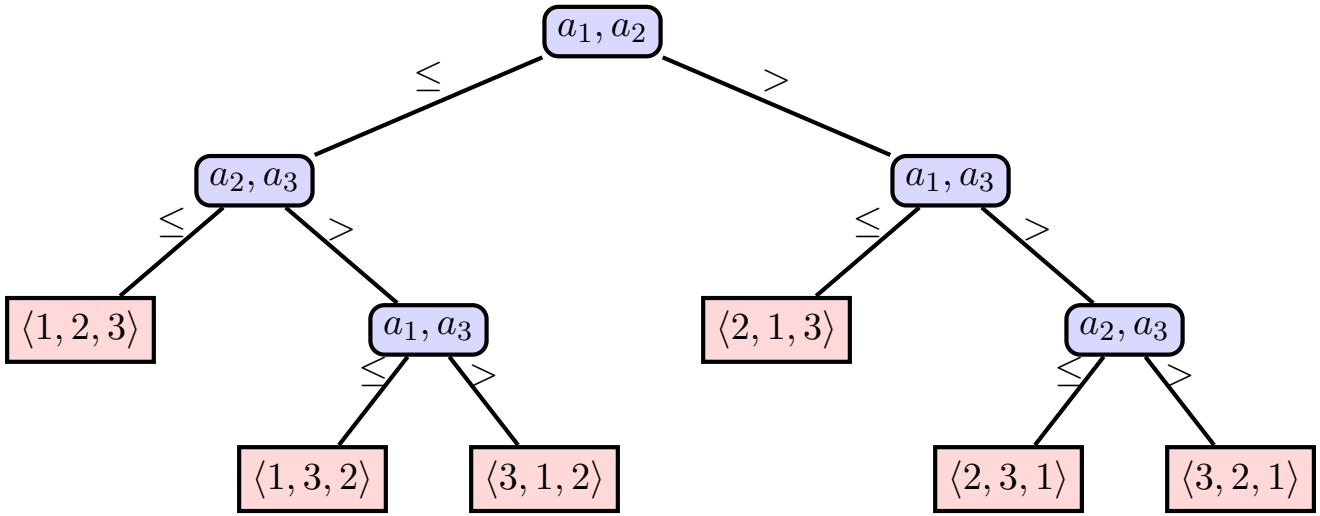


- Cada folha é uma permutação das posições do arranjo;
- Caminho da raiz: comparações realizadas para chegar a uma permutação.

PPgSC

Árvores de decisão

Ordenação por inserção com três elementos



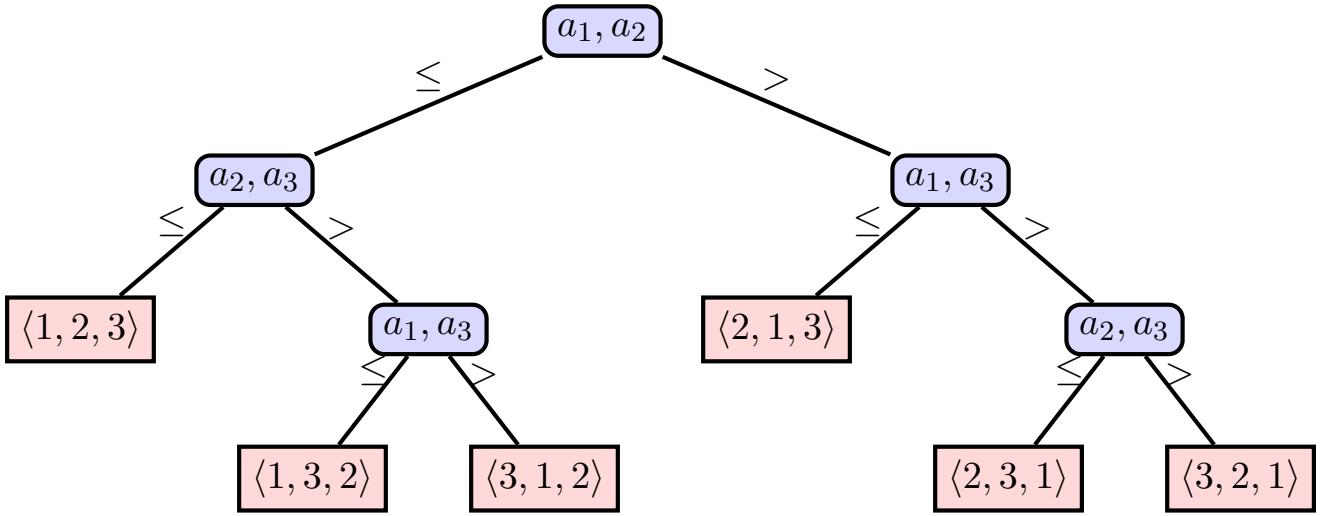
- Cada folha é uma permutação das posições do arranjo;
- Caminho da raiz: comparações realizadas para chegar a uma permutação.

Quantas permutações existem para um arranjo de n posições?

PPgSC

Árvores de decisão

Ordenação por inserção com três elementos



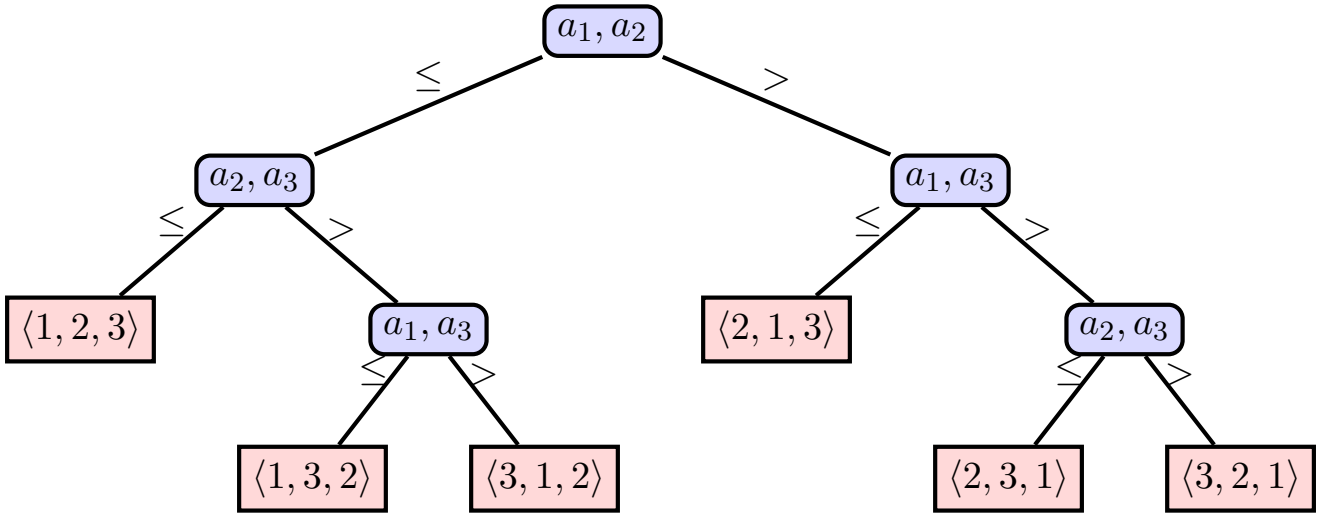
- Cada folha é uma permutação das posições do arranjo;
- Caminho da raiz: comparações realizadas para chegar a uma permutação.

$$n!$$

PPgSC

Árvores de decisão

Ordenação por inserção com três elementos



- Pior caso de um algoritmo de ordenação
  - maior caminho da raiz até uma folha
  - altura da árvore
- O algoritmo com o pior caso de menor custo corresponde à árvore de menor altura.
- Qual a menor altura possível para uma árvore binária de $n!$ folhas?

PPgSC

Teorema

Qualquer árvore de decisão para ordenar n elementos tem altura $\Omega(n \lg n)$.

Teorema

Qualquer árvore de decisão para ordenar n elementos tem altura $\Omega(n \lg n)$.

- Considere uma árvore de decisão para ordenar n elementos.
  - Seja h a altura da árvore;
  - A árvore tem (pelo menos) $n!$ folhas.

Teorema

Qualquer árvore de decisão para ordenar n elementos tem altura $\Omega(n \lg n)$.

- Considere uma árvore de decisão para ordenar n elementos.
  - Seja h a altura da árvore;
  - A árvore tem (pelo menos) $n!$ folhas.
- Uma árvore binária de altura h tem no máximo 2^h folhas.

Teorema

Qualquer árvore de decisão para ordenar n elementos tem altura $\Omega(n \lg n)$.

- Considere uma árvore de decisão para ordenar n elementos.
  - Seja h a altura da árvore;
  - A árvore tem (pelo menos) $n!$ folhas.
- Uma árvore binária de altura h tem no máximo 2^h folhas.
- Logo $n! \leq 2^h$, ou seja $h \geq \lg(n!)$.

Teorema

Qualquer árvore de decisão para ordenar n elementos tem altura $\Omega(n \lg n)$.

- Considere uma árvore de decisão para ordenar n elementos.
  - Seja h a altura da árvore;
  - A árvore tem (pelo menos) $n!$ folhas.
- Uma árvore binária de altura h tem no máximo 2^h folhas.
- Logo $n! \leq 2^h$, ou seja $h \geq \lg(n!)$.
- Fórmula de Stirling $n! > \left(\frac{n}{e}\right)^n$ (aula 04).
- Logo $h \geq \lg((\frac{n}{e})^n) = n \lg n - n \lg e \in \Omega(n \lg n)$.

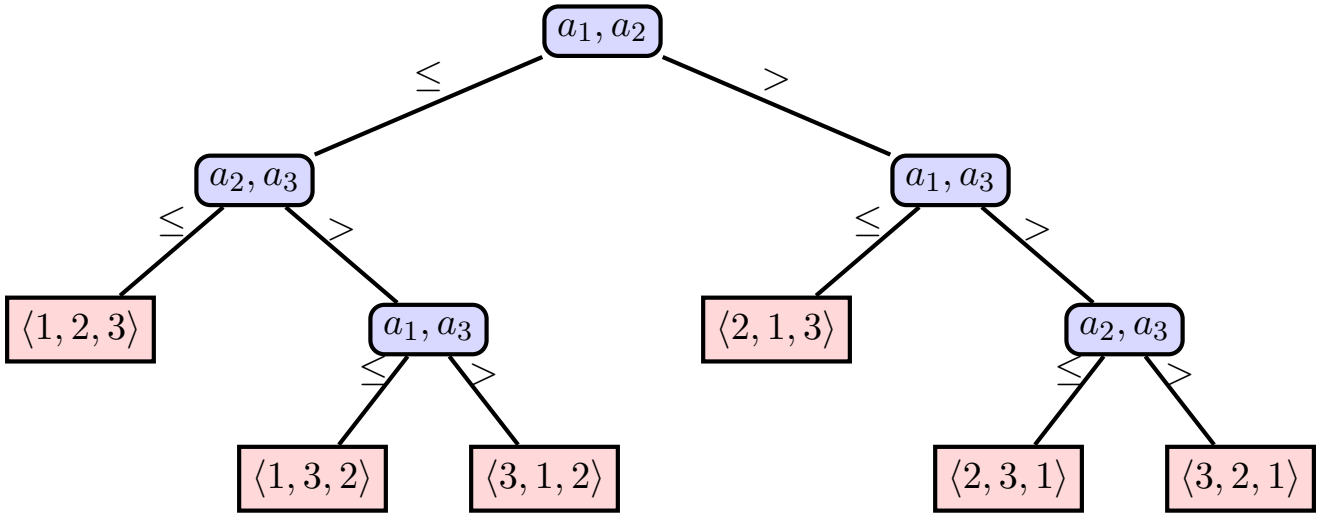
Corolário

Os algoritmos de ordenação por fusão e por heap são asintoticamente ótimos.

- Ambos são $O(n \lg n)$.
- Corresponde com o limite inferior de $\Omega(n \lg n)$ no pior caso enunciado no teorema 1.

Árvores de decisão

Melhor caso

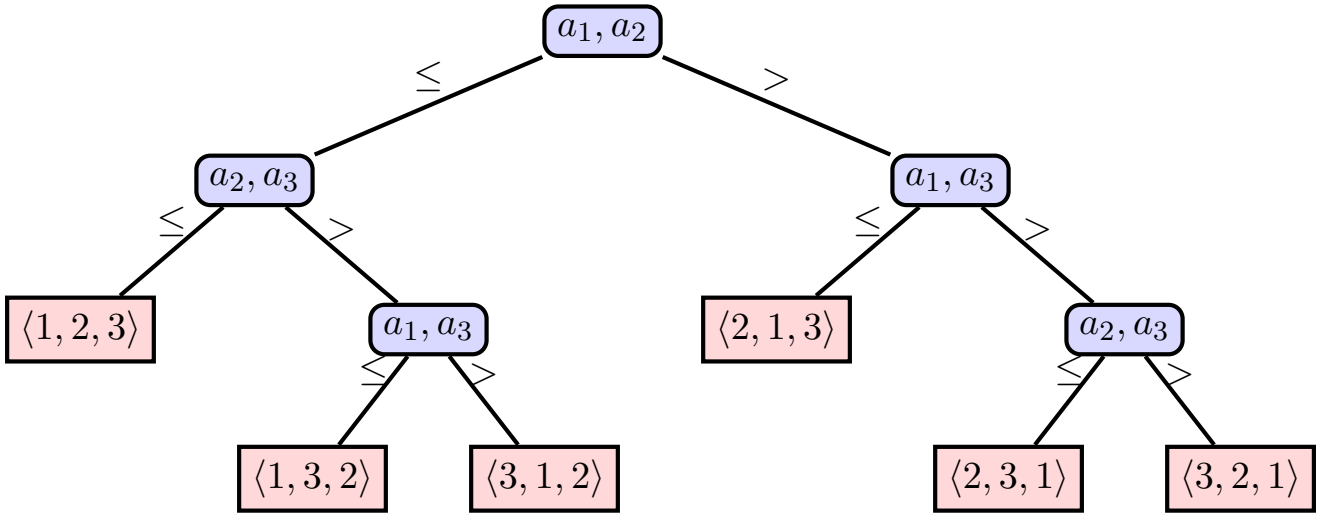


- Melhor caso de um algoritmo de ordenação

PPgSC

Árvores de decisão

Melhor caso

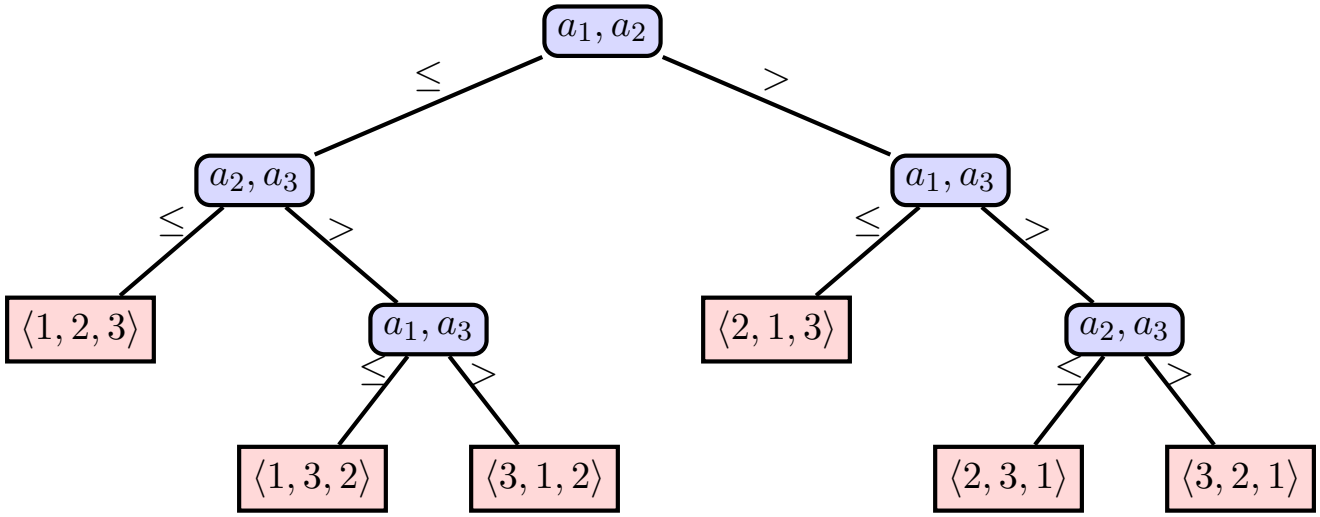


- Melhor caso de um algoritmo de ordenação
  - menor caminho da raiz até uma folha

PPgSC

Árvores de decisão

Melhor caso



- Melhor caso de um algoritmo de ordenação
  - menor caminho da raiz até uma folha
- Qual o menor caminho possível até uma folha em uma árvore de decisão para n valores?

PPgSC

Ordenação por contadores

Introdução

- Ordenação para valores inteiros em uma faixa pré-estabelecida $1 \ldots k$.
  - Os valores comparados na ordenação podem fazer parte de um registro de dados complexo.
  - A ordenação dos registros é realizada então utilizando esses números como chave de comparação.
- Se $k \in O(n)$, então a complexidade é $O(n)$.

Ordenação por contadores

Ideia

- Para cada número a ordenar, contar quantos valores são menores que este número
- Esta quantidade (mais um) é a posição do número na ordem.

Ordenação por contadores

Realização

- Entrada: arranjo $A[1\,..\,n]$
- Saída: arranjo $B[1\,..\,n]$
- Memória auxiliar: arranjo $C[1..k]$ (arranjo de contadores)

Ordenação por contadores

Algoritmo

COUNTING-SORT(A, B, k)

```
    // zera os contadores
1   for i = 1 to k
2       C[i] = 0
    // conta em o número de ocorrências de cada valor
3   for i = 1 to length(A)
4       C[A[i]] = C[A[i]] + 1
    // conta em C[i] o número de valores menores ou iguais a A[i]
5   for i = 2 to k
6       C[i] = C[i] + C[i − 1]
    // copia cada element de A na sua posição final em B
7   for i = length(A) downto 1
8       B[C[A[i]]] = A[i]
9       C[A[i]] = C[A[i]] − 1
```

Ordenação por contadores

Simulação

COUNTING-SORT(A, B, k)

```
1  for i = 1 to k
2      C[i] = 0
3  for i = 1 to length(A)
4      C[A[i]] = C[A[i]] + 1
5  for i = 2 to k
6      C[i] = C[i] + C[i − 1]
7  for i = length(A) downto 1
8      B[C[A[i]]] = A[i]
9      C[A[i]] = C[A[i]] − 1
```

$A = \langle 3, 6, 4, 1, 3, 4, 1 \rangle, k = 8, B = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

PPgSC

Ordenação por contadores

Simulação

COUNTING-SORT(A, B, k)

```
1   for i = 1 to k
2       C[i] = 0
    // C = ⟨0, 0, 0, 0, 0, 0, 0⟩
3   for i = 1 to length(A)
4       C[A[i]] = C[A[i]] + 1
5   for i = 2 to k
6       C[i] = C[i] + C[i − 1]
7   for i = length(A) downto 1
8       B[C[A[i]]] = A[i]
9       C[A[i]] = C[A[i]] − 1
```

$A = \langle 3, 6, 4, 1, 3, 4, 1 \rangle, k = 8, B = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

PPgSC

Ordenação por contadores

Simulação

COUNTING-SORT(A, B, k)

```
1   for i = 1 to k
2       C[i] = 0
    // C = ⟨0, 0, 0, 0, 0, 0, 0⟩
3   for i = 1 to length(A)
4       C[A[i]] = C[A[i]] + 1
    // C = ⟨2, 0, 2, 2, 0, 1, 0⟩
5   for i = 2 to k
6       C[i] = C[i] + C[i − 1]
7   for i = length(A) downto 1
8       B[C[A[i]]] = A[i]
9       C[A[i]] = C[A[i]] − 1
```

$A = \langle 3, 6, 4, 1, 3, 4, 1 \rangle, k = 8, B = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

PPgSC

Ordenação por contadores

Simulação

COUNTING-SORT(A, B, k)

```
1   for i = 1 to k
2       C[i] = 0
3   for i = 1 to length(A)
4       C[A[i]] = C[A[i]] + 1
    // C = ⟨2, 0, 2, 2, 0, 1, 0⟩
5   for i = 2 to k
6       C[i] = C[i] + C[i − 1]
    // C = ⟨2, 2, 4, 6, 6, 7, 7⟩
7   for i = length(A) downto 1
8       B[C[A[i]]] = A[i]
9       C[A[i]] = C[A[i]] − 1
```

$A = \langle 3, 6, 4, 1, 3, 4, 1 \rangle, k = 8, B = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

PPgSC

Ordenação por contadores

Simulação

COUNTING-SORT(A, B, k)

```
1  for i = 1 to k
2      C[i] = 0
3  for i = 1 to length(A)
4      C[A[i]] = C[A[i]] + 1
5  for i = 2 to k
6      C[i] = C[i] + C[i − 1]
   // C = ⟨2, 2, 4, 6, 6, 7, 7⟩
7  for i = length(A) downto 1
8      B[C[A[i]]] = A[i]
9      C[A[i]] = C[A[i]] − 1
   // B = ⟨1, 1, 3, 3, 4, 4, 6⟩
```

$A = \langle 3, 6, 4, 1, 3, 4, 1 \rangle, k = 8, B = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

PPgSC

Ordenação por contadores

Complexidade

- Número de operações: $\Theta(n + k)$
- Espaço: $\Theta(n + k)$

Ordenação por contadores

Complexidade

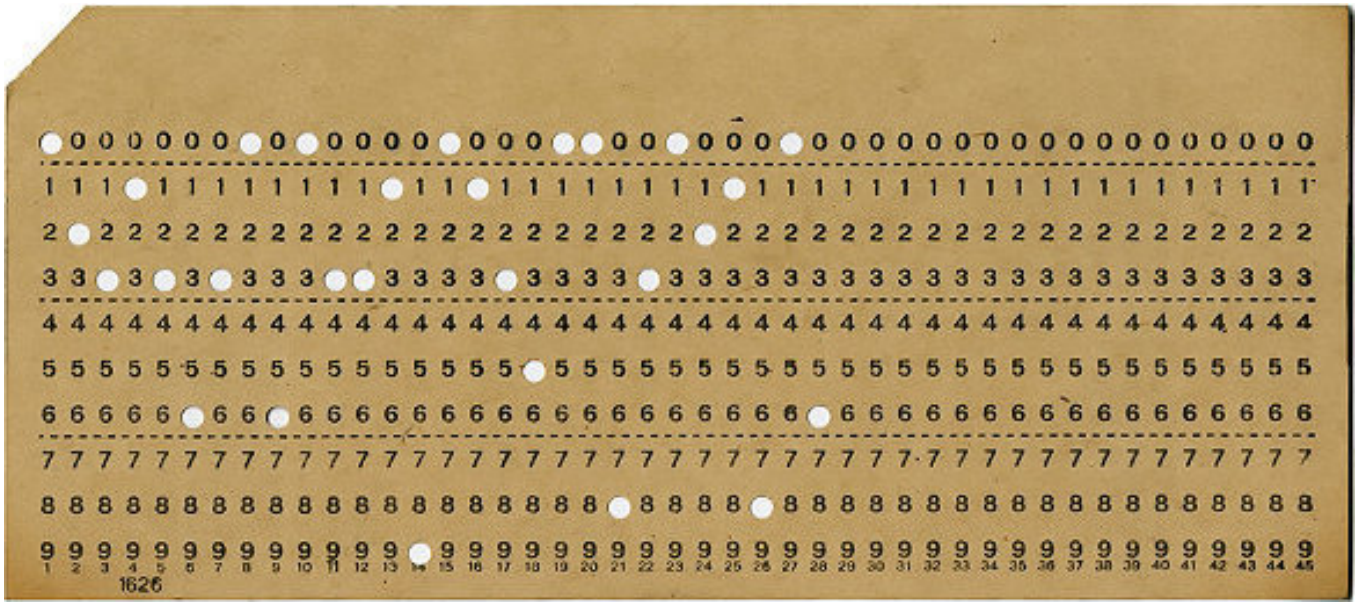
- Número de operações: $\Theta(n + k)$
- Espaço: $\Theta(n + k)$

- Um algoritmo de ordenação é estável quando preserva a ordem inicial entre elementos iguais.
  - O algoritmo de ordenação por contadores é estável?
  - Justifique.
- Se não houver repetições no arranjo inicial: o que pode-se fazer para ter um algoritmo mais eficiente?

PPgSC

Ordenação por algarismos (*radix sort*)

Introdução



(Computer History Museum)

- 80 colunas $\times$ 12 linhas
- máquina de ordenar cartões:
  - configurada com um número de coluna c
  - separa os cartões em 12 caixas em função de qual linha foi perfurada naquela coluna.
  - o operador junta as pilhas de cartões: furo na 1a posição primeiro, etc.

PPgSC

Ordenação *radix sort*

Ideia

- Para números decimais, só 10 linhas são usadas.
- Como configurar a máquina para ordenar cartões?

Ordenação *radix sort*

Ideia

- Para números decimais, só 10 linhas são usadas.
- Como configurar a máquina para ordenar cartões?
- Solução
  1. Separar por dígito menos significativo (unidades)
  2. Juntar
  3. Separar por segundo dígito menos significativo (dezenas)
  4. Juntar
  5. etc. até processar todos os dígitos.

Ordenação *radix sort*

Simulação

Inicial	1o dígito	2o dígito	3o dígito	4o dígito
8091				
0893				
8635				
8805				
8856				
8164				
3868				
9038				
9224				

Ordenação *radix sort*

Simulação

Inicial	1o dígito	2o dígito	3o dígito	4o dígito
8091	8091			
0893	0893			
8635	8164			
8805	9224			
8856	8635			
8164	8805			
3868	8856			
9038	3868			
9224	9038			

Ordenação *radix sort*

Simulação

Inicial	1o dígito	2o dígito	3o dígito	4o dígito
8091	8091	8805		
0893	0893	9224		
8635	8164	8635		
8805	9224	9038		
8856	8635	8856		
8164	8805	8164		
3868	8856	3868		
9038	3868	8091		
9224	9038	0893		

Ordenação *radix sort*

Simulação

Inicial	1o dígito	2o dígito	3o dígito	4o dígito
8091	8091	8805	9038	
0893	0893	9224	8091	
8635	8164	8635	8164	
8805	9224	9038	9224	
8856	8635	8856	8635	
8164	8805	8164	8805	
3868	8856	3868	8856	
9038	3868	8091	3868	
9224	9038	0893	0893	

Ordenação *radix sort*

Simulação

Inicial	1o dígito	2o dígito	3o dígito	4o dígito
8091	8091	8805	9038	0893
0893	0893	9224	8091	3868
8635	8164	8635	8164	8091
8805	9224	9038	9224	8164
8856	8635	8856	8635	8635
8164	8805	8164	8805	8805
3868	8856	3868	8856	8856
9038	3868	8091	3868	9038
9224	9038	0893	0893	9224

Ordenação *radix sort*

Algoritmo

RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d

2 Aplique uma ordenação estável utilizando o dígito i

Ordenação *radix sort*

Algoritmo

RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d

2 Aplique uma ordenação estável utilizando o dígito i

Observações

- Algoritmo candidato: ordenação por contador
- Este algoritmo funciona para qualquer tipo de dados que pode ser visto como uma sequência de tamanho fixo (d), ordenável lexicograficamente.

Ordenação *radix sort*

Análise

Correção Verificado por indução utilizando a coluna sendo ordenada, e o fato da ordenação sobre cada coluna ser estável.

Complexidade Utilizando a ordenação por contadores, assumindo k dígitos diferentes,

- o custo de cada iteração é $\Theta(n + k)$;
- se há d colunas, o custo é $\Theta(d(n + k))$;
- quando d é constante, e $k \in O(n)$, então o algoritmo é $\Theta(n)$.

Ordenação por lotes (*bucket sort*)

Introdução

- Este algoritmo é aplicável quando os dados a serem ordenados estão distribuídos uniformemente sobre um intervalo.
- Intervalo: $\{x \mid 0 \leq x < 1\}$

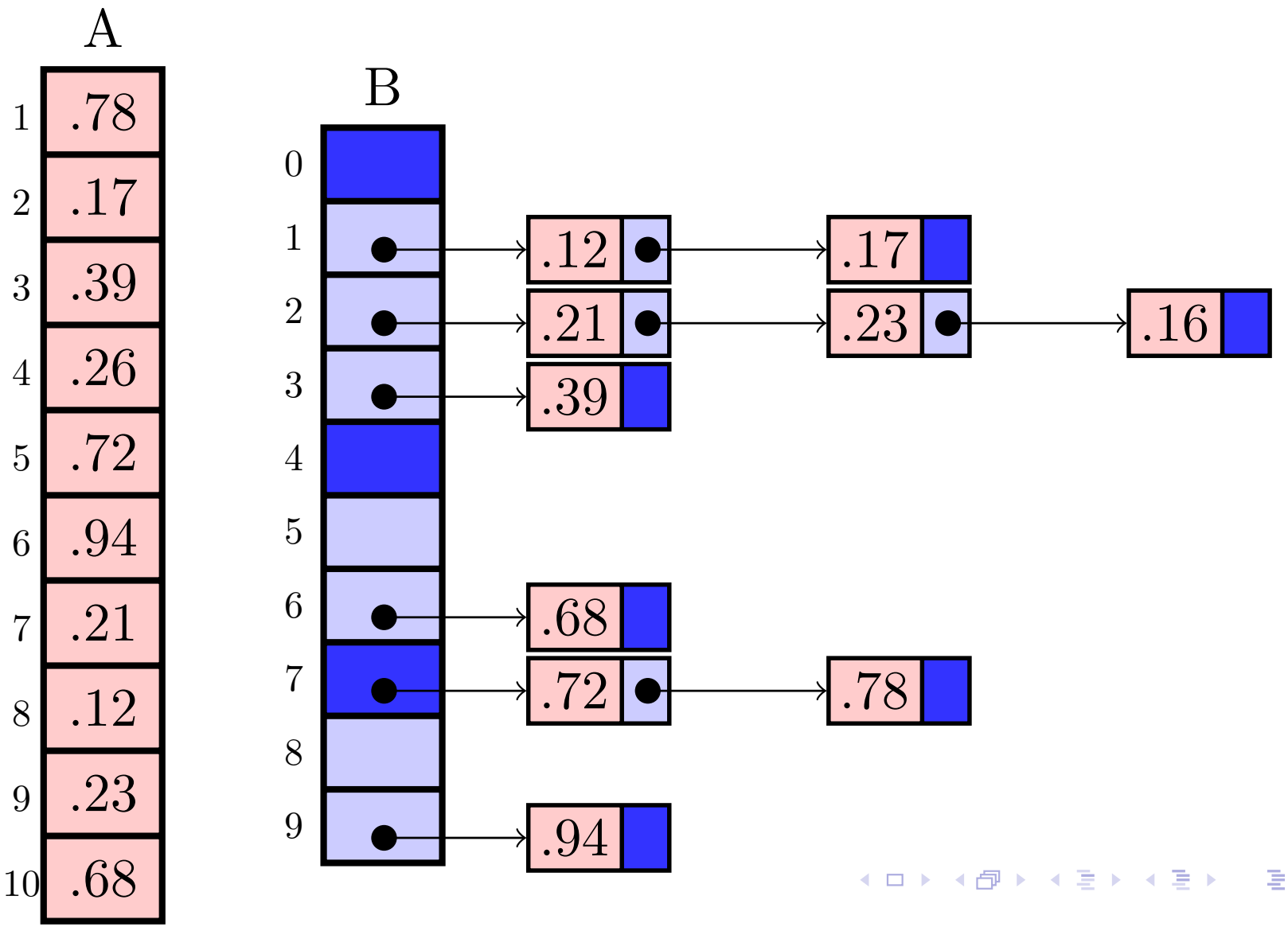
Ordenação por lotes (*bucket sort*)

Ideia

- Assumindo que há n valores a ordenar;
- Criar um arranjo de n listas;
- Cada lista armazena os valores em uma parte do intervalo: $0 \leq x < 1/n$, $1/n \leq x < 2/n$, $\ldots n - 1/n \leq x < 1$.
- Os valores a ordenar são inseridos na lista correspondente.
- Com alta probabilidade, cada lista é pequena e pode ser processada eficientemente pelo algoritmo de ordenação por inserção
- As listas são copiadas para o arranjo final em ordem.

Ordenação por lotes (*bucket sort*)

Ilustração



PPgSC

Ordenação por lotes (*bucket sort*)

Algoritmo

BUCKET-SORT(A)

1 $n = length(A)$
2 **for** $i = 1$ **to** n
3 Inserir $A[i]$ na lista $B[\lfloor n \times A[i] \rfloor]$
4 **for** $i = 0$ **to** $n - 1$
5 Ordenar $B[i]$ por inserção
6 Concatenar $B[0]$, $B[1]$, ... $B[n-1]$, nesta ordem

Ordenação por lotes (*bucket sort*)

Análise

Correção

Complexidade

Ordenação por lotes (*bucket sort*)

Análise

Correção Considere dois valores $A[i]$ e $A[j]$, tais que $A[i] < A[j]$

- os valores são copiados para os lotes $B[i']$ e $B[j']$, onde:
  - $i' = \lfloor nA[i] \rfloor$
  - $j' = \lfloor nA[j] \rfloor$
  - Logo $i' \leq j'$
- se $i' = j'$, então $A[i]$ e $A[j]$ serão ordenados (pois cada lote é ordenado)
- se $i' < j'$, então $A[i]$ e $A[j]$ serão ordenados, pois cada valor de i' vem antes de cada valor de j'

Complexidade

PPgSC

Ordenação por lotes (*bucket sort*)

Análise

Correção

Complexidade

- A única parte do algoritmo que tem complexidade linear é a ordenação dos lotes (linha 5).
- Utilizando resultados da teoria da probabilidade:
  - havendo n lotes e n valores,
  - assumindo uma distribuição uniforme dos valores no intervalo $[0, 1[$
  - o número esperado de valor por lote é $\Theta(1)$.
- O custo esperado de ordenar cada lote é $\Theta(1)$
- O custo do algoritmo é linear.

Síntese

Algoritmos de ordenação

- Complexidade teórica: $O(n^2)$ ou $O(n \lg n)$.
- Importância da randomização para algoritmos "quase sempre ótimos".
- Memória auxiliar: constante ou não?
- Estabilidade da ordenação.
- Características dos dados a serem ordenados
  - Completamente aleatórios ou "quase-ordenados"?
  - Passíveis de serem tratados por um algoritmo de complexidade linear?

PPgSC