Governo Federal

Ministério da Educação

deINF departamentodeInformática

Universidade Federal do Maranhão

A Universidade que Cresce com Inovação e Inclusão Social

Algoritmos de Ordenação
Ordenação Linear

Estrutura de Dados II

Prof. João Dallyson

Email: Joao.dallyson@ufma.br

Introdução

- **Ordenação por comparação:**
  - InsertSort
  - QuickSort
  - MergeSort
  - HeapSort
- **Limite assintótico inferior Ω ($nlog_2n$)**
  - MergeSort e HeapSort alcançam esse limite no pior caso
  - O QuickSort alcança na média

Introdução

- **Podem existir algoritmos melhores?**
  - Não, se o algoritmo for baseado em comparações
  - Todo algoritmo de ordenação baseado em comparações faz $\Omega(n\log_2 n)$ no pior caso
- **Em alguns casos especiais é possível superar esse limite e realizar a ordenação em $\Theta(n)$**
  - A entrada possua características especiais
  - Algumas restrições sejam respeitadas
  - O algoritmo não seja puramente baseado em comparações
  - A implementação seja feita de maneira adequada

Algoritmos de tempo linear

- **Ordenação por contagem (Counting Sort)**
- **Radix Sort**
- **Bucket Sort**

Counting Sort - Funcionamento

- **Pressupõe que cada elemento da entrada é um inteiro na faixa de 0 a k, para algum inteiro k**
- **Ideia básica:**
  – Determinar para cada elemento da entrada x o número de elementos maiores que x
  – Com esta informação, determinar a posição de cada elemento
  – Ex.: Se 17 elementos forem menores que x então x ocupa a posição de saída 18
- **http://visualgo.net/sorting.html**

Counting Sort

- **Exemplo:**

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

B =						7			
	1	2	3	4	5	6	7	8	9

- **Na lista A acima o elemento circulado 7 possui 5 elementos menores que ele. Dessa forma o elemento 7 deverá ser inserido no índice 6 (5 + 1) do vetor de saída B.**

Counting Sort

O algoritmo recebe um vetor desordenado como entrada:

	1	2	3	4	5	6	7	8
A =	2	5	3	0	2	3	0	3

Em seguida, gera os vetores adicionais ***B*** e ***C***:

	1	2	3	4	5	6	7	8
B =								

→ O vetor ***B*** é do mesmo tamanho do vetor ***A*** (8 elementos).

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

→ O vetor ***C*** é do tamanho do maior elemento de ***A*** + 1 (5 + 1 = *6*).

Counting Sort

Se o valor de um elemento de entrada é ***i***, incrementamos ***C[i]:***

	0	1	2	3	4	5
C =	**2**	**0**	**2**	**3**	**0**	**1**

→***C[i]*** contém um número de elementos de entrada igual a ***i*** para cada ***i = 0,1,2,…,k.***

Agora fazemos ***C[i] = C[i] + C[i-1]*** para determinarmos quantos elementos de entrada são menores que ou iguais a ***i:***

	0	1	2	3	4	5
C =	**2**	**2**	**4**	**7**	**7**	**8**

Counting Sort

Agora, partindo do maior para o menor índice, fazemos ***B[C[A[j]]] = A[j]***. Assim, colocamos cada elemento ***A[j]*** em sua Posição ordenada no vetor ***B:***

Exemplo:

→ Para j = 8
B[C[A[8]]] → B[C[3]] → B[7]
B[7] = A[8] → B[7] = 3

Counting Sort

Em seguida decrementamos o valor de C[A[j]] toda vez que Inserimos um valor no vetor B. isso faz com que o próximo elemento de entrada com valor igual a A[j], se existir, vá para a posição imediatamente anterior a A[j] no vetor B.

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

C[3] = C[3] – 1:

	0	1	2	3	4	5
C =	2	2	4	6	7	8

Counting Sort - Algoritmo

```
for i ← 0 to k
   do C[i] ← 0
for j ← 1 to length[A]
   do C[A[j]] ← C[A[j]] + 1
// C[i] contém o número de elementos iguais a i
for i ← 2 to k
   do C[i] ← C[i] + C[i − 1]
// C[i] contém o número de elementos menores ou iguais a
  i
for j ← length[A] downto 1
   do B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]] − 1
```

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

	0	1	2	3	4	5
C						

k=5

	1	2	3	4	5	6	7	8
B								

Counting Sort - Algoritmo

```
for i ← 0 to k
   do C[i] ← 0
for j ← 1 to length[A]
   do C[A[j]] ← C[A[j]] + 1
// C[i] contém o número de elementos iguais a i
for i ← 2 to k
   do C[i] ← C[i] + C[i − 1]
// C[i] contém o número de elementos menores ou iguais a i
for j ← length[A] downto 1
   do B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]] − 1
```

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

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

k=5

	1	2	3	4	5	6	7	8
B								

Counting Sort

```
for i ← 0 to k
   do C[i] ← 0
for j ← 1 to length[A]
   do C[A[j]] ← C[A[j]] + 1
// C[i] contém o número de elementos iguais a i
for i ← 2 to k
   do C[i] ← C[i] + C[i − 1]
// C[i] contém o número de elementos menores ou iguais a
   i
for j ← length[A] downto 1
   do B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]] − 1
```

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

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

k=5

	1	2	3	4	5	6	7	8
B								

Counting Sort - Algoritmo

```
for i ← 0 to k
   do C[i] ← 0
for j ← 1 to length[A]
   do C[A[j]] ← C[A[j]] + 1
// C[i] contém o número de elementos iguais a i
for i ← 2 to k
   do C[i] ← C[i] + C[i − 1]
// C[i] contém o número de elementos menores ou iguais a i
for j ← length[A] downto 1
   do B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]] − 1
```

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

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

k=5

	1	2	3	4	5	6	7	8
B								

Counting Sort - Algoritmo

```
for i ← 0 to k
   do C[i] ← 0
for j ← 1 to length[A]
   do C[A[j]] ← C[A[j]] + 1
// C[i] contém o número de elementos iguais a i
for i ← 2 to k
   do C[i] ← C[i] + C[i – 1]
// C[i] contém o número de elementos menores ou iguais a
  i
for j ← length[A] downto 1
   do B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]] – 1
```

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

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

k=5

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

Count Sort - Complexidade

- **O primeiro for tem O(k)**
- **O Segundo, O(n)**
- **O Terceiro, O(k)**
- **O Quarto, O(n)**
- **Assim, o tempo é O(n+k)**
- **Na prática o Counting Sort deve ser usado quando k=O(n) o que leva a ter um custo O(n)**

Counting Sort

- **Vantagens**
  - Ordena vetores em tempo linear para o tamanho do vetor inicial;
  - Não realiza comparações;
  - É um algoritmo de ordenação estável;

- **Desvantagens**
  - Necessita de dois vetores adicionais para sua execução, utilizando, assim mais espaço na memória

RadixSort

- **Radix Sort é o algoritmo utilizado pelas máquinas de ordenação de cartões**
- **Pressupõe que as chaves de entrada possuem limite no valor e no tamanho (quantidade de dígitos)**
- **Ordena em função dos dígitos (um de cada vez)**
  - ordena a partir do dígito menos significativo
- **É essencial utilizar um segundo algoritmo estável para realizar a ordenação de cada dígito**
- **http://visualgo.net/sorting.html**

RadixSort

- **Algoritmo**
  – Ordena-se coluna por coluna de dígitos dos números do digito menos significativo para o mais significativo (fazendo as devidas trocas caso existam)

RadixSort

3 2 9	7 2 0	7 2 0	3 2 9
4 5 7	3 5 5	3 2 9	3 5 5
6 5 7	4 3 6	4 3 6	4 3 6
8 3 9	4 5 7	8 3 9	4 5 7
4 3 6	6 5 7	3 5 5	6 5 7
7 2 0	3 2 9	4 5 7	7 2 0
3 5 5	8 3 9	6 5 7	8 3 9

RadixSort

- **Utiliza um outro método de ordenação (estável) para ordenar as chaves em relação a cada dígito**

```
1 for i ← 1 to d
2 do utilize um algoritmo estável para ordenar o array A
     pelo i-ésimo dígito
```

em que **d** é o número de dígitos e **A** o vetor de entrada

RadixSort – Análise Complexidade

- **A corretude de radix sort segue por indução na coluna sendo ordenada**
  - A análise depende do algoritmo de ordenação estável utilizado no passo intermediário
  - Quando cada dígito varia entre 1 e **k**, e **k** não é muito grande, counting sort é a escolha óbvia
  - Cada passagem sobre **n** números de **d** dígitos leva tempo **(n + k**)
  - Há **d** passos, portanto o tempo total de radix sort é **(dn + dk)**
  - Quando d é uma constante e k = O(n), radix sort roda em tempo O(n)

Bucket Sort

- **Assume que a entrada consiste em elementos distribuídos de forma uniforme sobre o intervalo [0,1)**
- **A idéia do *Bucket Sort é dividir o intervalo [0,1) em n subintervalos de mesmo tamanho (baldes), e então distribuir os n números nos baldes***

- **Uma vez que as entradas são uniformemente distribuídas não se espera que muitos números caiam em cada *balde***

Bucket Sort

- **Para produzir a saída ordenada, basta inserir os números em cada balde, e depois examinar os baldes em ordem, listando seus elementos**

- **A função para determinação do índice do balde correto é**

$$\lfloor n \times A[i] \rfloor$$

Bucket Sort - Algoritmo

- **O código assume que:**
  - A entrada é um vetor A[1...n]
  - Utiliza um vetor auxiliar B[0 . . . n – 1] de listas ligadas e assume que existe um mecanismo para tratamento das listas

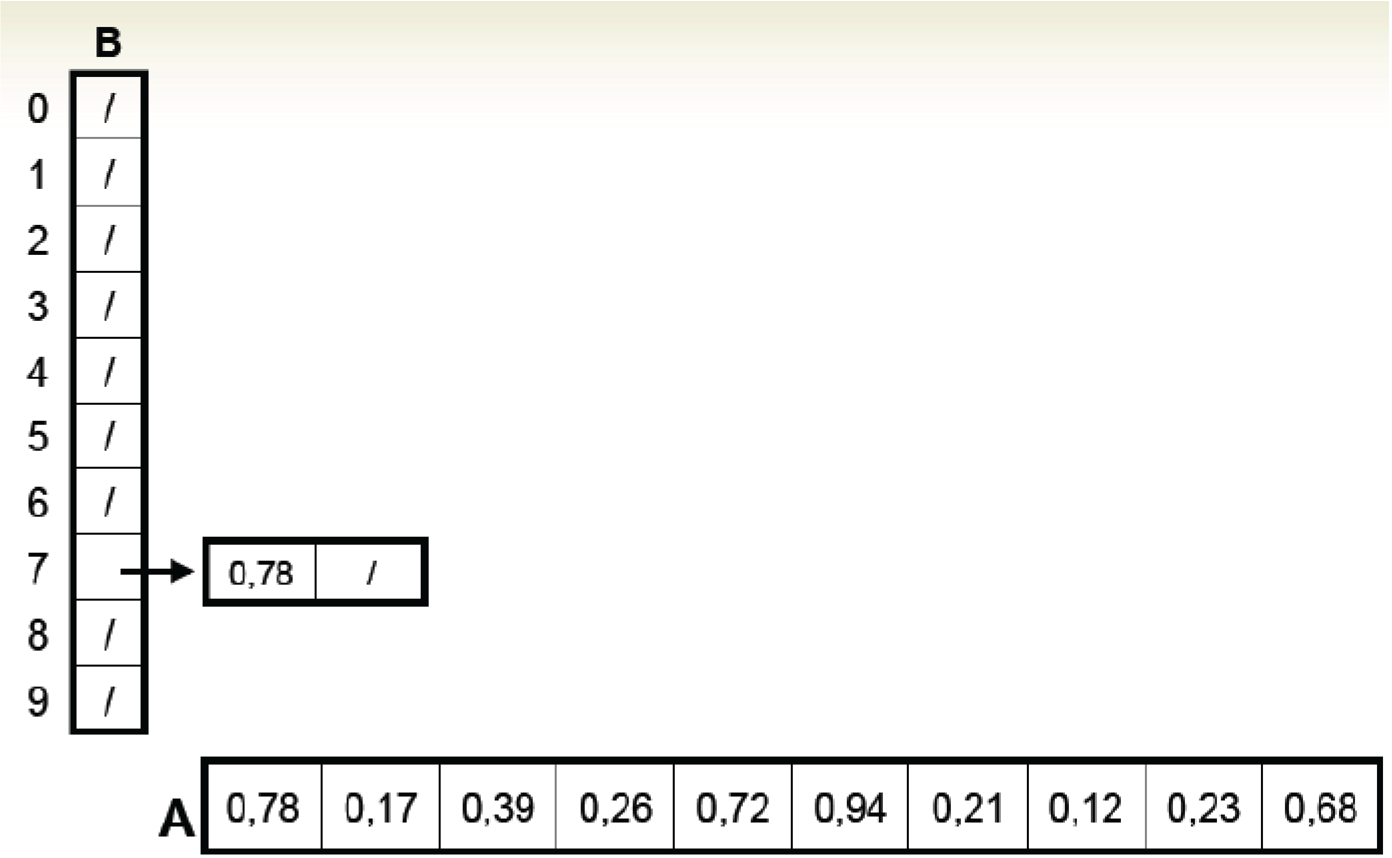
BUCKET-SORT(A)
1 $n \leftarrow$ *comprimento*[A]
2 **for** $i \leftarrow 1$ **to** n
3   **do** inserir $A[i]$ na lista $B[\lfloor nA[i] \rfloor]$
4 **for** $i \leftarrow 0$ **to** $n - 1$
5   **do** ordenar lista $B[i]$ com ordenação por inserção
6 concatenar as listas $B[0], B[1], \ldots, B[n-1]$ juntas em ordem

Bucket Sort - Exemplo

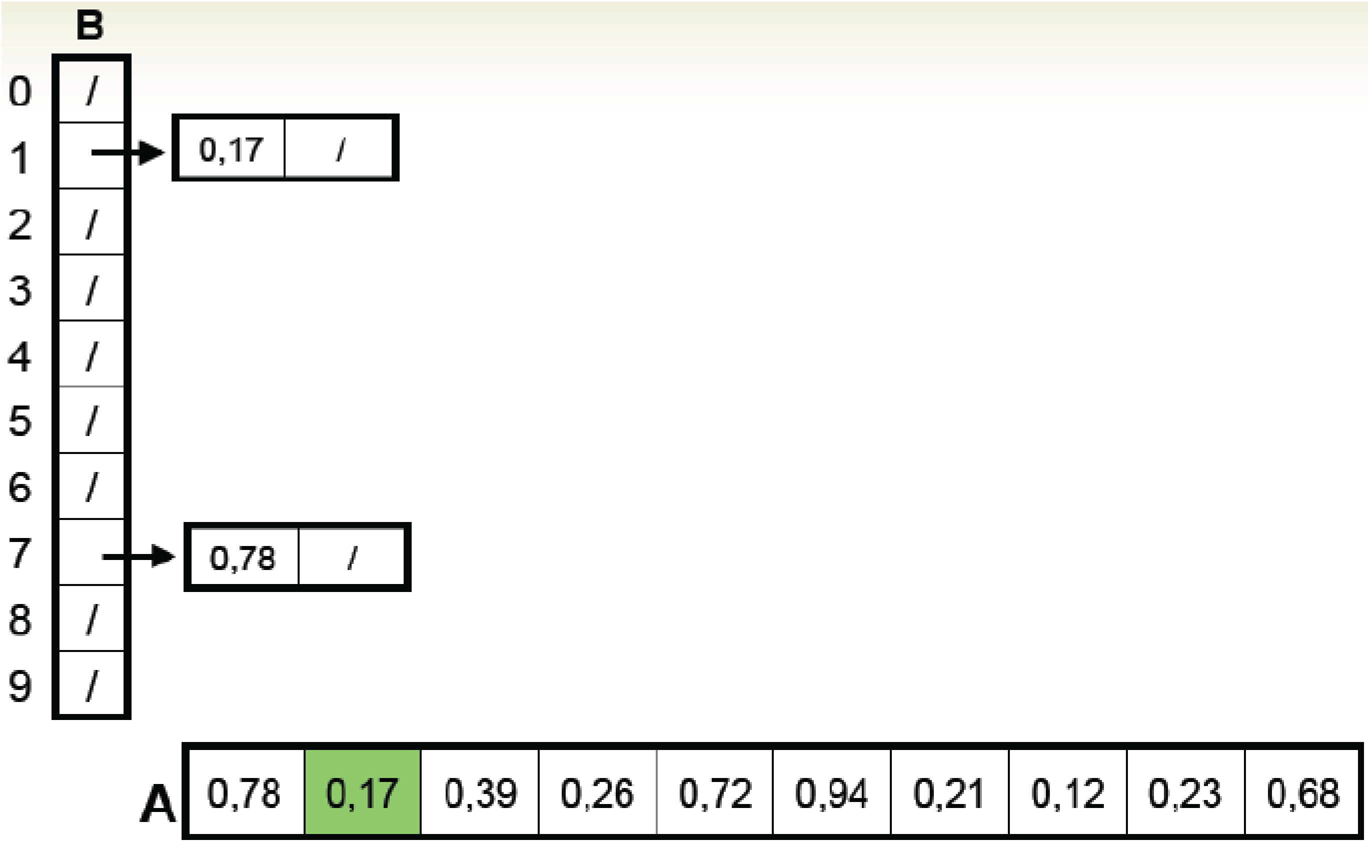
	B
0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/
9	/

A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68

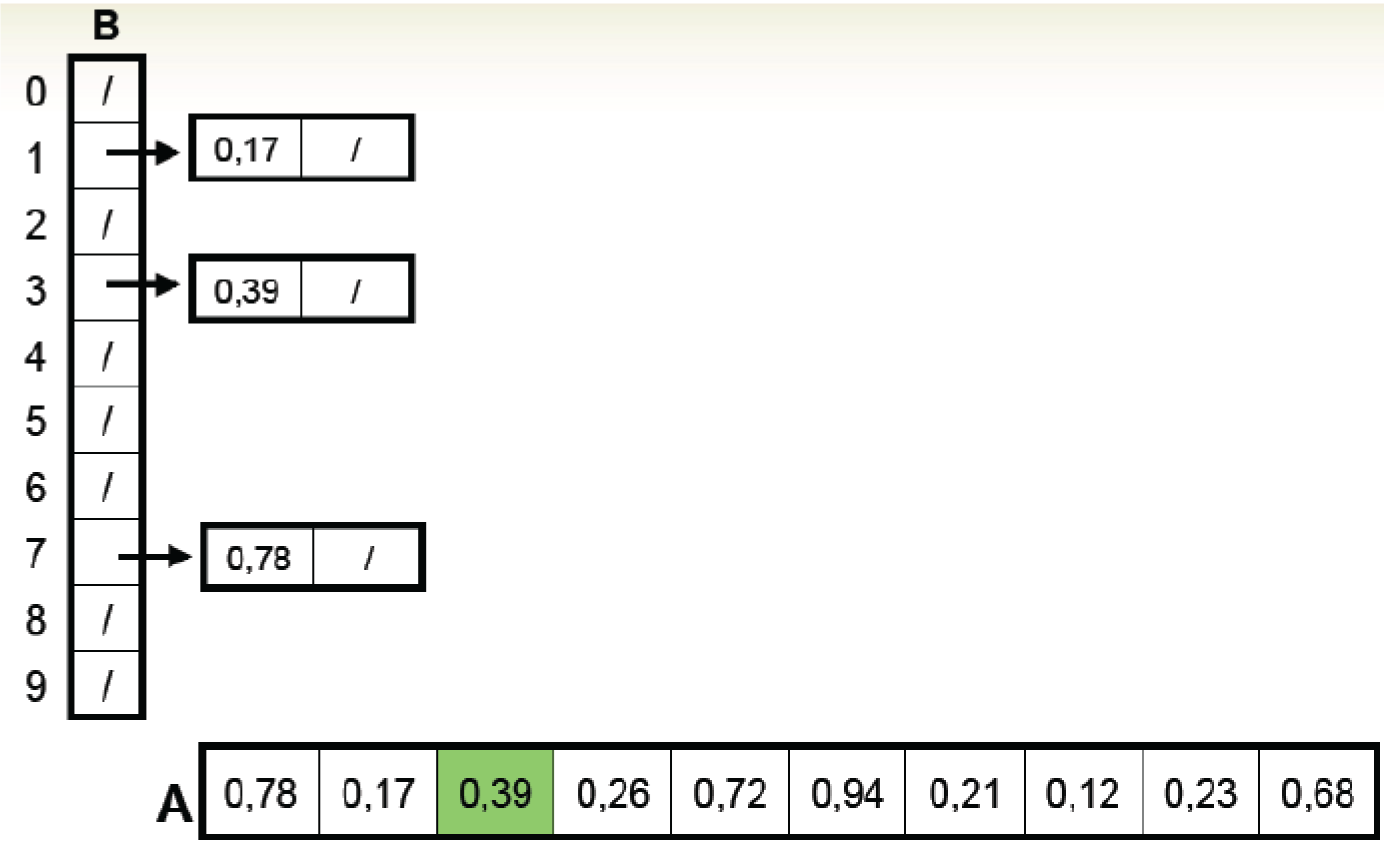
Bucket Sort - Exemplo



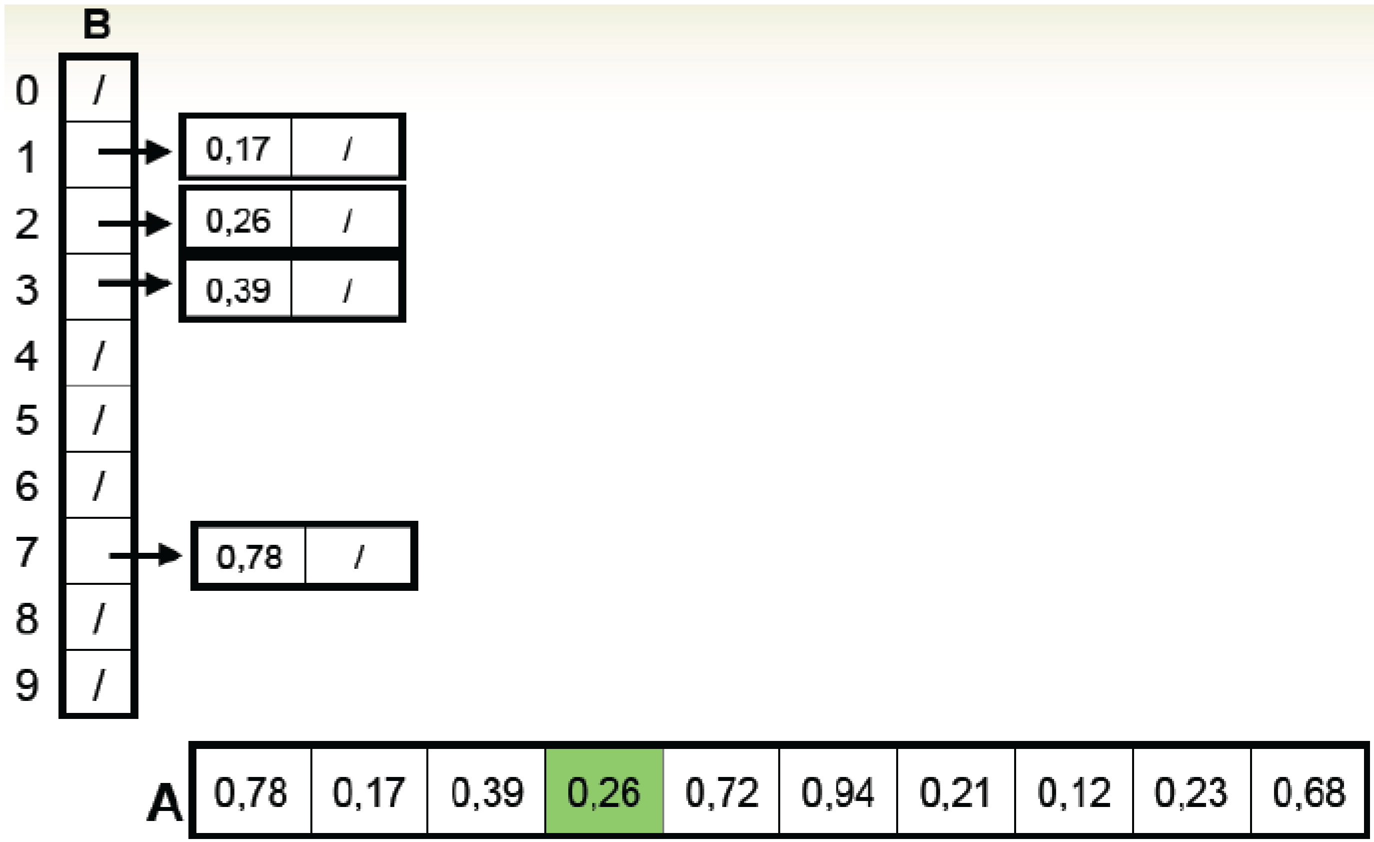
Bucket Sort - Exemplo



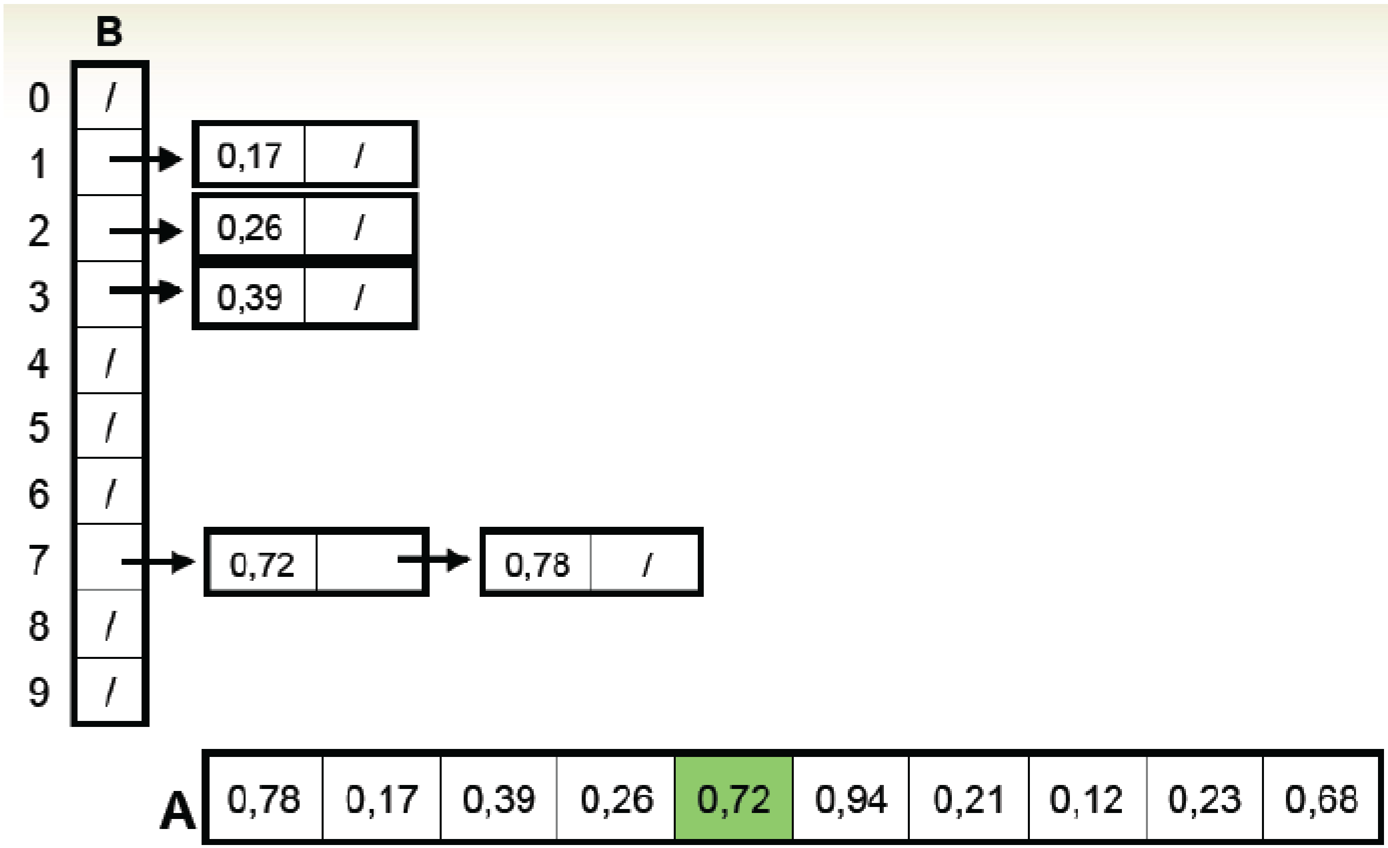
Bucket Sort - Exemplo



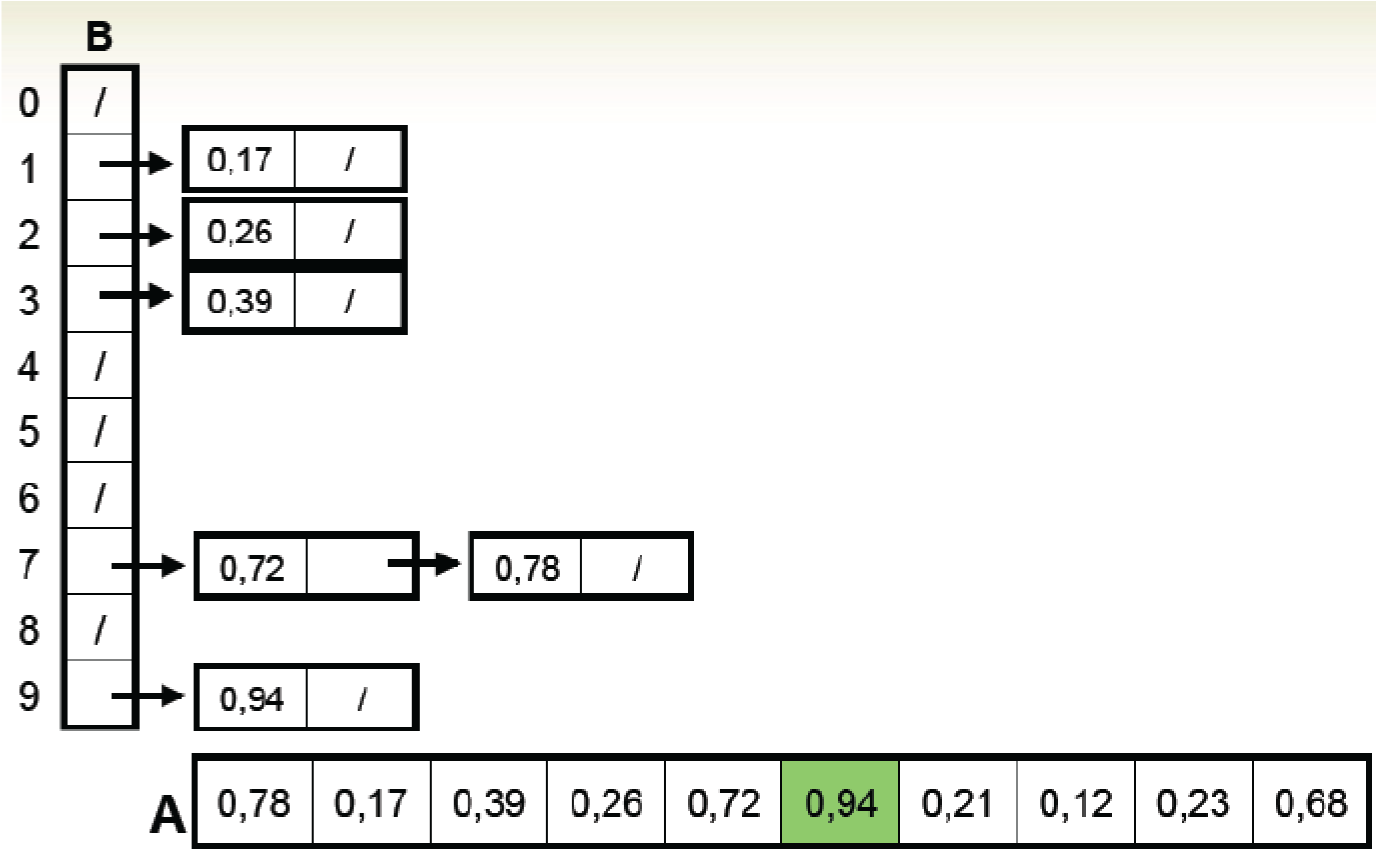
Bucket Sort - Exemplo



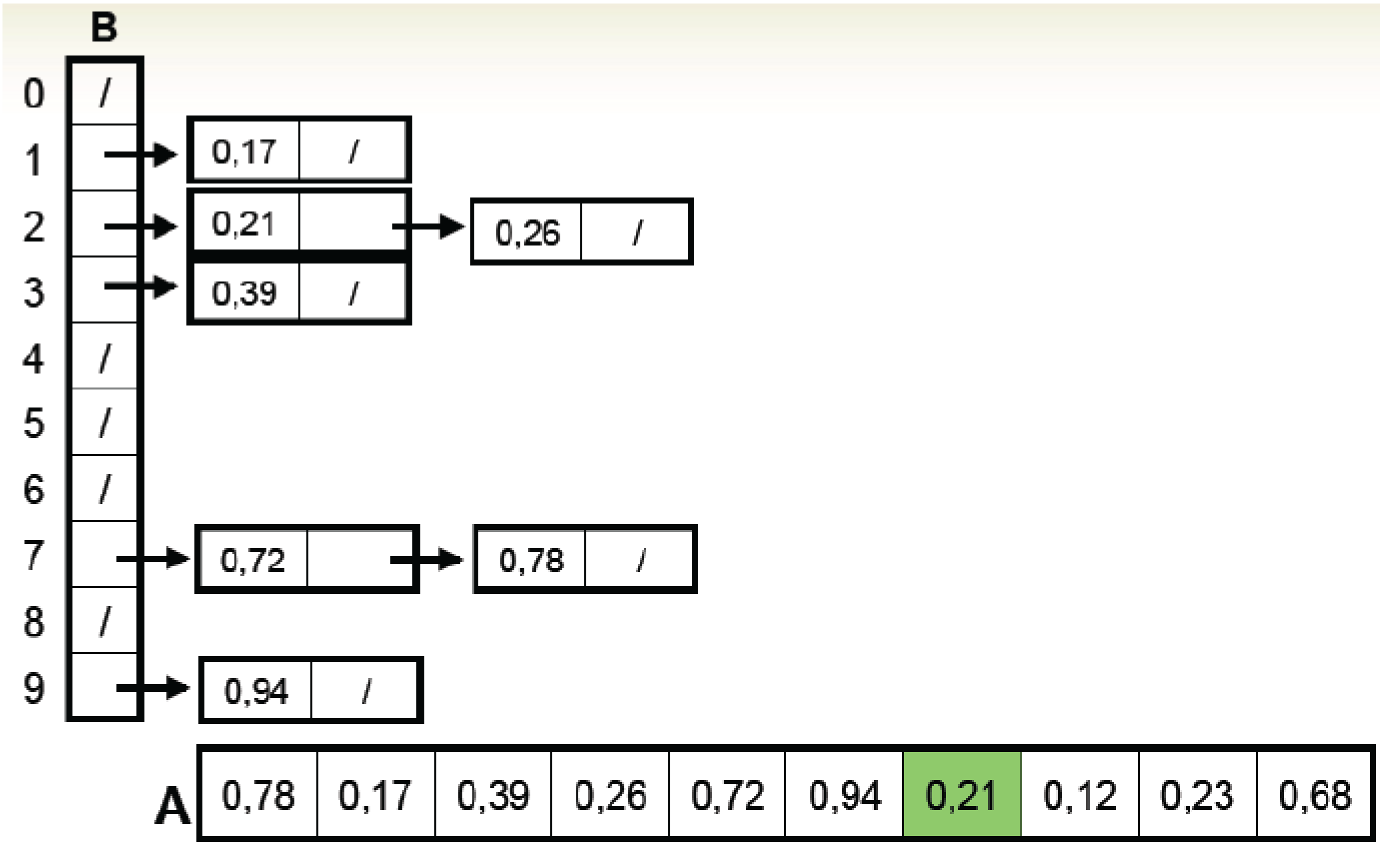
Bucket Sort - Exemplo



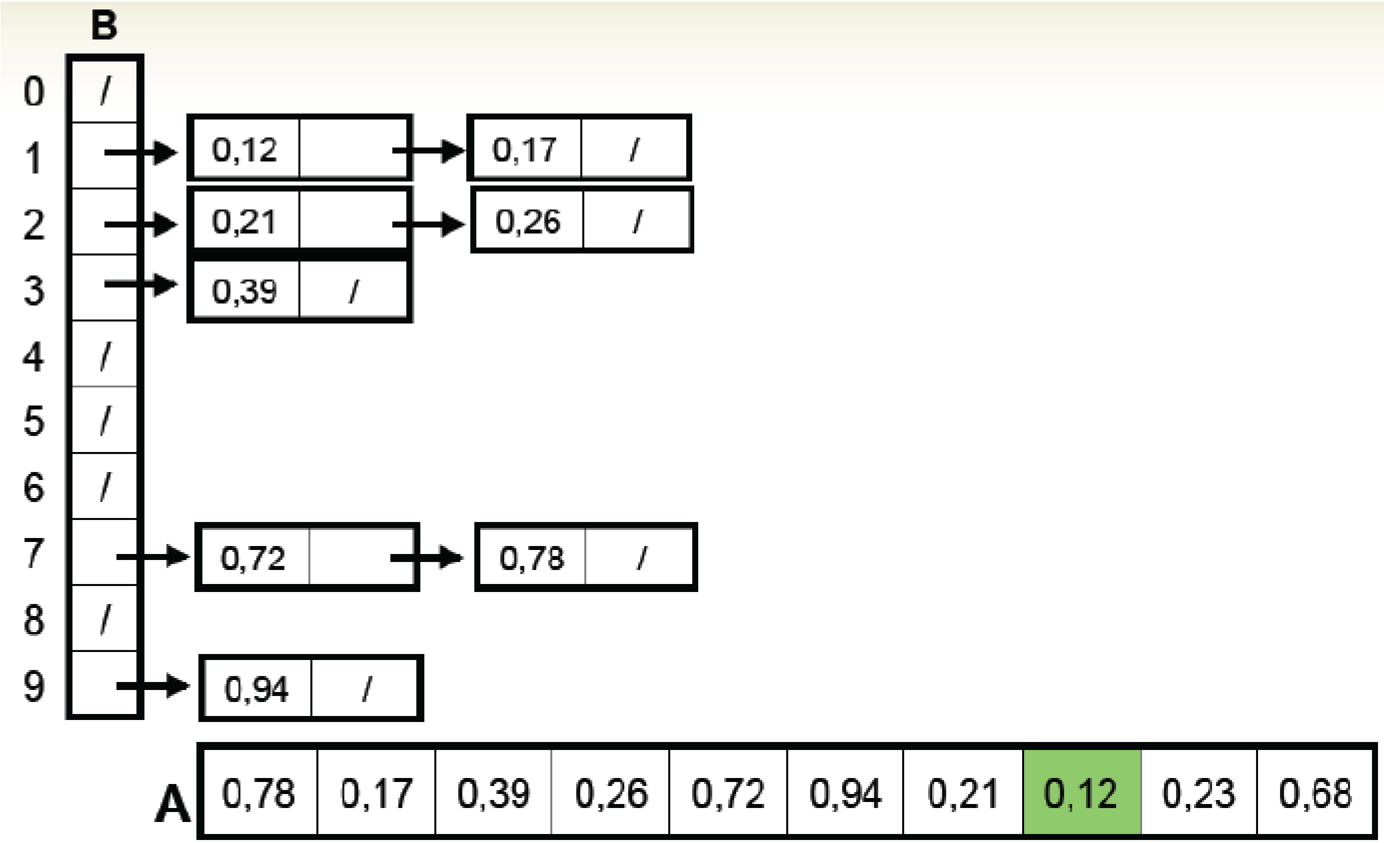
Bucket Sort - Exemplo



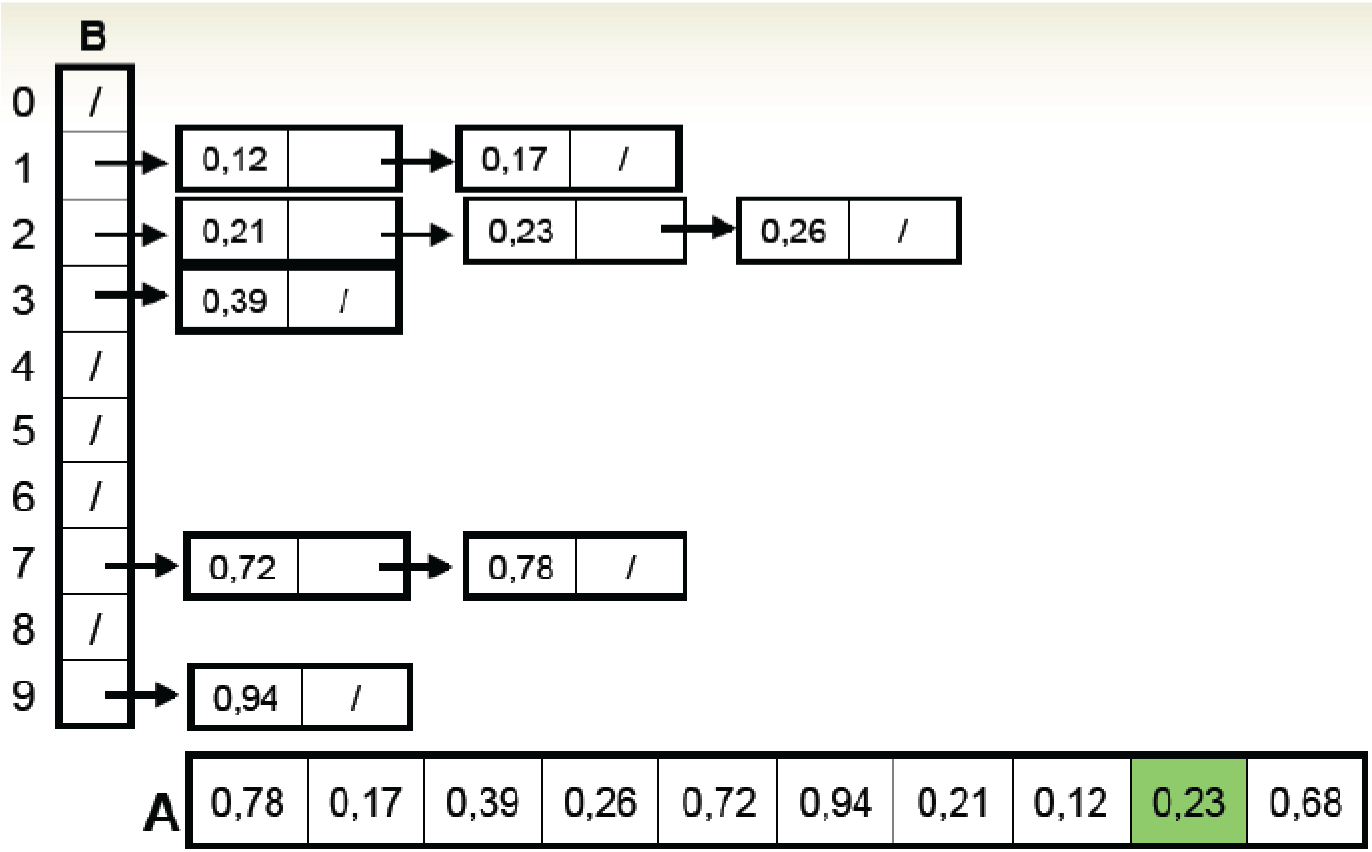
Bucket Sort - Exemplo



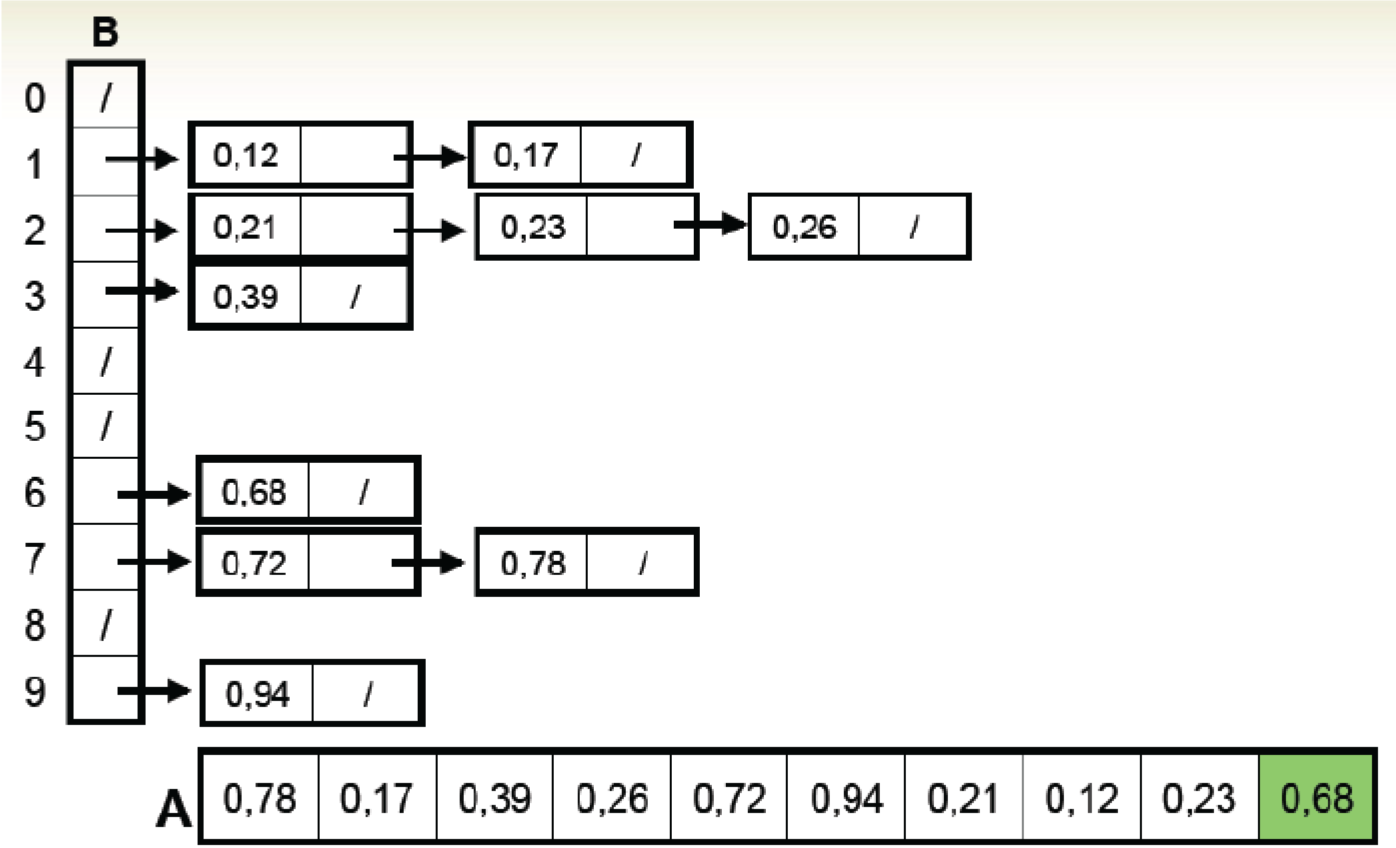
Bucket Sort - Exemplo



Bucket Sort - Exemplo



Bucket Sort - Exemplo



Exercício

1) Execute e apresente os passos do CountSort sobre o arranjo A (6, 0 ,2, 0, 1, 3, 4, 6, 1, 3, 2)

Algoritmos de Tempo Linear

- **Foram estudados três algoritmos de ordenação linear (tempo Θ(n))**
  - Que são portanto melhores que os algoritmos de ordenação por comparação (tempo $O(n \log_2 n)$)

- **Entretanto, nem sempre é interessante utilizar um destes três algoritmos:**
  - Todos eles pressupõem algo sobre os dados de entrada a serem ordenados

Referências

Básica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.

Complementar

- ASCENCIO, Ana Fernanda Gomes; ARAUJO, Graziela Santos. Estruturas de Dados: Algoritmos, análise da complexidade e implementações em Java e C/C++. Pearson Prentice Hall, 2010
- Notas de aula: prof. Rafael Fernandes. DAI/IFMA
- Notas de aula: prof. Geraldo Braz. DEINF/UFMA

Perguntas....