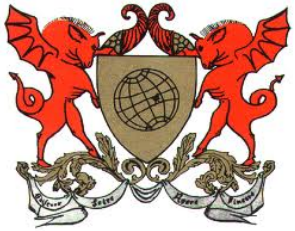
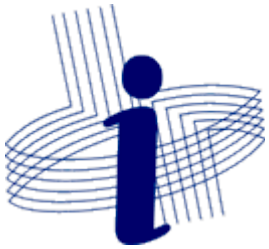


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF 112
Programação 2
Aula “10”
Ordenação – 4
Ordenação em tempo linear

Ordenação em tempo linear

- Até agora, os algoritmos mais eficientes que vimos possuem complexidade de tempo O(n log n).
- Teria como desenvolver algum algoritmo melhor?
- Seria possível criar um algoritmo de ordenação O(1) ? O(log n) ? O(n) ?

Ordenação em tempo linear

- Os algoritmos vistos até agora são chamados de algoritmos que realizam ordenação por **comparação**.
- Isso porque a sequencia ordenada é feita com base na comparação entre os elementos da entrada.
- Pode-se provar que qualquer método de ordenação por **comparação** tem complexidade de, no mínimo, O(n log n) no pior caso.
- Porém, sob certas circunstâncias, é possível desenvolver algoritmos (que não se baseiam em comparação) com tempo O(n).

Ordenação em tempo linear

- Imagine que temos um arranjo contendo n elementos e que tais elementos podem ser apenas os números 1, 2 e 3 (ou, então, podem ter chaves com apenas esses três valores).
- Como poderíamos ordenar tal arranjo de forma eficiente?
- Ex: [3,1,1,2,2,2,3,3,3,1,3,2,1,2,3]

Ordenação em tempo linear

- Ordenação por contagem: a ordenação por contagem pressupõe que cada um dos n elementos a serem ordenados é um inteiro entre 0 e k, para algum inteiro k. Quando k = O(n), a ordenação é executada em tempo O(n).

- A ideia da ordenação por contagem é determinar, para cada elemento de entrada *x*, o número de elementos menores que *x*.

Ordenação em tempo linear

- Com isso, podemos saber a posição onde cada elemento seria inserido no array. Por exemplo, se há 10 números menores que 50 na entrada, então o número 50 deveria ser inserido na posição 11 do array.

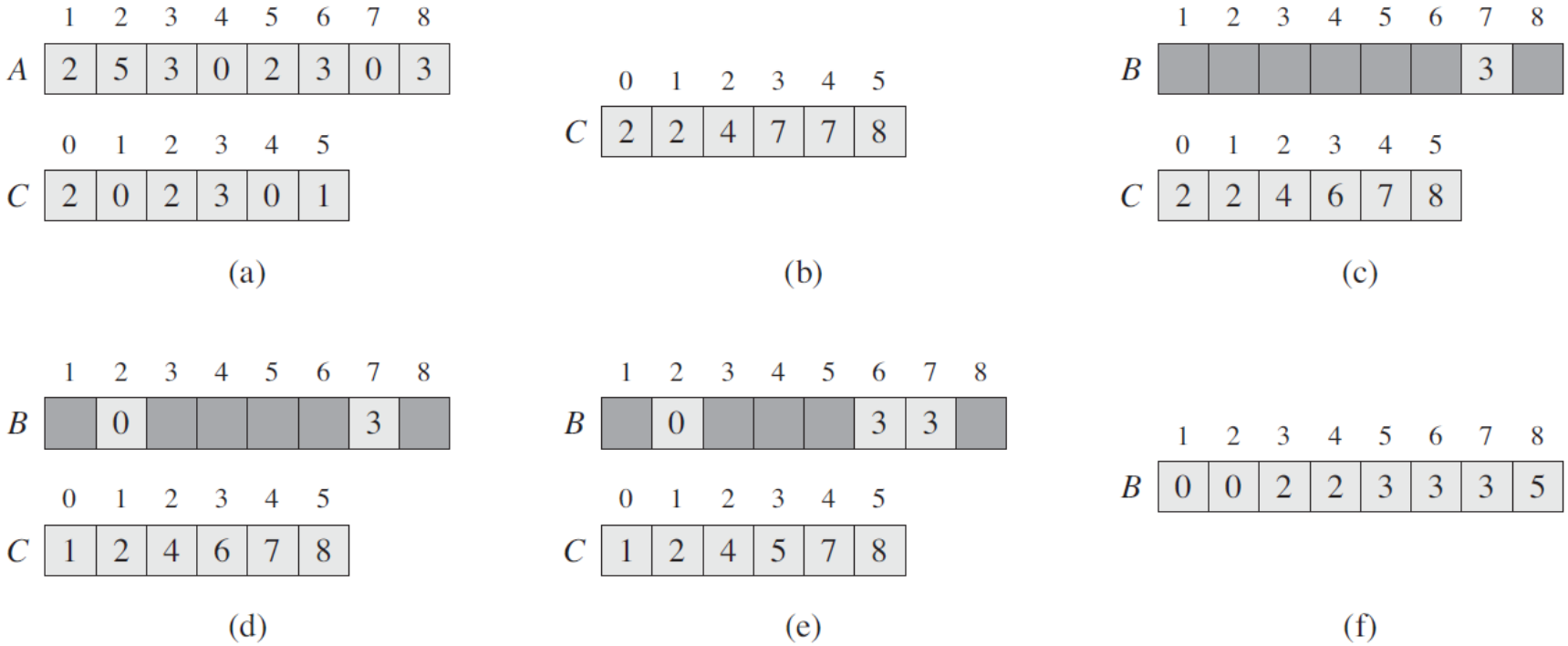
- Deve-se fazer pequenas modificações nesse algoritmo para tratar o caso onde podem haver várias entradas com a mesma chave.

Ordenação em tempo linear

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

- A[1..n] é o vetor de entrada;
- B[1..n] é o vetor que conterá a saída ordenada;
- C[0..k] é usado como armazenamento temporário

Ordenação em tempo linear



- (a) Vetor A e C após a inicialização de C com contagem dos elementos;
- (b) Vetor C agora contém quantos elementos <= i (sendo i os índices de C);
- (c)-(e) Vetor B e C depois de 1, 2 e 3 iterações do loop das linhas 10-12;
- (f) Vetor B ordenado.

Ordenação em tempo linear

- Exercício: continuando o exemplo dado, rastreie o código do countingSort para o seguinte arranjo: [2,5,3,0,2,3,0,3]

Ordenação em tempo linear

- O counting-sort é um algoritmo in-place?
- Ele é estável?
- Qual a complexidade dele?
- Quais as principais vantagens e desvantagens?

Ordenação em tempo linear

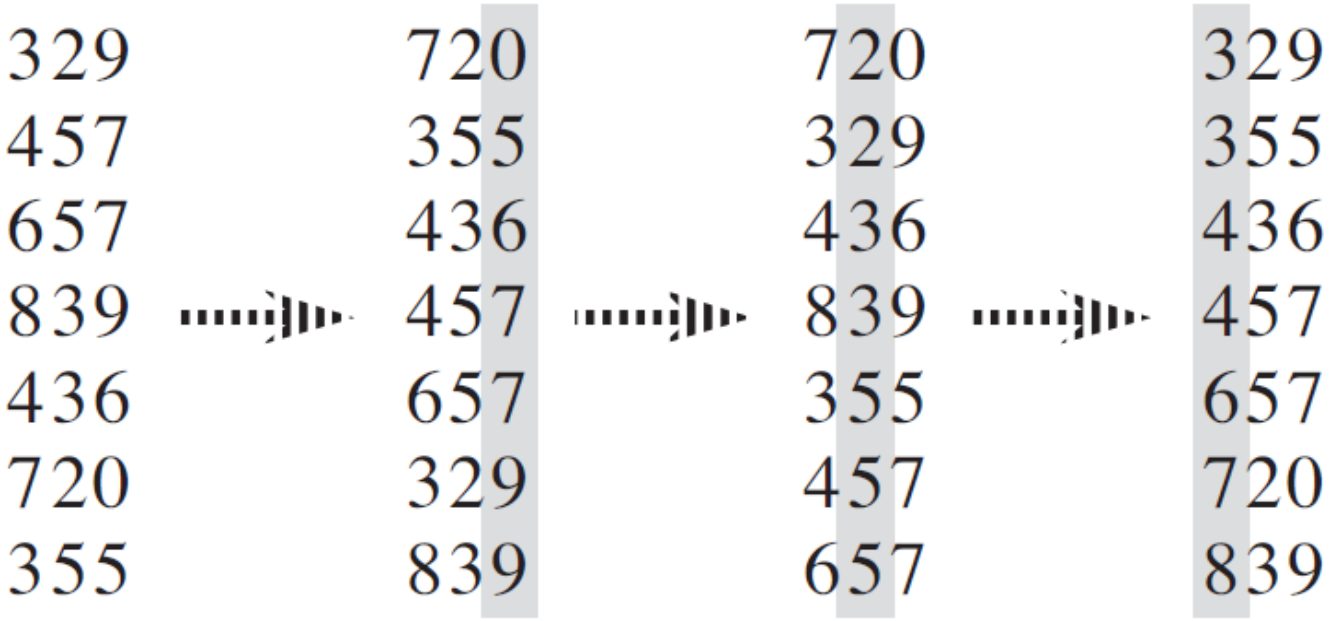
- O counting-sort é um algoritmo in-place? R: não
- Ele é estável? R: sim!!!!
- Qual a complexidade dele? R: O(n+k) = O(n) para k sendo O(n).
- Quais as principais vantagens e desvantagens?

  R: pode gastar muita memória, pode ser eficiente dependendo da entrada... funciona apenas para chaves inteiras (ou para chaves que possam ser convertidas para inteiros..)

Ordenação em tempo linear

- RadixSort: algoritmo que era utilizado nas máquinas de ordenação de cartões.
- Ele utiliza chaves representadas por números inteiros ou arrays de caracteres.
- Há duas versões: o MSD radix sort (most significant digit) e LSD radix sort (least significant digit).
- Vamos nos concentrar no que trabalha com os dígitos menos significativos. Porém, vamos dar uma ideia da outra versão também.

Ordenação em tempo linear



Adaptado do livro do Cormen, terceira edição.

Ordenação em tempo linear

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

Adaptado do livro do Cormen, terceira edição.

Ordenação em tempo linear

- MSD RadixSort:
  - Pegue as chaves e as distribua em *bins* ou “baldes” contendo os dígitos mais significativos.
  - Ordene de forma recursiva cada balde de elementos (ao ordenar recursivamente, considere o próximo dígito como o mais significativo) e, então, os concatene.
- Exemplo: [50,98,96,101,2]
- [**0**50,**0**98,**0**96,**1**01,**0**02] → [050,098,096,002] [101]
- [0**5**0,0**9**8,0**9**6,0**0**2] [101] → [002] [050] [098,096] [101]
- A ordenação recursiva só precisa ser chamada para o balde com mais de 1 elemento
- [002][050][09**8**,09**6**][101] → [002][050][096][098][101]
- [002][050][096][098][101] → [002][050][096,098][101]
- [002][050][096,098][101] → [002,050,096,098][101]
- [002,050,096,098][101] → [002,050,096,098,101]
- Problema: uso de memória!

Ordenação em tempo linear

- LSD RadixSort:
  - Pegue as chaves e ordene os elementos com base no dígito menos significativo utilizando algum método **estável** de ordenação (ex: counting-sort – note que se utilizarmos dígitos decimais a complexidade disso seria O(n+10) = O(n)).
  - Repita o processo para cada outro dígito (do menos significativo para o mais significativo).
- Exemplo: [50,98,96,101,2]
- [05**0**,09**8**,09**6**,10**1**,00**2**]
- [0**5**0,1**0**1,0**0**2,0**9**6,0**9**8]
- [**1**01,**0**02,**0**50,**0**96,**0**98]
- [002,050,096,098,101]

Ordenação em tempo linear

- LSD RadixSort:
  - Pegue as chaves e ordene os elementos com base no dígito menos significativo utilizando algum método **estável** de ordenação (ex: counting-sort).
  - Repita o processo para cada outro dígito (do menos significativo para o mais significativo).
- Por que o método de ordenação utilizado deve ser estável?
- Qual a complexidade do algoritmo?

Ordenação em tempo linear

- LSD RadixSort:
  - Pegue as chaves e ordene os elementos com base no dígito menos significativo utilizando algum método **estável** de ordenação (ex: counting-sort).
  - Repita o processo para cada outro dígito (do menos significativo para o mais significativo).
- Por que o método de ordenação utilizado deve ser estável? R: veja o exemplo anterior... a ordenação feita em um passo poderia ser perdida devido à ordenação não estável!
- Qual a complexidade do algoritmo? R: O(nk), onde k é o número de dígitos.

Ordenação em tempo linear

- Possíveis problemas do radixSort:
  - Uso de memória do algoritmo escolhido para fazer a ordenação estável.
  - Restrições sobre as chaves serem inteiras/strings.
  - Possíveis constantes “escondidas” na notação O pode deixar o algoritmo, na prática, mais lento que outros algoritmos O(n log n).