

# Algoritmos de Ordenação - Counting e Radix

Prof<sup>a</sup>. Rose Yuri Shimizu

# Roteiro

## 1 Algoritmos de Ordenação

- Counting Sort
- Radix Sort

# Roteiro

- 1 Algoritmos de Ordenação
  - Counting Sort
  - Radix Sort

# Counting Sort

Em breve ;).

# Roteiro

- 1 Algoritmos de Ordenação
  - Counting Sort
  - Radix Sort

# Radix Sort

## Ordenação: compara-se as chaves/dados

- Detalhes do processamento de comparação são “abstraídos”
- Como o processador faz as comparações?
  - ▶ Como os dados são representados por bits, as comparações ocorrem bit a bit

Todos os bits dos dados são processados a cada iteração

# Radix Sort

## Ordenação: compara-se as chaves/dados

- Detalhes do processamento de comparação são “abstraídos”
- Como o processador faz as comparações?
  - ▶ Como os dados são representados por bits, as comparações ocorrem bit a bit
  - ▶ Todos os bits dos dados são processados a cada iteração

# Radix Sort

## Como os dados são compostos por dígitos

- Ideia: diminuir as comparações de cada etapa
  - ▶ ao invés de comparar a chave inteira em toda iteração ou recursão
  - ▶ compara-se somente parte dela
  - ▶ ordenando parcialmente
- Exemplo:
  - ▶ ao procurar/posicionar uma palavra em um dicionário
  - ▶ cada letra que forma a palavra
  - ▶ contribui para localizar a página exata da palavra
  - ▶ cada letra restringe as possibilidades de posições



# Radix Sort

170 045 075 090 802 024 002 066

# Radix Sort

170 045 075 090 802 024 002 066

170 090 802 002 024 045 075 066

# Radix Sort

170 090 802 002 024 045 075 066

# Radix Sort

170 090 802 002 024 045 075 066

802 002 024 045 066 170 075 090

# Radix Sort

802 002 024 045 066 170 075 090

# Radix Sort

802 002 024 045 066 170 075 090

002 024 045 066 075 090 170 802

# Radix Sort

170 045 075 090 802 024 002 066



002 024 045 066 075 090 170 802

- Ordena pela unidade, dezena, centena, etc.
- Problema?

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte



# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte

▶ até 255  $\rightarrow 0, 1, 2, 3, \dots, 255$

▶  $257 = 00000001\ 00000001 \rightarrow 256 + 1$

▶  $258 = 00000001\ 00000010 \rightarrow 256 + 2$

▶  $65792 = 00000001\ 00000001\ 00000000 \rightarrow 65536 + 256 + 0$

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - ▶ até 255  $\rightarrow 0, 1, 2, 3, \dots, 255$
  - ▶  $257 = 00000001\ 00000001 \rightarrow 256 + 1$
  - ▶  $258 = 00000001\ 00000010 \rightarrow 256 + 2$
  - ▶  $65792 = 00000001\ 00000001\ 00000000 \rightarrow 65536 + 256 + 0$

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - ▶ até 255  $\rightarrow 0, 1, 2, 3, \dots, 255$
  - ▶  $257 = 00000001\ 00000001 \rightarrow 256 + 1$
  - ▶  $258 = 00000001\ 00000010 \rightarrow 256 + 2$
  - ▶  $65792 = 00000001\ 00000001\ 00000000 \rightarrow 65536 + 256 + 0$

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - ▶ até 255  $\rightarrow 0, 1, 2, 3, \dots, 255$
  - ▶  $257 = 00000001\ 00000001 \rightarrow 256 + 1$
  - ▶  $258 = 00000001\ 00000010 \rightarrow 256 + 2$
  - ▶  $65792 = 00000001\ 00000001\ 00000000 \rightarrow 65536 + 256 + 0$

# Radix Sort

- Envolve operações custosas
  - ▶ 802:  $802\%10$ ,  $(802/10)\%10$ ,  $(802/100)\%10$
- Como o computador trabalha com a base binária, melhor dividir as partes da chaves (dígitos) por bits ou bytes
- Exemplo com inteiros: partes de 1 byte
  - ▶ até 255  $\rightarrow 0, 1, 2, 3, \dots, 255$
  - ▶  $257 = 00000001\ 00000001 \rightarrow 256 + 1$
  - ▶  $258 = 00000001\ 00000010 \rightarrow 256 + 2$
  - ▶  $65792 = 00000001\ 00000001\ 00000000 \rightarrow 65536 + 256 + 0$

# Radix Sort

- Radix:
  - ▶ ordenar pela a raiz(radix) da representação dos dados
  - ▶ extraíndo o i-ésimo dígito da chave
- Obs.: caso a chave seja pequena não compensa a extração dos bits: use o *counting sort*



# Radix Sort

Para extrair o D-ésimo dígito da chave

```
1  #define bitsbyte 8
2  #define bytesword 4
3
4  //00000001 << 8 = 00000001 00000000 = 2^8 = 256
5  #define R (1 << bitsbyte)
6
7  //extraíndo o D-ésimo dígito de N
8  #define digit(N,D) (((N) >> ((D)*bitsbyte)) & (R-1))
9
```

- $((N) \gg ((D) * \text{bitsbyte}))$ : remove os primeiros D bytes
- $\&(R - 1)$ : pegando os últimos bitbytes (máscara)

# Radix Sort

## Para extrair o D-ésimo dígito da chave

- $((N) \gg ((D) * \text{bitsbyte}))$ : remove os primeiros D bytes
- $\&(R - 1)$ : pegando os últimos bitbytes (máscara)

```
1      65792 = 00000001 00000001 00000000
2
3      //pegando o dígito 1
4      00000001 00000001 00000000 >> 1*8 =
5      00000000 00000001 00000001
6
7      //aplicando a máscara
8      00000000 00000001 00000001
9      & 00000000 00000000 11111111
10     -----
11     00000000 00000000 00000001
12
```

- E para strings? como acessar o byte-ésimo dígito?

# Radix Sort

## Métodos de classificação - LSD

- A partir dígito menos significativo (least significant digit – LSD):  
direita para esquerda
  - ▶ Ordena estavelmente chaves de comprimento fixo
    - ★ Tamanho da palavra (word) que representa o dado
    - ★ int:  $4 * 8 \text{ bits} = 32 \text{ bits} = 4 \text{ bytes}$
    - ★ strings de tamanho  $W$
- Complexidade: aprox.  $7WN + 3WR$  acessos:
  - ▶  $N$  chaves
  - ▶ chaves de tamanho  $W$
  - ▶ cujo alfabeto são de tamanho  $R$
- $R$ , em geral, é muito menor que  $N$ , portanto a complexidade é proporcional a  $WN$
- Vamos ver os códigos para inteiros e strings.

# Radix Sort

## Métodos de classificação - MSD

- A partir dígito mais significativo (most significant digit – MSD): esquerda para direita
  - ▶ Ordenação de propósito geral: chaves com tamanhos variáveis
    - ★ Conjunto de várias palavras
    - ★ string:  $N * 8 \text{ bits} = N * 1 \text{ byte}$ ,  $N$  variável
  - ▶ Usa-se primeiro o counting sort
  - ▶ Depois, recursivamente, ordena-se os sub-vetores de cada caractere
- Complexidade: aprox.  $7WN + 3WR$  acessos
- Vamos ver os códigos para inteiros e strings.

# Radix Sort - MSD - strings

use key-indexed counting on first character

recursively sort subarrays

count frequencies	transform counts to indices	distribute and copy back	indices at completion of distribute phase
0 she	0 0	0 are	0 0 0
1 sells	1 a 0	1 by	1 a 1
2 seashells	2 b 1	2 she	2 b 2
3 by	3 c 1	3 sells	3 c 2
4 the	4 d 0	4 seashells	4 d 2
5 sea	5 e 0	5 sea	5 e 2
6 shore	6 f 0	6 shore	6 f 2
7 the	7 g 0	7 shells	7 g 2
8 shells	8 h 0	8 she	8 h 2
9 she	9 i 0	9 sells	9 i 2
10 sells	10 j 0	10 surely	10 j 2
11 are	11 k 0	11 seashells	11 k 2
12 surely	12 l 0	12 the	12 l 2
13 seashells	13 m 0	13 the	13 m 2
	14 n 0		14 n 2
	15 o 0		15 o 2
	16 p 0		16 p 2
	17 q 0		17 q 2
	18 r 0		18 r 2
	19 s 0		19 s 12
	20 t 10		20 t 14
	21 u 2		21 u 14
	22 v 0		22 v 14
	23 w 0		23 w 14
	24 x 0		24 x 14
	25 y 0		25 y 14
	26 z 0		26 z 14
	27 0		27 14

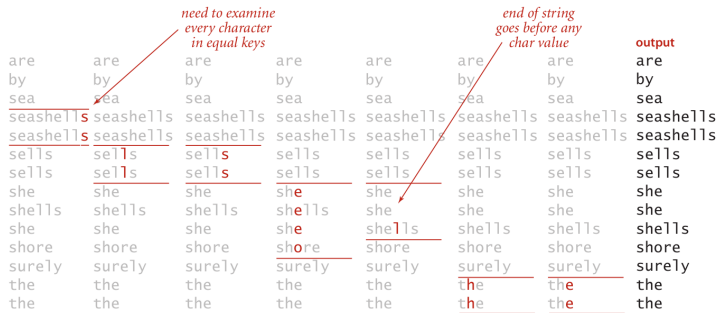
start of s subarray  
1 + end of s subarray

Trace of MSD string sort: top level of sort(a, 0, 14, 0)

# Radix Sort - MSD - strings

input		d							
she	are	are	are	are	are	are	are	are	are
sells	by	by	by	by	by	by	by	by	by
seashells	she	sells	seashells	sea	sea	sea	seashells	seashells	seashells
by	sells	seashells	sea	seashells	seashells	seashells	seashells	seashells	seashells
the	seashells	sea	seashells	seashells	seashells	seashells	seashells	seashells	seashells
sea	sea	sells	sells	sells	sells	sells	sells	sells	sells
shore	shore	seashells	sells	sells	sells	sells	sells	sells	sells
the	shells	she	she	she	she	she	she	she	she
shells	she	shore	shore	shore	shore	shore	shells	shells	shells
she	sells	shells	shells	shells	shells	shells	shore	shore	shore
sells	surely	she	she	she	she	she	she	she	she
are	seashells	surely	surely	surely	surely	surely	surely	surely	surely
surely	the	the	the	the	the	the	the	the	the
seashells	the	the	the	the	the	the	the	the	the

# Radix Sort - MSD - strings



# Radix Sort - MSD - inteiros

170 045 075 090 802 024 002 066

045 075 090 024 002 066 170 802



# Radix Sort - MSD - inteiros

045 075 090 024 002 066 170 802

002 802 024 045 066 075 170 090

# Radix Sort - MSD - inteiros

00**2** 80**2** 02**4** 04**5** 06**6** 07**5** 17**0** 09**0**

17**0** 09**0** 00**2** 80**2** 02**4** 04**5** 07**5** 06**6**

# Radix Sort - MSD - inteiros

170 045 075 090 802 024 002 066



170 090 002 802 024 045 075 066

# Radix Sort

## Métodos de ordenação

algorithm	stable?	inplace?	order of growth of typical number calls to <code>charAt()</code> to sort $N$ strings from an $R$ -character alphabet (average length $w$ , max length $W$ )		sweet spot
			running time	extra space	
<i>insertion sort for strings</i>	yes	yes	between $N$ and $N^2$	1	small arrays, arrays in order
<i>quicksort</i>	no	yes	$N \log^2 N$	$\log N$	general-purpose when space is tight
<i>mergesort</i>	yes	no	$N \log^2 N$	$N$	general-purpose stable sort
<i>3-way quicksort</i>	no	yes	between $N$ and $N \log N$	$\log N$	large numbers of equal keys
<i>LSD string sort</i>	yes	no	$NW$	$N$	short fixed-length strings
<i>MSD string sort</i>	yes	no	between $N$ and $Nw$	$N + WR$	random strings