

# SBVORIN: Organização e Recuperação da Informação

## Aula 08: Algoritmos de Ordenação Linear em Strings

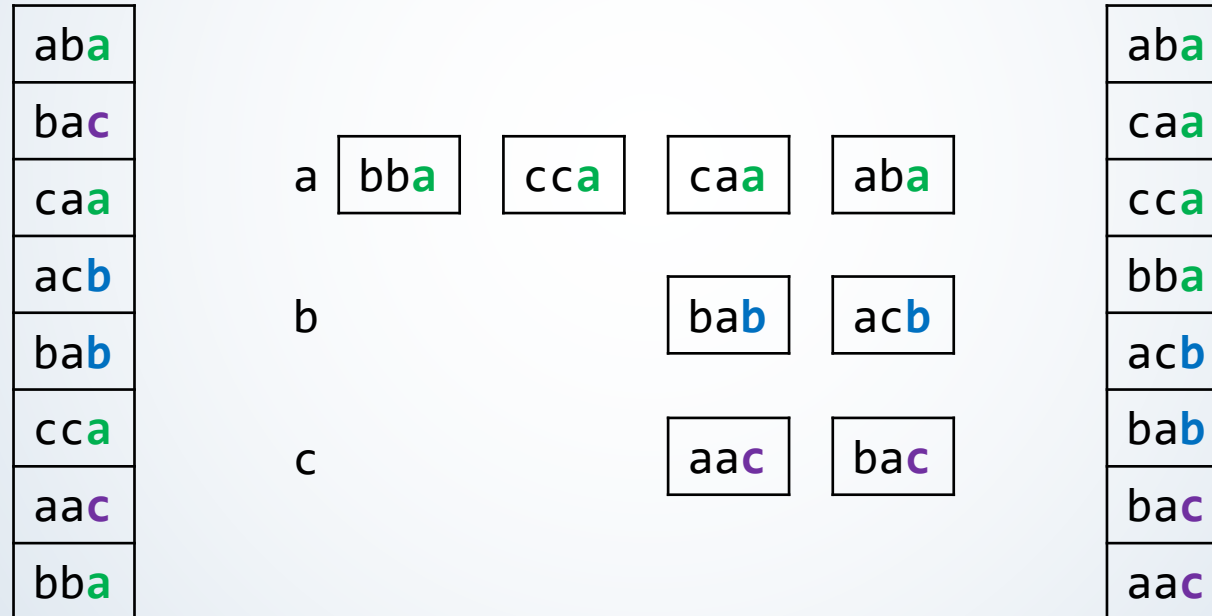
# Radix Sort

- ▶ Usaremos a ideia do Radix Sort para ordenar Strings;
- ▶ Para números:
  - ▶ Os valores de entrada, escritos em alguma base numérica, têm exatamente  $d$  dígitos;
  - ▶ A ordenação é realizada em  $d$  passos: um dígito por vez, começando a partir dos menos significativos, ou seja, os dígitos mais à direita;
- ▶ Podemos extrapolar essa ideia para Strings.

# Radix Sort

## Exemplo com 3 Dígitos

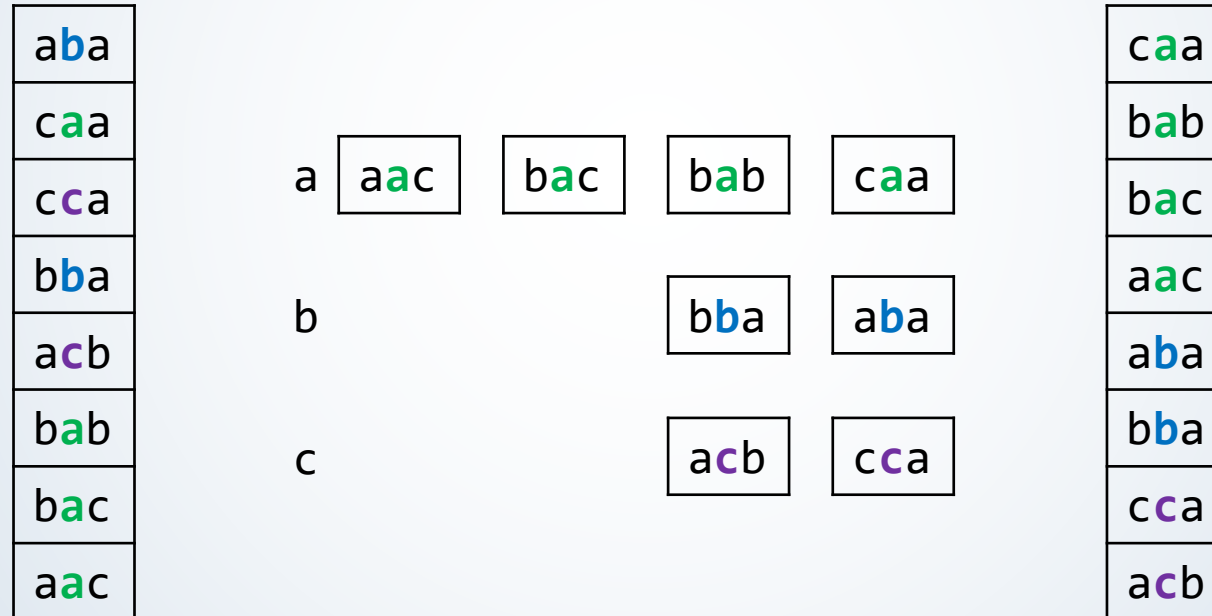
- Usaremos 3 filas para organizar a ordenação, uma por dígito.



# Radix Sort

## Exemplo com 3 Dígitos

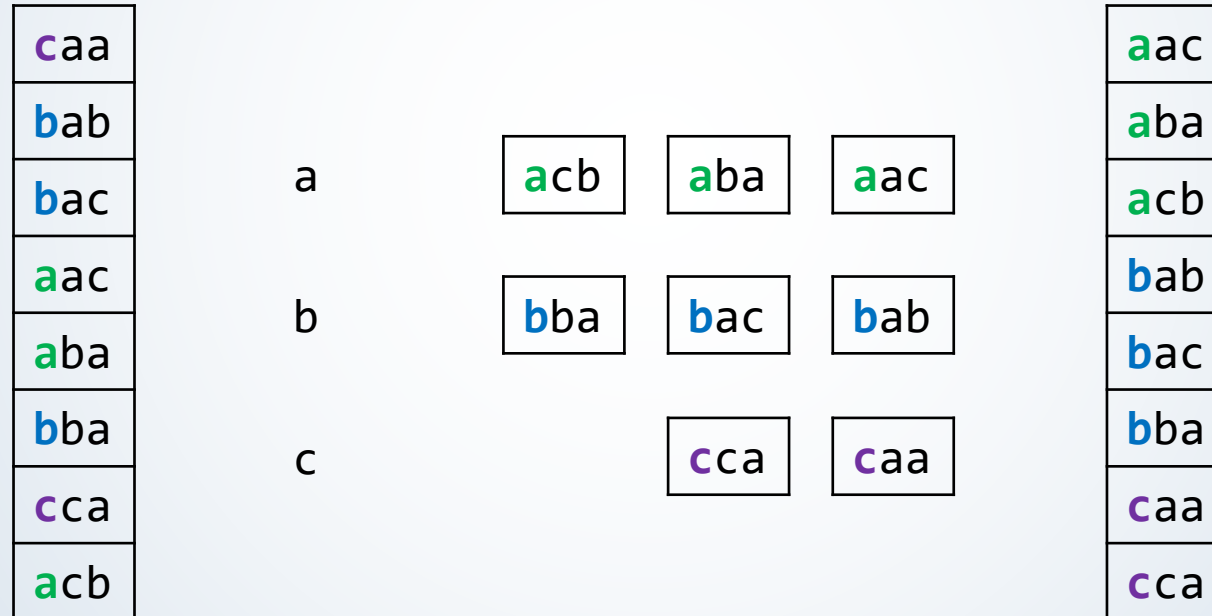
- Usaremos 3 filas para organizar a ordenação, uma por dígito.



# Radix Sort

## Exemplo com 3 Dígitos

- ➔ Usaremos 3 filas para organizar a ordenação, uma por dígito.



# Ordenação de Strings

## Least Significant Digit (LSD)

- Ordenação de Strings LSD (Radix Sort):
  - Considera os caracteres da direita para a esquerda;
  - Classifique, de forma estável, usando o  $d$ -ésimo caractere como chave:

		sort key (d = 2)	sort key (d = 1)	sort key (d = 0)
0	d	a	b	
1	a	d	d	
2	c	a	b	
3	f	a	d	
4	f	e	e	
5	b	a	d	
6	d	a	d	
7	b	e	e	
8	f	e	d	
9	b	e	d	
10	e	b	b	
11	a	c	e	

0	d	a	b
1	c	a	b
2	e	b	b
3	a	d	d
4	f	a	d
5	b	a	d
6	d	a	d
7	f	e	d
8	b	e	d
9	f	e	e
10	b	e	e
11	a	c	e

0	d	a	b
1	c	a	b
2	f	a	d
3	b	a	d
4	d	a	d
5	e	b	b
6	a	c	e
7	a	d	d
8	f	e	d
9	b	e	d
10	f	e	e
11	b	e	e

0	a	c	e
1	a	d	d
2	b	a	d
3	b	e	d
4	b	e	e
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	d
11	f	e	e

# Ordenação de Strings

## *Least Significant Digit (LSD)*

```
public class LSD
{
    public static void sort(String[] a, int W)
    {
        int R = 256;
        int N = a.length;
        String[] aux = new String[N];

        for (int d = W-1; d >= 0; d--)
        {
            int[] count = new int[R+1];
            for (int i = 0; i < N; i++)
                count[a[i].charAt(d) + 1]++;
            for (int r = 0; r < R; r++)
                count[r+1] += count[r];
            for (int i = 0; i < N; i++)
                aux[count[a[i].charAt(d)]++] = a[i];
            for (int i = 0; i < N; i++)
                a[i] = aux[i];
        }
    }
}
```

Strings de tamanho  
fixo

Tamanho do  
Alfabeto

Fazer a contagem  
indexada por chave,  
para cada dígito, da  
direita para a  
esquerda

Contagem indexada  
por chave



# Ordenação de Strings

## *Least Significant Digit (LSD)*

algorithm	guarantee	random	extra space	stable?
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	✓
mergesort	$N \lg N$	$N \lg N$	$N$	✓
quicksort	$1.39 N \lg N^*$	$1.39 N \lg N$	$c \lg N$	
heapsort	$2 N \lg N$	$2 N \lg N$	1	
LSD sort	$2 W (N + R)$	$2 W (N + R)$	$N + R$	✓



# Ordenação de Strings

## Reverse LSD

- Ordenação de Strings Reverse LSD:
  - Considera os caracteres da esquerda para a direita;
  - Classifique, de forma estável, usando o  $d$ -ésimo caractere como chave:

		sort key (d = 0)	sort key (d = 1)	sort key (d = 2)
		↓	↓	↓
0	d a b	a d d	b a d	c a b
1	a d d	a c e	c a b	d a b
2	c a b	<b>b</b> a d	d a b	e b b
3	f a d	<b>b</b> e e	d a d	b a d
4	f e e	<b>b</b> e d	f a d	d a d
5	b a d	c a b	e b b	f a d
6	d a d	d a b	a c e	a d d
7	b e e	d a d	a d d	b e d
8	f e d	e b b	b e e	f e d
9	b e d	f a d	b e d	a c e
10	e b b	f e e	f e e	b e e
11	a c e	f e d	f e d	f e e

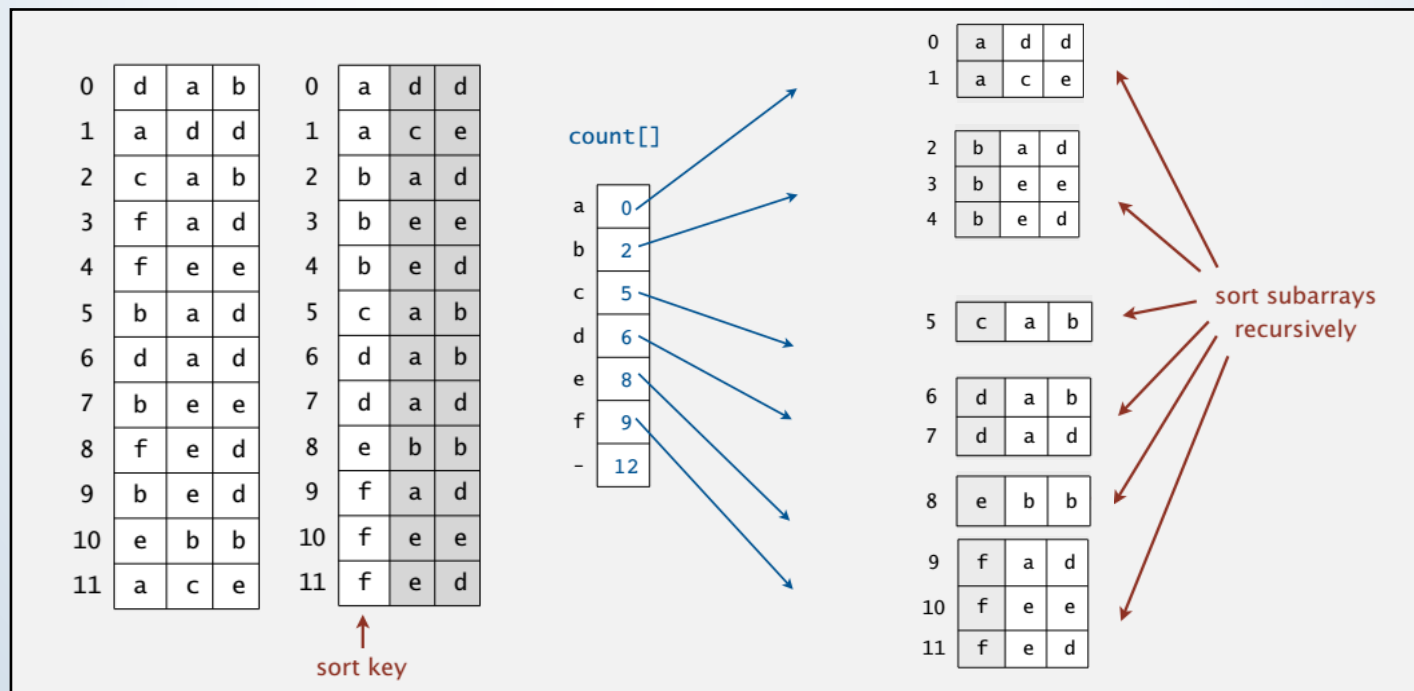
**not sorted!**

# Ordenação de Strings

## *Most Significant Digit (MSD)*

### ➤ Ordenação de Strings MSD (Radix Sort):

- Particione o array em  $R$  partes de acordo com o primeiro caractere, utilizando a contagem indexada por chave;
- Classificar, recursivamente, todas as Strings que começam com cada caractere, pois as contagens indexadas por chave delineiam subarrays a serem classificados.



# Ordenação de Strings

## Most Significant Digit (MSD)

- ➔ Ordenação de Strings MSD (Radix Sort):

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

are	are	are	are	are	are	are	output
by	by	by	by	by	by	by	by
sea	sea	sea	sea	sea	sea	sea	sea
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
sells	sells	sells	sells	sells	sells	sells	sells
sells	sells	sells	sells	sells	sells	sells	sells
she	she	she	she	she	she	she	she
shore	shore	shore	shells	she	she	she	she
shells	hells	shells	she	shells	shells	shells	shells
she	she	she	shore	shore	shore	shore	shore
surely	surely	surely	surely	surely	surely	surely	surely
the	the	the	the	the	the	the	the
the	the	the	the	the	the	the	the

need to examine every character in equal keys

end of string goes before any char value

# Ordenação de Strings

## *Most Significant Digit (MSD)*

- ▶ Como seria para Strings de tamanho variável?
  - ▶ Basta tratar as Strings como se tivessem um caractere extra no final que é menor do que qualquer caractere.

0	s	e	a	-1						
1	s	e	a	s	h	e	l	l	s	-1
2	s	e	l	l	s	-1				
3	s	h	e	-1						
4	s	h	e	-1						
5	s	h	e	l	l	s	-1			
6	s	h	o	r	e	-1				
7	s	u	r	e	l	y	-1			

she before she'lls

```
private static int charAt(String s, int d)
{
    if (d < s.length()) return s.charAt(d);
    else return -1;
}
```

# Ordenação de Strings

## *Most Significant Digit (MSD)*

```
public static void sort(String[] a)
{
    aux = new String[a.length];
    sort(a, aux, 0, a.length - 1, 0);
}

private static void sort(String[] a, String[] aux, int lo, int hi, int d)
{
    if (hi <= lo) return;
    int[] count = new int[R+2];
    for (int i = lo; i <= hi; i++)
        count[charAt(a[i], d) + 2]++;
    for (int r = 0; r < R+1; r++)
        count[r+1] += count[r];
    for (int i = lo; i <= hi; i++)
        aux[count[charAt(a[i], d) + 1]++] = a[i];
    for (int i = lo; i <= hi; i++)
        a[i] = aux[i - lo];

    for (int r = 0; r < R; r++)
        sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);
}
```

recycles aux[] array  
but not count[] array

key-indexed counting

sort R subarrays recursively



# Ordenação de Strings


## *Most Significant Digit (MSD)*

- ▶ Muito lento para pequenos subarrays:
  - ▶ Cada chamada de método precisa de seu próprio `count[]`
  - ▶ ASCII (256 contagens): 100x mais lento do que a passagem de cópia para  $N = 2$
  - ▶ Unicode (65.536 contagens): 32.000 vezes mais lento para  $N = 2$
- ▶ Grande número de pequenos subarrays em função da recursão;
- ▶ **Solução:** Utilizar o Insertion Sort para pequenos subarrays.

# Ordenação de Strings

## *Most Significant Digit (MSD)*

algorithm	guarantee	random	extra space	stable?	operations on keys
<b>insertion sort</b>	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	✓	compareTo()
<b>mergesort</b>	$N \lg N$	$N \lg N$	$N$	✓	compareTo()
<b>quicksort</b>	$1.39 N \lg N^*$	$1.39 N \lg N$	$c \lg N$		compareTo()
<b>heapsort</b>	$2 N \lg N$	$2 N \lg N$	1		compareTo()
<b>LSD sort</b> †	$2 W (N + R)$	$2 W (N + R)$	$N + R$	✓	charAt()
<b>MSD sort</b> ‡	$2 W (N + R)$	$N \log_R N$	$N + D R$	✓	charAt()


  
 $D$  = function-call stack depth  
 (length of longest prefix match)



# Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

GOODRICH M. T.; TAMASSIA, R. **Estruturas de Dados & Algoritmos em Java**. Porto Alegre: Bookman, 2013. 700 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.