# ALGORITMOS E ESTRUTURAS DE DADOS

## radix sort

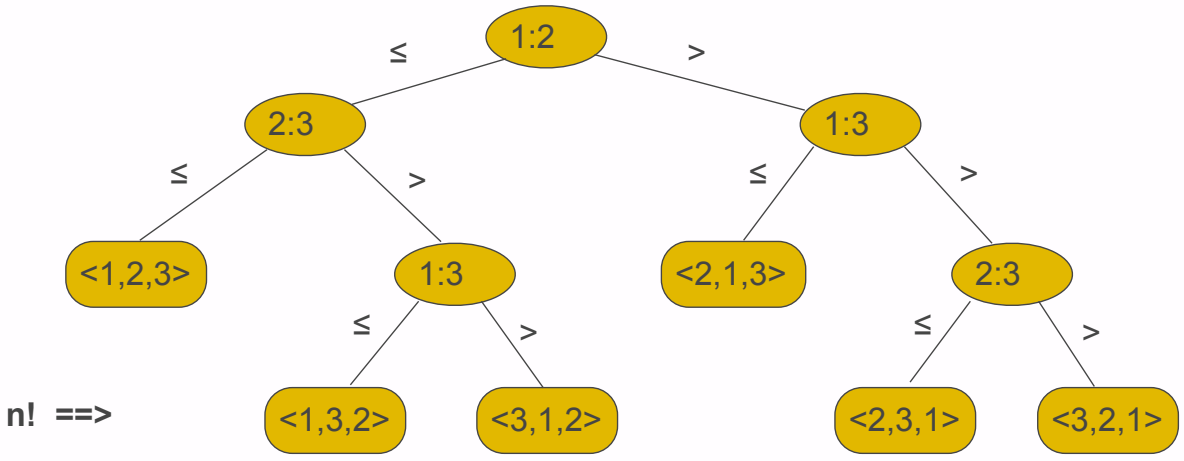
2010-2011

Carlos Lisboa Bento

## Radix Sort

Limites para os algoritmos de comparação de chaves

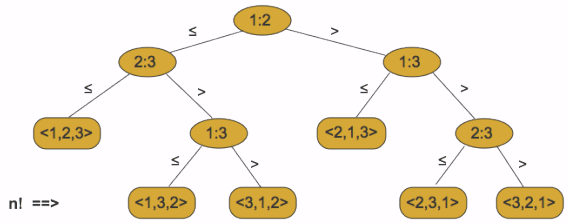
Árvore de decisão para o ordenamento por inserção com três elementos



# Radix Sort

## Limites para os algoritmos de comparação de chaves

Árvore de decisão para o ordenamento por inserção com três elementos




$$n! \leq l \leq 2^h$$

$$h \geq \lg(n!)$$
$$= \Omega(n \lg n)$$



# Counting Sort

Pressupostos:

- Cada elemento a ordenar é um inteiro
- Os valores são entre 0 e k c/ k um inteiro dado
-
- Para k = O(n) o ordenamento tem complexidade θ (n)

sendo que θ (F(n)) quando crescimento de T(n) = crescimento de F(n)

# Counting Sort

c/ A[1..n] :: array de entrada B[1..n] :: array ordenado
C[1..k] array temporário c/ k o número de chaves diversas

Counting-Sort (A, B, k)

for i ← 0 to k
do C[i] ← 0;
for j ← 1 to length[A]
do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
do C[i] ← C[i] + C[i-1]
for j ← length[A] downto 1
do B[C[A[j]]] ← A[j]
C[A[j]] ← C[A[j]] - 1



# Radix Sort (Least Significant Digit - LSD)

Applet

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/radixSort/radixSort.html

# Counting Sort

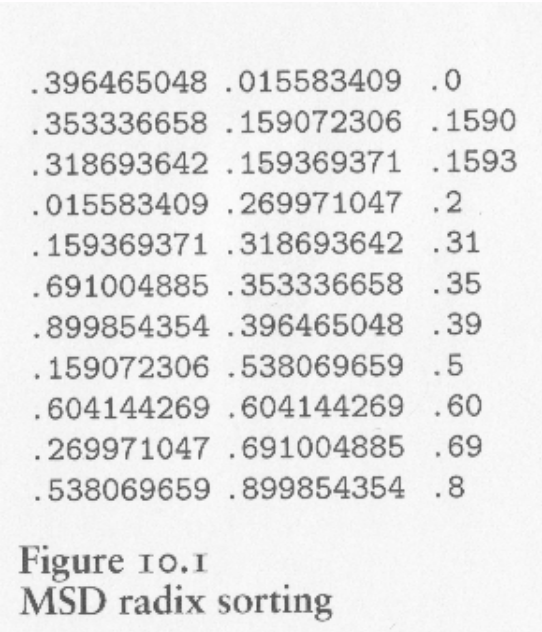
DE REALÇAR:

-Complexidade temporal do counting sort ?

- Este algoritmo é estável? Ou não?



# Radix Sort (Most Significant Digit - MSD)

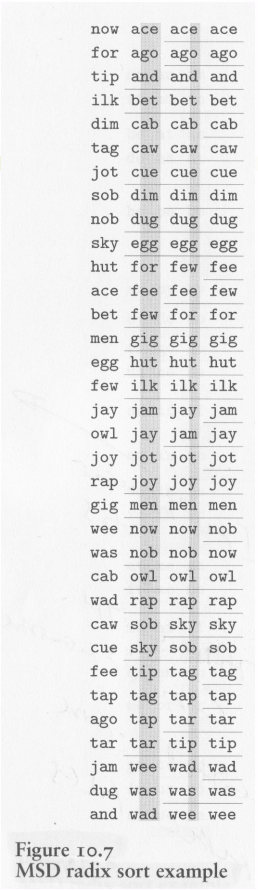


```
.396465048 .015583409  .0
.353336658 .159072306  .1590
.318693642 .159369371  .1593
.015583409 .269971047  .2
.159369371 .318693642  .31
.691004885 .353336658  .35
.899854354 .396465048  .39
.159072306 .538069659  .5
.604144269 .604144269  .60
.269971047 .691004885  .69
.538069659 .899854354  .8
```

Figure 10.1
MSD radix sorting

R = base
Qual a base neste caso?

# Radix Sort (MSD)

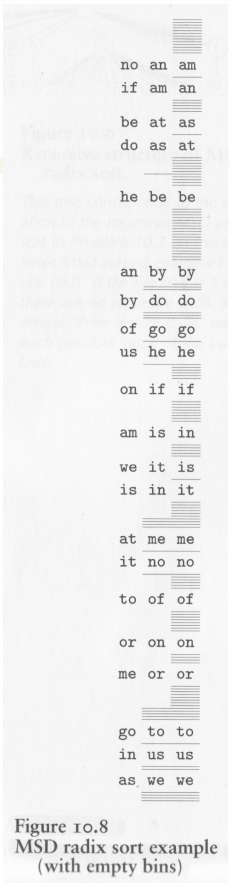


| | | | |
|---|---|---|---|
| now | ace | ace | ace |
| for | ago | ago | ago |
| tip | and | and | and |
| ilk | bet | bet | bet |
| dim | cab | cab | cab |
| tag | caw | caw | caw |
| jot | cue | cue | cue |
| sob | dim | dim | dim |
| nob | dug | dug | dug |
| sky | egg | egg | egg |
| hut | for | few | fee |
| ace | fee | fee | few |
| bet | few | for | for |
| men | gig | gig | gig |
| egg | hut | hut | hut |
| few | ilk | ilk | ilk |
| jay | jam | jay | jam |
| owl | jay | jam | jay |
| joy | jot | jot | jot |
| rap | joy | joy | joy |
| gig | men | men | men |
| wee | now | now | nob |
| was | nob | nob | now |
| cab | owl | owl | owl |
| wad | rap | rap | rap |
| caw | sob | sky | sky |
| cue | sky | sob | sob |
| fee | tip | tag | tag |
| tap | tag | tap | tap |
| ago | tap | tar | tar |
| tar | tar | tip | tip |
| jam | wee | wad | wad |
| dug | was | was | was |
| and | wad | wee | wee |

Figure 10.7
MSD radix sort example



# Radix Sort (MSD)

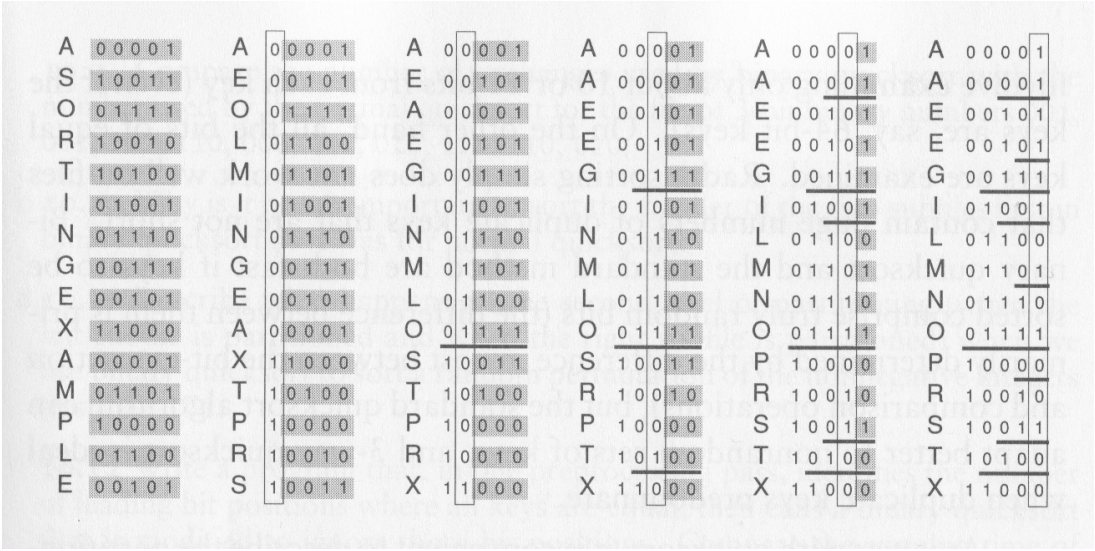


| | | |
|---|---|---|
| no | an | am |
| if | am | an |
| be | at | as |
| do | as | at |
| he | be | be |
| an | by | by |
| by | do | do |
| of | go | go |
| us | he | he |
| on | if | if |
| am | is | in |
| we | it | is |
| is | in | it |
| at | me | me |
| it | no | no |
| to | of | of |
| or | on | on |
| me | or | or |
| go | to | to |
| in | us | us |
| as | we | we |

Figure 10.8
MSD radix sort example
(with empty bins)

R = base
Qual a base neste caso?

# Radix Sort (MSD) c/ R = 2






# Radix Sort (MSD) c/ R = 2

MSD Radix c/ R=2 ➔ BINARY QUICK SORT

-Diferenças em relação ao QUICK SORT?
- chamada recursiva, argumentos?
- profundidade máxima da chamada recursiva?

- o que acontece com chaves iguais?

# Three-Way Radix Quicksort

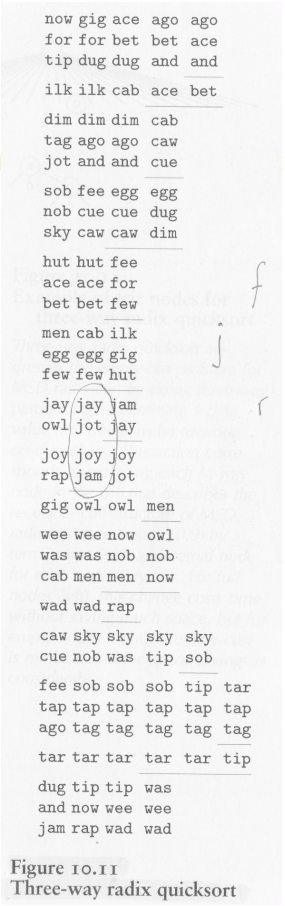


Figure 10.11
Three-way radix quicksort



# Counting Sort

Pressupostos:

- Cada elemento a ordenar é um inteiro
- Os valores são entre 0 e k c/ k um inteiro dado
-
- Para k = O(n) o ordenamento tem complexidade θ (n)

c/ θ (F(n)) crescimento de T(n) = crescimento de F(n)

# Counting Sort

c/ A[1..n] :: array de entrada B[1..n] :: array ordenado
C[1..k] array temporário c/ k o número de chaves diversas

Counting-Sort (A, B, k)

```
for i ← 0 to k
        do C[i] ← 0;
for j ← 1 to length[A]
        do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
        do C[i] ← C[i] + C[i-1]
for j ← length[A] downto 1
        do B[C[A[j]]] ← A[j]
           C[A[j]] ← C[A[j]] - 1
```



# Radix Sort (LSD)

Applet

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/radixSort/radixSort.html

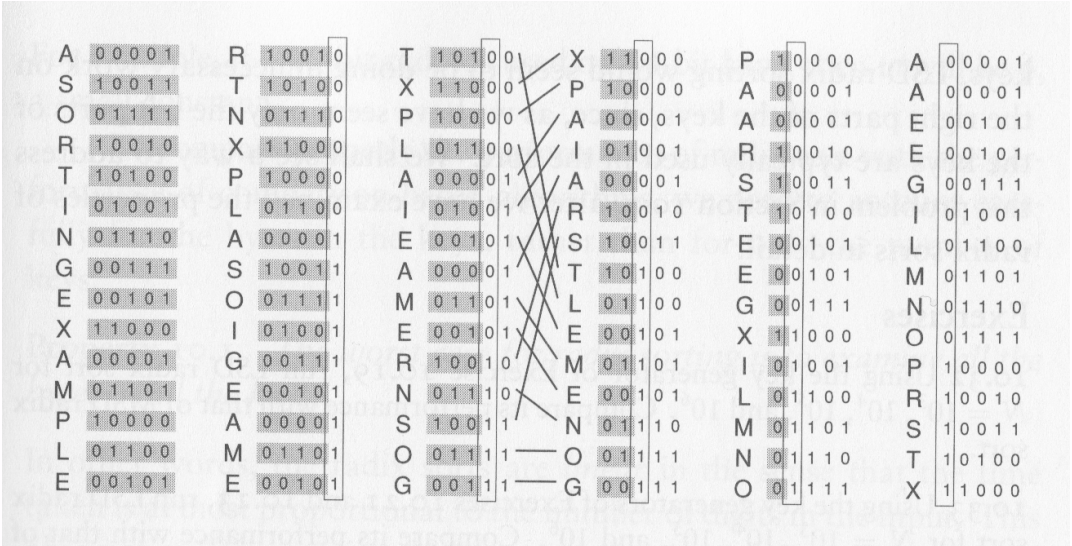
# Radix Sort (LSD)

| | | | |
|---|---|---|---|
| now | sob | cab | ace |
| for | nob | wad | ago |
| tip | cab | tag | and |
| ilk | wad | jam | bet |
| dim | and | rap | cab |
| tag | ace | tap | caw |
| jot | wee | tar | cue |
| sob | cue | was | dim |
| nob | fee | caw | dug |
| sky | tag | raw | egg |
| hut | egg | jay | fee |
| ace | gig | ace | few |
| bet | dug | wee | for |
| men | ilk | fee | gig |
| egg | owl | men | hut |
| few | dim | bet | ilk |
| jay | jam | few | jam |
| owl | men | egg | jay |
| joy | ago | ago | jot |
| rap | tip | gig | joy |
| gig | rap | dim | men |
| wee | tap | tip | nob |
| was | for | sky | now |
| cab | tar | ilk | owl |
| wad | was | and | rap |
| tap | jot | sob | raw |
| caw | hut | nob | sky |
| cue | bet | for | sob |
| fee | you | jot | tag |
| raw | now | you | tap |
| ago | few | now | tar |
| tar | caw | joy | tip |
| jam | raw | cue | wad |
| dug | sky | dug | was |
| you | jay | hut | wee |
| and | joy | owl | you |

Figure 10.14
LSD radix sort example



# Radix Sort (LSD)



# Radix Sort (LSD)

Applet

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/radixSort/radixSort.html



# Radix Sort (LSD)

Algum pressuposto importante para que o LSD funcione?

Que implicações tem o aumento do R?... no MSD (ex. Número de slots? Velocidade de processamento ?)

Que implicações tem o aumento do R?... No LSD (ex. Número de slots!?… algoritmo de counting? Velocidade de processamento ?)… e o caso particular de R=2 ou R $2^m$ ?

Que dizer sobre o trabalho de ordenamento sobre os radicais menos significativos?

…e sobre ter por exemplo um R >> N ?

Que dizer sobre o LSD aplicado a chaves que sejam cadeiras de caracteres?

Complexidade temporar dos varios algoritmos de Radix Sort? c/ R muito pequeno e.: R=2 ?.... Para que tende a complexidade? E R grande?...como fica o k ?... Para que tende a complexidade?

# Algoritmos de ordenamento

… end ;-)