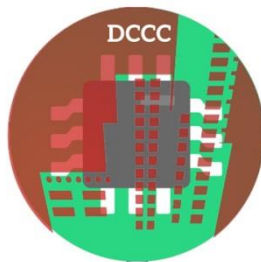


AEDII - Algoritmos e Estruturas de Dados II

Aula 03 – Limite assintótico para a ordenação, Ordenação em tempo linear



Departamento de Computação
de Construção Civil

Prof. Aléssio Miranda Júnior
alessio@timoteo.cefetmg.br
2Q-2019



15/08/2019

Ordenação

- Ordenar corresponde ao **processo de re-arranjar um conjunto de objetos** em ordem ascendente ou descendente.
- O objetivo principal da ordenação **é facilitar a recuperação posterior** de itens do conjunto ordenado.
- **Diversos algoritmos** de ordenação já foram estudados e implementados...

Ordenação

- Os métodos de ordenação são classificados em 2 grandes grupos:

Ordenação Interna:

Se o arquivo a ser ordenado cabe todo na memória principal

Ordenação Externa:

Se o arquivo a ser ordenado não cabe todo na memória principal



Ordenação

- Os métodos de ordenação são classificados em 2 grandes grupos:

Ordenação Interna:

Se o arquivo a ser ordenado cabe todo na memória principal

- **Algoritmos Baseados em Comparações**
- **Algoritmos Não Baseados em Comparações**

Ordenação baseada em comparações

Ordenação

➤ Algoritmos baseados em Comparações

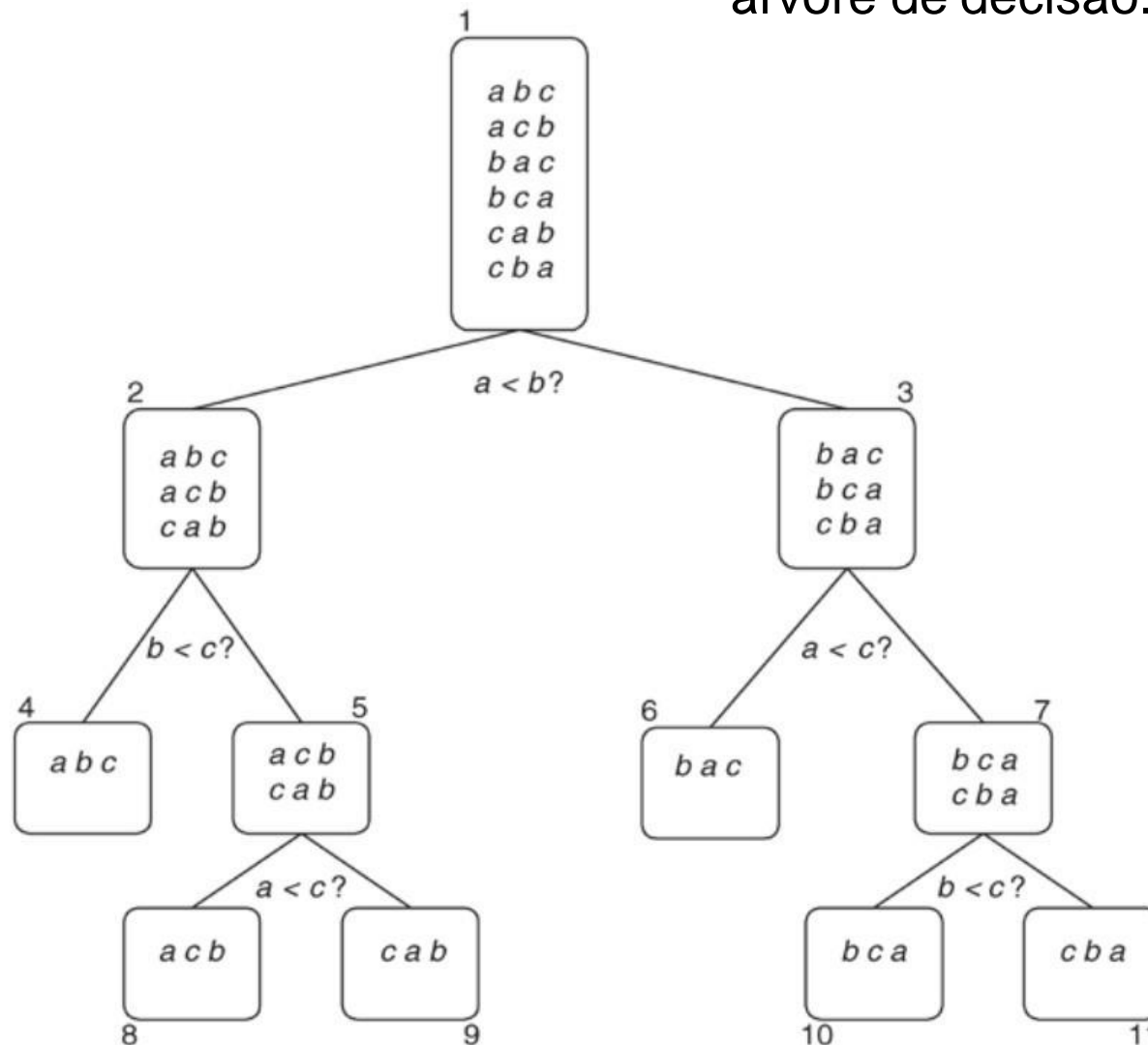
- Insertion sort
- Selection sort
- Bubble sort
- Merge sort
- Quick sort

➤ Complexidade computacional $\Omega(n \log(n))$

- [limite matemático]
- [limite assintótico para a ordenação]

[Árvore de decisão]

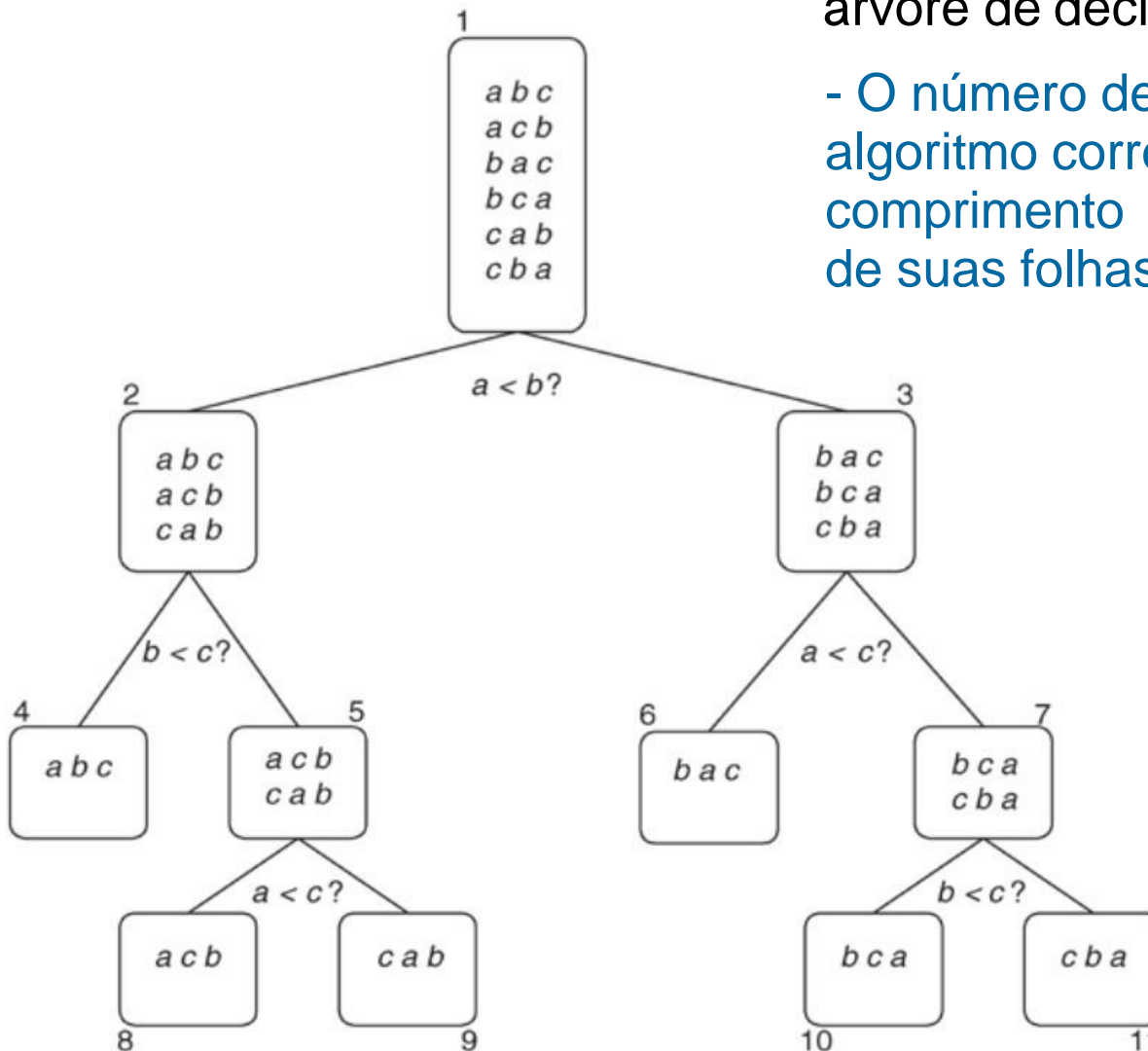
- Qualquer algoritmo de ordenação por comparação pode ser representado por uma árvore de decisão.



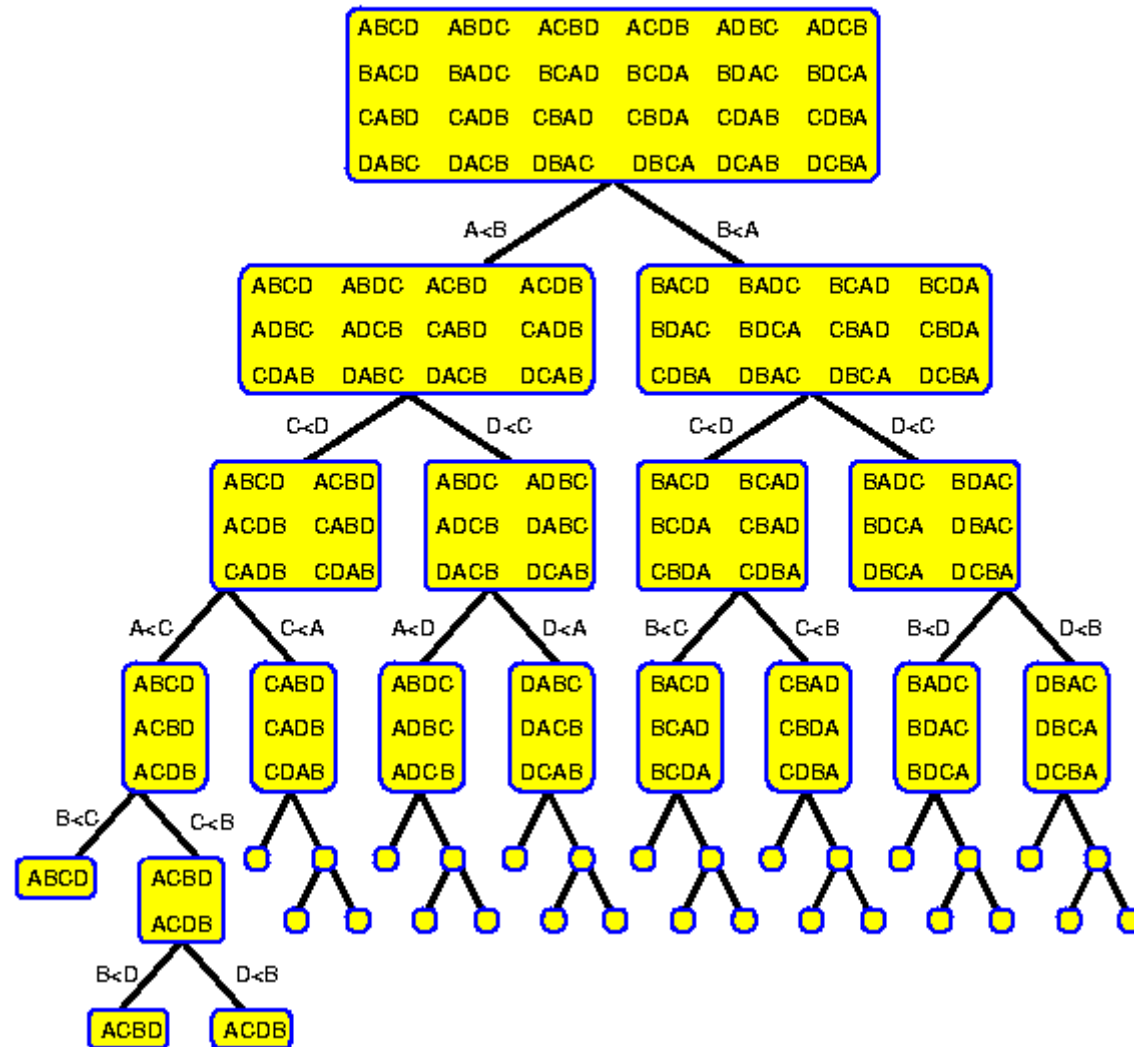
[Árvore de decisão]

- Qualquer algoritmo de ordenação por comparação pode ser representado por uma árvore de decisão.

- O número de comparações efetuadas pelo algoritmo corresponde ao maior comprimento do caminho da raiz até uma de suas folhas.



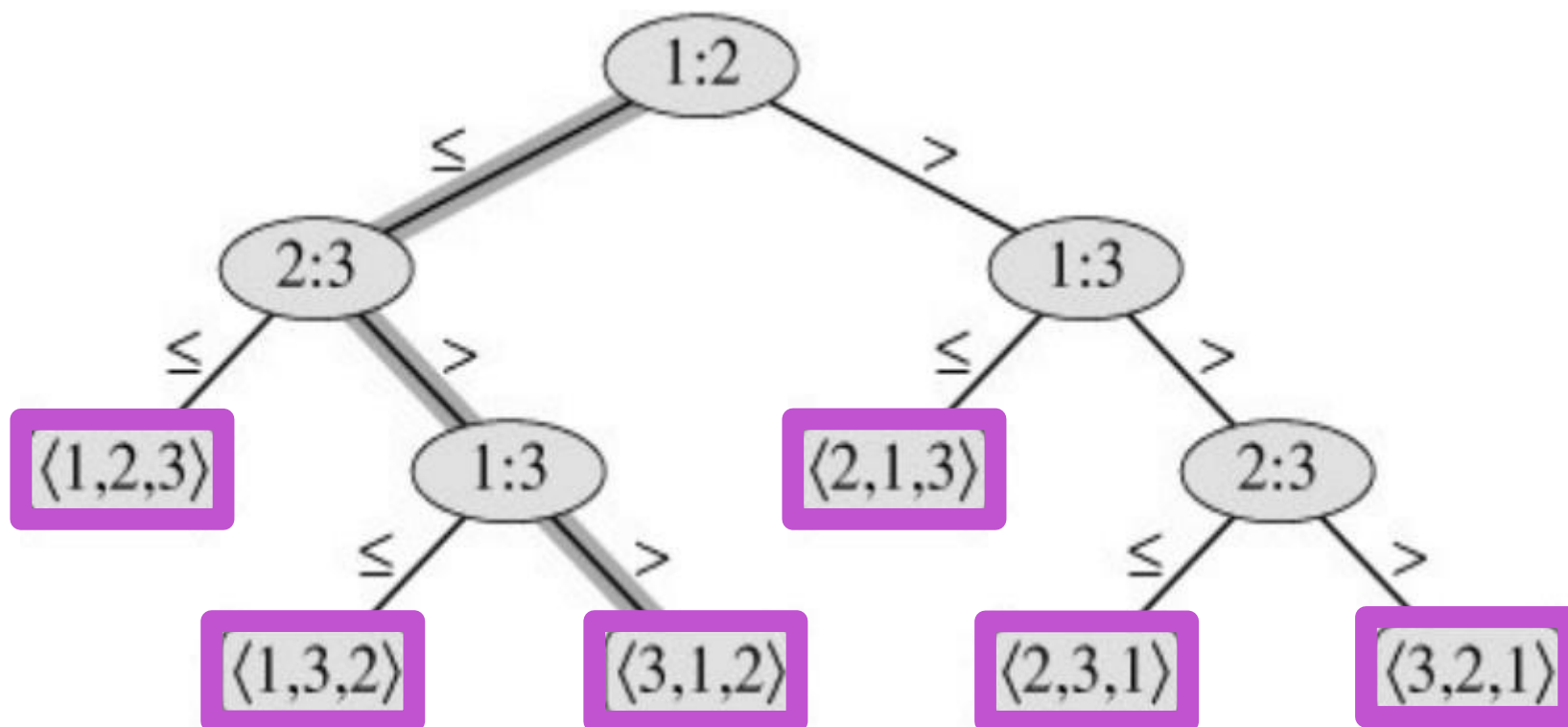
DECISION TREE



Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

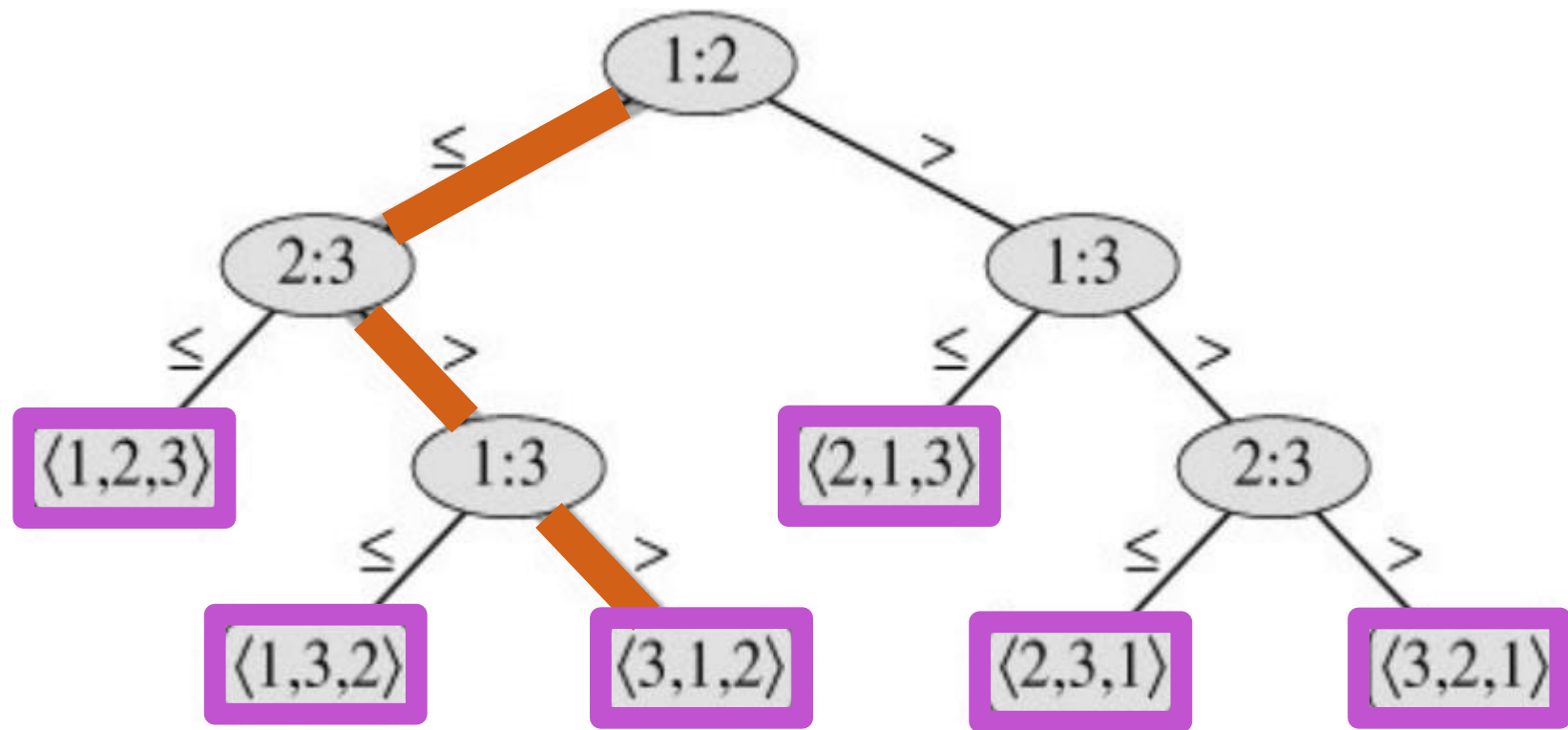


[Cada nó folha está associada a uma permutação dos elementos do vetor] 15/08/2019

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]



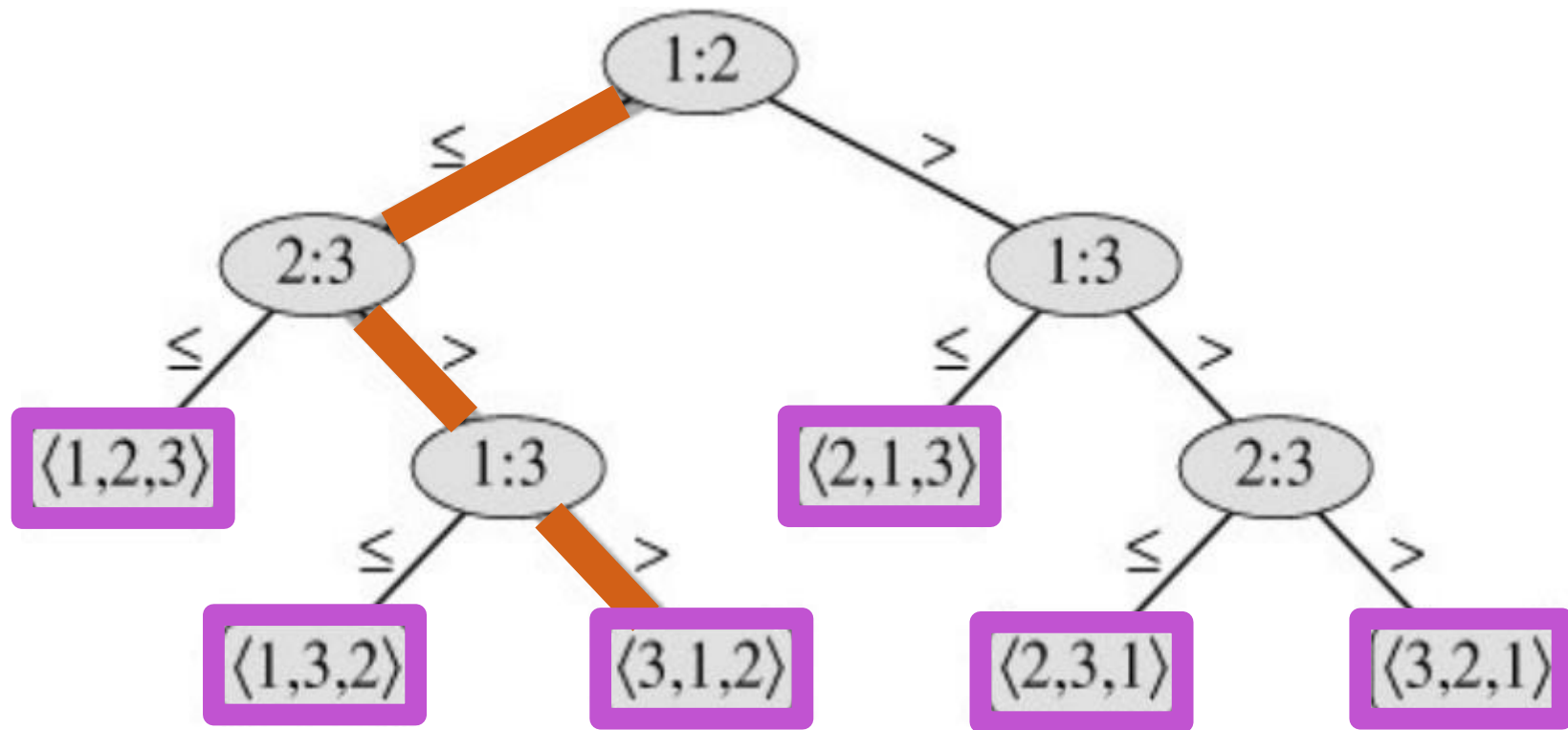
[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

Número de folhas = $n!$

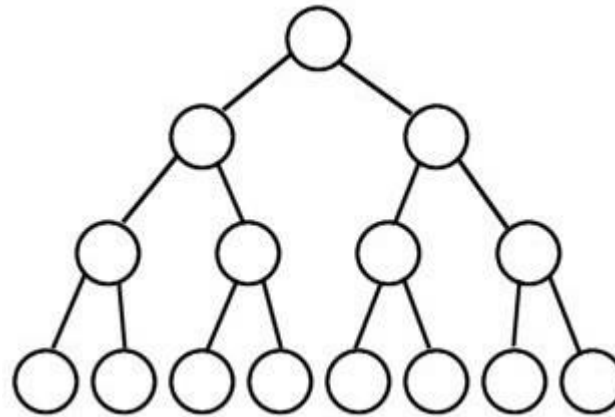


[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

Ordenação baseada em comparações

Seja L o número de folhas de uma árvore binária e h sua altura.

Então $L \leq 2^h$



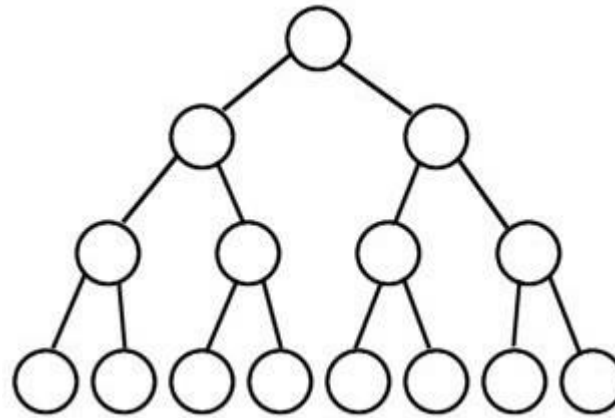
$h=3$

$L=8$

Ordenação baseada em comparações

Seja L o número de folhas de uma árvore binária e h sua altura.

Então $L \leq 2^h$



$h=3$

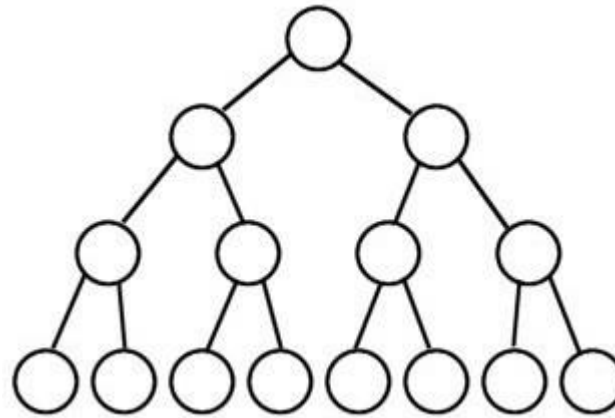
$L=8$

$$h \geq \log(n!)$$

Ordenação baseada em comparações

Seja L o número de folhas de uma árvore binária e h sua altura.

Então $L \leq 2^h$



$h=3$

$L=$
 8

$$h \geq \log(n!)$$

$$h = \Omega(n \log n)$$

Ordenação baseada em comparações

- **Algoritmos baseado em Comparações**
 - Insertion sort Selection sort
 - Bubble sort Merge sort
 - Quick sort

Vários algoritmos aqui listados são ótimos pois a sua complexidade computacional é $O(n \log n)$

Ordenação em tempo linear

Ordenação

➤ Algoritmos baseado em Comparações

- Insertion sort
- Selection sort
- Bubble sort
- Merge sort Quick sort

➤ Algoritmos não baseados em Comparações

➤ (utilizam alguma informação sobre os dados)

- Counting sort
- Radix sort
- Bin sort / Bucket sort

➤ Algoritmos que fazem a ordenação em tempo linear $O(n)$

Counting sort

- Os elementos a serem ordenados são números inteiros “pequenos”
- **Números inteiros todos menores ou iguais a k ,**
 $k \in O(n)$

Counting sort

- Os elementos a serem ordenados são números inteiros “pequenos”
- **Números inteiros todos menores ou iguais a k ,**
 $k \in O(n)$
- Counting sort ordena estes **n** números em tempo $O(n + k)$

Exemplo:

→ $n = 1\ 000\ 000$

→ $k = 30$

Counting sort

- O algoritmo usa dois vetores auxiliares para ordenar **A**:
- **C (de tamanho k)**, que guarda em **C[i]** o número de ocorrências de elementos i em **A**.
- **B (de tamanho n)**, onde se constrói o vetor ordenado.

Simulação:

<https://www.cs.usfca.edu/~galles/visualization/CountingSort.html>

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	1	1	3	3	1	2	1	2	2	0	1	1	1	1	1	0	1	1	0	1	2	0	1	0	0	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	1	3	3	1	2	1	2	2	0	1	1	1	1	1	0	1	1	0	1	2	0	1	0	0	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	3	3	1	2	1	2	2	0	1	1	1	1	1	0	1	1	0	1	2	0	1	0	0	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	17	17	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	17	17	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

																10													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	16	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	30	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

																10													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	17	17	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

																	10																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	17	17	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	29
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

																10													30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Counting sort

Esboce o algoritmo! (~7 min)

Pode usar vetores auxiliares:

- **C (de tamanho k)**, que guarda em **C[i]** o número de ocorrências de elementos i em **A**.
- **B (de tamanho n)**, onde se constrói o vetor ordenado.

Counting sort

```
int CountingSort (int *A, int N)
{
    int k = encontrarK(A,N);
    int B[N];
    int C[k+1];

    for(int i=0; i<=k; i++)
        C[i] = 0;

    for(int j=0; j<N; j++)
        C[A[j]] = C[A[j]] + 1;
    for(int i=1; i<=k; i++)
        C[i] = C[i] + C[i-1];

    for(int j=N-1; j>=0; j--)
    {
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

    for(int i=0; i<N; i++)
        printf("%d ",B[i]);
}
```

Counting sort

```

int encontrarK(int *A, int N)
{
    int max = A[0];
    for(int i=1; i<N; i++)
        if(max<A[i])
            max = A[i];
    return max;
}

```

```

int CountingSort (int *A, int N)
{
    int k = encontrarK(A,N);
    int B[N];
    int C[k+1];

    for(int i=0; i<=k; i++)
        C[i] = 0;

    for(int j=0; j<N; j++)
        C[A[j]] = C[A[j]] + 1;
    for(int i=1; i<=k; i++)
        C[i] = C[i] + C[i-1];

    for(int j=N-1; j>=0; j--)
    {
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

    for(int i=0; i<N; i++)
        printf("%d ",B[i]);
}

```

Counting sort

```

int encontrarK(int *A, int N)
{
    int max = A[0];
    for(int i=1; i<N; i++)
        if(max<A[i])
            max = A[i];
    return max;
}

```

```

int CountingSort (int *A, int N)
{
    int k = encontrarK(A,N);
    int B[N];
    int C[k+1];

```

```

    for(int i=0; i<=k; i++)
        C[i] = 0;

```

```

    for(int j=0; j<N; j++)
        C[A[j]] = C[A[j]] + 1;
    for(int i=1; i<=k; i++)
        C[i] = C[i] + C[i-1];

```

```

    for(int j=N-1; j>=0; j--)
    {
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

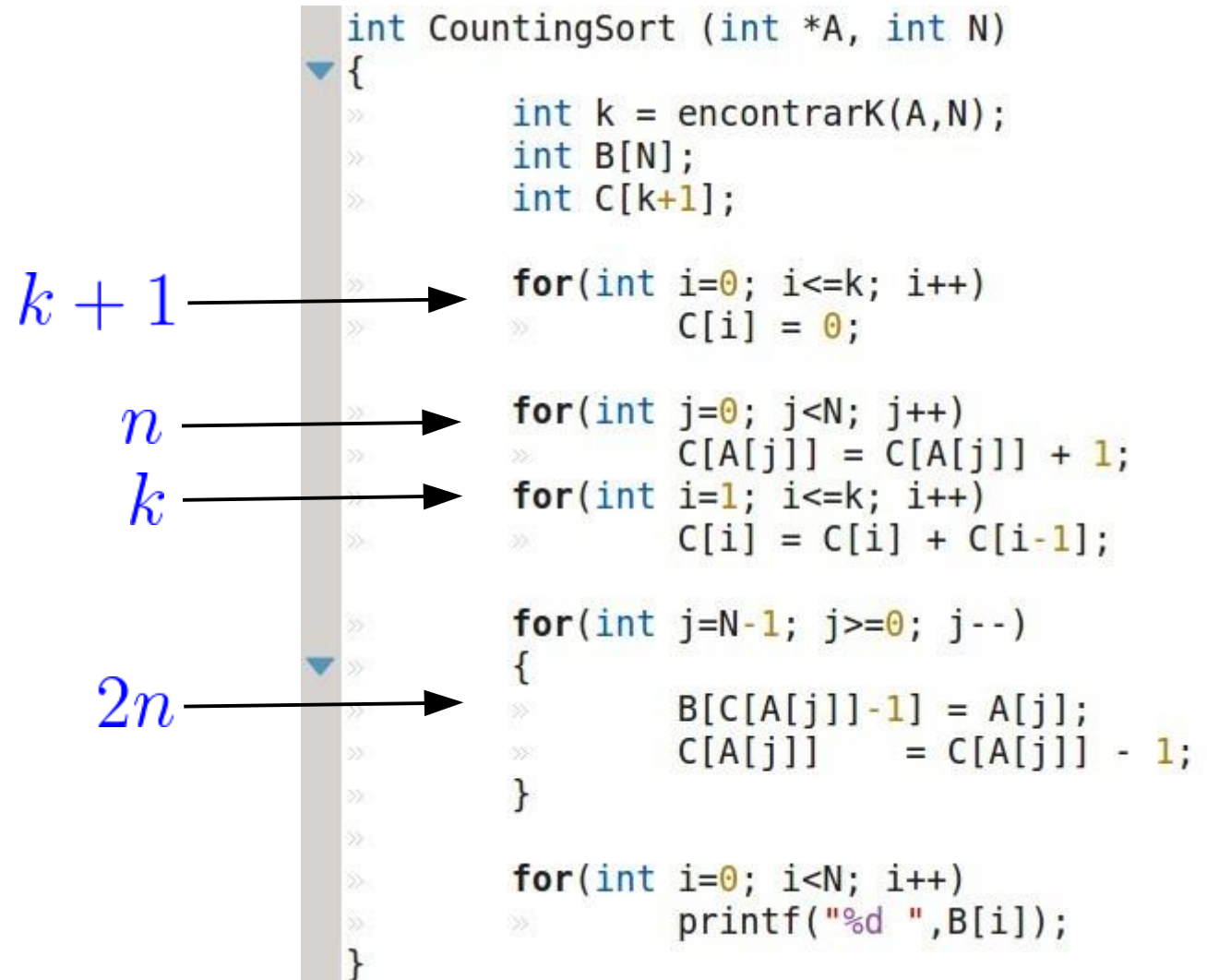
```

```

    for(int i=0; i<N; i++)
        printf("%d ", B[i]);
}

```

Counting sort



The diagram illustrates the Counting Sort algorithm with a vertical grey bar representing the execution flow. Blue annotations on the left indicate the number of register movements (M) for different parts of the code:

- $k + 1$: Points to the initialization of the counting array `C` from `i=0` to `i=k`.
- n : Points to the counting phase where `C[A[j]] = C[A[j]] + 1` is executed for all `j` from `0` to `N-1`.
- k : Points to the cumulative sum phase where `C[i] = C[i] + C[i-1]` is executed for `i` from `1` to `k`.
- $2n$: Points to the sorting phase where elements are placed into the output array `B` and the counting array `C` is decremented.

```

int CountingSort (int *A, int N)
{
    int k = encontrarK(A,N);
    int B[N];
    int C[k+1];

    for(int i=0; i<=k; i++)
        C[i] = 0;

    for(int j=0; j<N; j++)
        C[A[j]] = C[A[j]] + 1;

    for(int i=1; i<=k; i++)
        C[i] = C[i] + C[i-1];

    for(int j=N-1; j>=0; j--)
    {
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

    for(int i=0; i<N; i++)
        printf("%d ",B[i]);
}
  
```

$$M(n) = 3n + 2k + 1 = O(n + k)$$

M: número de movimentações de registros

Counting sort

```

int CountingSort (int *A, int N)
{
    int k = encontrarK(A,N);
    int B[N];
    int C[k+1];

    for(int i=0; i<=k; i++)
        C[i] = 0;

    for(int j=0; j<N; j++)
        C[A[j]] = C[A[j]] + 1;
    for(int i=1; i<=k; i++)
        C[i] = C[i] + C[i-1];

    for(int j=N-1; j>=0; j--)
    {
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

    for(int i=0; i<N; i++)
        printf("%d ",B[i]);
}

```

$$M(n) = 3n + 2k + 1 = O(n + k)$$

*M: número de
movimentações de
registros*

Se $k=n^2$, o algoritmo
teria custo linear?

01. Atividade em aula

Dado o seguinte vetor de 7 elementos e quatro sequências de índices do vetor que representam uma permutação. Selecione todas as sequências que pertencem ao resultado da execução de um algoritmo de ordenação estável.

0	1	2	3	4	5	6
40	30	10	40	10	30	40

(a) 4 2 5 1 6 3 0

(b) 2 4 1 5 0 3 6

(c) 0 3 6 1 5 2 4

(d) 6 3 0 5 1 4 2

(e) nenhuma sequência de índices pertence a um algoritmo de ordenação estável

01. Atividade em aula

Um algoritmo de ordenação é estável quando números com o mesmo valor aparecem no arranjo de saída na mesma ordem em que se encontram no arranjo

7 5 2 5

2 5 5 7 ← Estável

2 5 5 7 ← Não-estável

01. Atividade em aula

Dado o seguinte vetor de 7 elementos e quatro sequências de índices do vetor que representam uma permutação. Selecione todas as sequências que pertencem ao resultado da execução de um algoritmo de ordenação estável.

0	1	2	3	4	5	6
40	30	10	40	10	30	40

(a) 4 2 5 1 6 3 0

(b) 2 4 1 5 0 3 6

(c) 0 3 6 1 5 2 4

(d) 6 3 0 5 1 4 2

(e) nenhuma sequência de índices pertence a um algoritmo de ordenação estável

02. Atividade em aula

```

void CountinhoSort (int v[], int n) {
    int i, j, count[n], s[n];

    for (i=0; i<n; i++)
        count[i] = 0;

    for (i=0; i<n-1; i++) {
        for (j=i+1; j<n; j++) {
            if (v[i]<v[j])
                count[j] += 1;
            else
                count[i] += 1;
        }
    }

    for (i=0; i<n; i++)
        s[count[i]] = v[i];

    for (i=0; i<n; i++)
        v[i] = s[i];
}

```

v :=

62 31 84 96 19 47

count :=

0 0 0 0 0 0

i=0

3 0 1 1 0 0

O algoritmo é estável?

02. Atividade em aula

```

void CountinhoSort (int v[], int n) {
    int i, j, count[n], s[n];

    for (i=0; i<n; i++)
        count[i] = 0;

    for (i=0; i<n-1; i++) {
        for (j=i+1; j<n; j++) {
            if (v[i]<v[j])
                count[j] += 1;
            else
                count[i] += 1;
        }
    }

    for (i=0; i<n; i++)
        s[count[i]] = v[i];

    for (i=0; i<n; i++)
        v[i] = s[i];
}

```

v :=

62	31	84	96	19	47
----	----	----	----	----	----

count :=

0	0	0	0	0	0
---	---	---	---	---	---

i=0

3	0	1	1	0	0
---	---	---	---	---	---

i=1

3	1	2	2	0	1
---	---	---	---	---	---

i=2

3	1	4	3	0	1
---	---	---	---	---	---

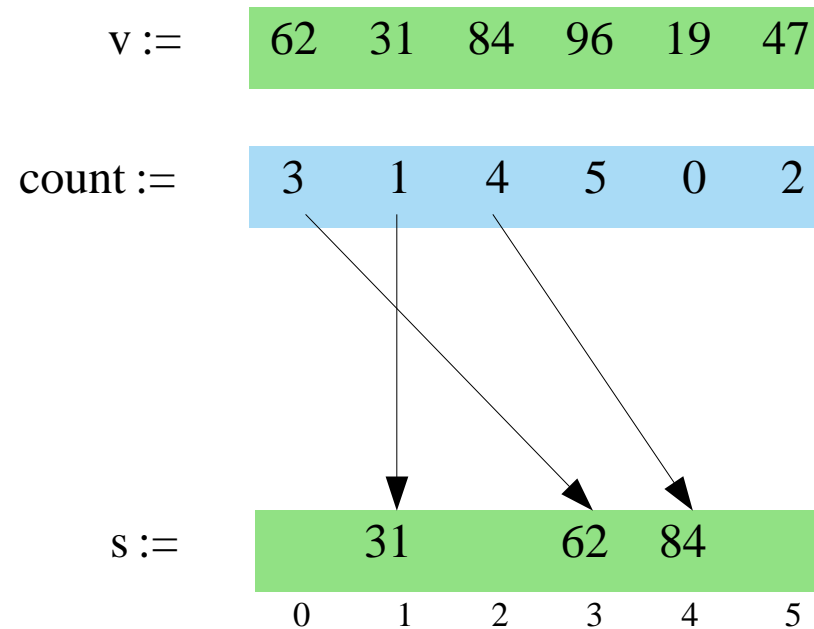
i=3

3	1	4	5	0	1
---	---	---	---	---	---

i=4

3	1	4	5	0	2
---	---	---	---	---	---

02. Atividade em aula



02. Atividade em aula

```
void CountinhoSort (int v[], int n) {  
    int i, j, count[n], s[n];  
  
    for (i=0; i<n; i++)  
        count[i] = 0;  
  
    for (i=0; i<n-1; i++) {  
        for (j=i+1; j<n; j++) {  
            if (v[i]<v[j])  
                count[j] += 1;  
            else  
                count[i] += 1;  
        }  
    }  
  
    for (i=0; i<n; i++)  
        s[count[i]] = v[i];  
  
    for (i=0; i<n; i++)  
        v[i] = s[i];  
}
```

(a) O algoritmo ordena através da contagem de elementos menores e maiores de cada elemento do vetor v.

→ Afirmação correta

(b) O algoritmo é $O(n^2)$ pois são realizadas n^2 comparações (2 laços aninhados)

→ Afirmação incorreta

Counting sort

➤ O Counting sort é um algoritmo que faz a ordenação de forma **estável**:

A posição relativa de elementos iguais que ocorrem no vetor ordenado permanecem na mesma ordem em que aparecem na entrada.

- Counting sort e Quick sort são estáveis
- Heap sort é não-estável.

Counting sort

A

13	2	18	9	3	1	6	12	7	9	19	28	5	14	4	16	15	10	7	22	21	8	5	5	4	4	24	22	30	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

C

0	1	2	3	6	9	10	12	13	15	17	17	18	19	20	21	22	22	23	24	24	25	27	27	28	28	28	28	29	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Radix sort

Ordena o vetor **A** de ***n*** números inteiros com um número constante **d** de dígitos

A ordem começa pelo dígito menos significativo.

RadixSort (A, d)

1. para $i := 1$ até d faça
2. ordene os elementos de A pelo i -ésimo dígito usando um método **estável**

Simulação: <https://www.cs.usfca.edu/~galles/visualization/RadixSort.html>

Radix sort

RADIX SORT

Initial situation

89	28	81	69	14	31	29	18	39	17
----	----	----	----	----	----	----	----	----	----

After sorting on second digit

81	31	14	17	28	18	89	69	29	39
----	----	----	----	----	----	----	----	----	----

After sorting on first digit

14	17	18	28	29	31	39	69	81	89
----	----	----	----	----	----	----	----	----	----

Fonte: <http://users.informatik.uni-halle.de/~jopsi/dinf205/chap8.shtml>

15/08/2019

Radix sort

A complexidade computacional, considerando o número de movimentações de registros, e o counting sort:

$$M(n) = O(d(n + k))$$

- **Counting sort** = $O(n + k)$ ← Vantajoso quando $d < \log(n)$
- **Merge sort** = $O(n \log n)$

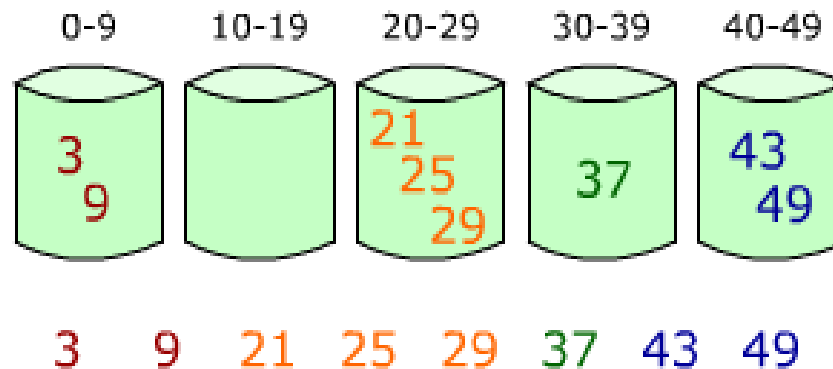
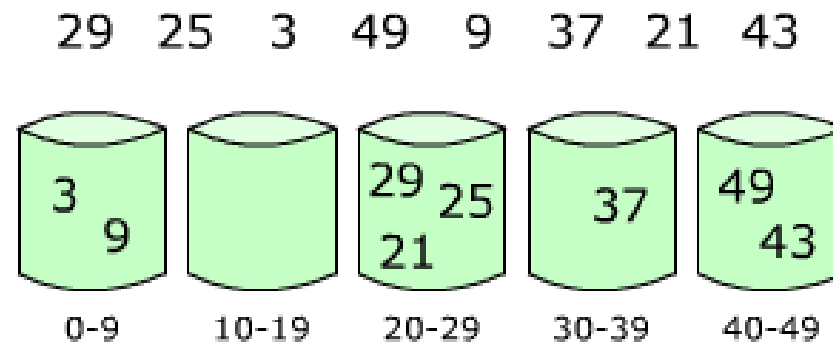
Bucket sort (Ordenação por balde)

Possui tempo linear, desde que os valores a serem ordenados sejam distribuídos uniformemente sobre o intervalo $[0, 1)$.

Divide o intervalo $[0, 1)$ em n sub-intervalos iguais, denominados buckets (baldes), e então distribui os n números reais nos n buckets.

Como a entrada é composta por dados distribuídos uniformemente, espera-se que cada balde possua, ao final deste processo, um número equivalente de elementos (usualmente 1).

Bucket sort (Ordenação por balde)



Bucket sort (Ordenação por balde)

Cada elemento $A[i]$ satisfaz $0 \leq A[i] < 1$.

Para obter o resultado, basta ordenar os elementos em cada bucket e então apresentá-los em ordem

BucketSort(A)

1. $n :=$ comprimento de A
2. **para** $i := 1$ **até** n **faça**
3. insira $A[i]$ na lista ligada $B[\lfloor nA[i] \rfloor]$
4. **para** $i := 0$ **até** $n - 1$ **faça**
5. ordene a lista $B[i]$ com *Insertion Sort*
6. Concatene as listas $B[0], B[1], \dots, B[n - 1]$ nessa ordem.

Desafio 02 – opcional – Envio até --/08 (23h50-maratona)

- Implemente o algoritmo de ordenação RadixSort.
- Seu programa não deve impor limitações sobre o número de elementos (n), nem o número de dígitos (d), nem o valor de k (os números podem estar em qualquer base, por exemplo, octal, hexadecimal, etc.).
- Apresentação livre de exemplos (quanto mais completo melhor).
- Apenas 2 arquivos que deverá submeter pelo Maratona: Código fonte em C/C++/Java
- Um PDF contendo uma simples descrição do programa (não maior a 4 páginas). O formato desse relatório é livre.