INSTITUTO FEDERAL
Ceará
Campus Tianguá

MÉTODOS DE ORDENAÇÃO PELA FORÇA BRUTA

Ciência da Computação

Disciplina: Construção e Análise de Algoritmos

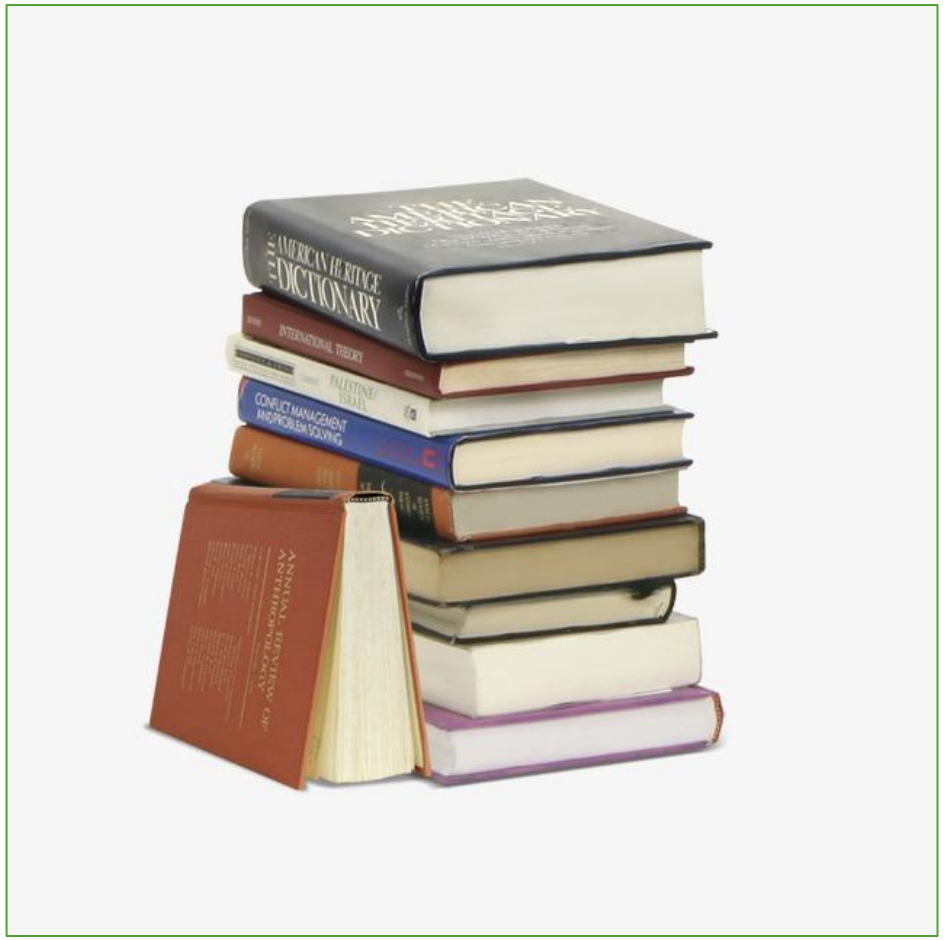
Professor: Adonias Caetano

Objetivos

- Compreender os conceitos sobre força bruta e ordenação;
- Compreender o método de ordenação do Bubble, Insertion e Selection sort.
- Apresentar implementações desses métodos em C.

Conteúdo da aula

- Força Bruta
- Ordenação
- Bubble Sort
- Insertion Sort
- Selection Sort



ifce.edu.br

FORÇA BRUTA

Definição

- É a mais simples das estratégias de projeto
- Pode ser definida como:

Uma solução direta para resolver um problema, geralmente baseada diretamente no enunciado do problema e nas definições dos conceitos envolvidos

Características

- A “força” é de um computador e não do intelecto de alguém.
- “Somente faça”
- Geralmente, a estratégia de “**força bruta**” é uma das mais fáceis de aplicar.

Exemplos

- Calculando a soma de $\boldsymbol{n}$ números ($\boldsymbol{a} > \mathbf{0}$, n sendo um inteiro não negativo)
- Calculando $\boldsymbol{n}!$
- Multiplicação de duas matrizes $\boldsymbol{n}$ por $\boldsymbol{n}$
- Selection Sort
- Busca sequencial
- Encontrar o maior elemento numa lista

Por que aplicar força bruta?

- Para alguns problemas importantes (ordenação, busca, multiplicação de matrizes, buscas, etc), a técnica de força bruta fornece:

- algoritmos razoáveis
- algoritmos de valor prático
- algoritmos sem limitações quanto ao tamanho da instância.
- aplicável a uma ampla variedade de problemas

Quando aplicar força bruta?

- Poucas instâncias de um problema precisam ser resolvidas (velocidade aceitável);
- Geralmente, o problema possui instâncias pequenas;
- Propósito teórico e educacional.

ORDENAÇÃO

Definição

- Uma das aplicações mais estudadas e realizadas sobre vetores é a **ordenação**.
- Ordenar um vetor significa permutar seus elementos de tal forma que eles fiquem em ordem crescente, ou seja,

$$v[0] \leq v[1] \leq v[2] \leq \cdots \leq v[n-1].$$

Exemplo

- Por exemplo, suponha o vetor.

v	5	6	−9	9	0	4

- Uma ordenação desse vetor resultaria em um rearranjo de seus elementos:

v	−9	0	4	5	6	9

Ordenação estável

- Um **algoritmo de ordenação** diz-se **estável** se preserva a ordem de registros de chaves iguais.
- Isto é, se tais registros aparecem na sequência ordenada na mesma ordem em que estão na sequência inicial.
- Esta propriedade é útil apenas quando há dados associados às chaves de ordenação.

Exemplo

- Por exemplo, suponha o seguinte vetor inicial.

v	5	6	9	9	6	4

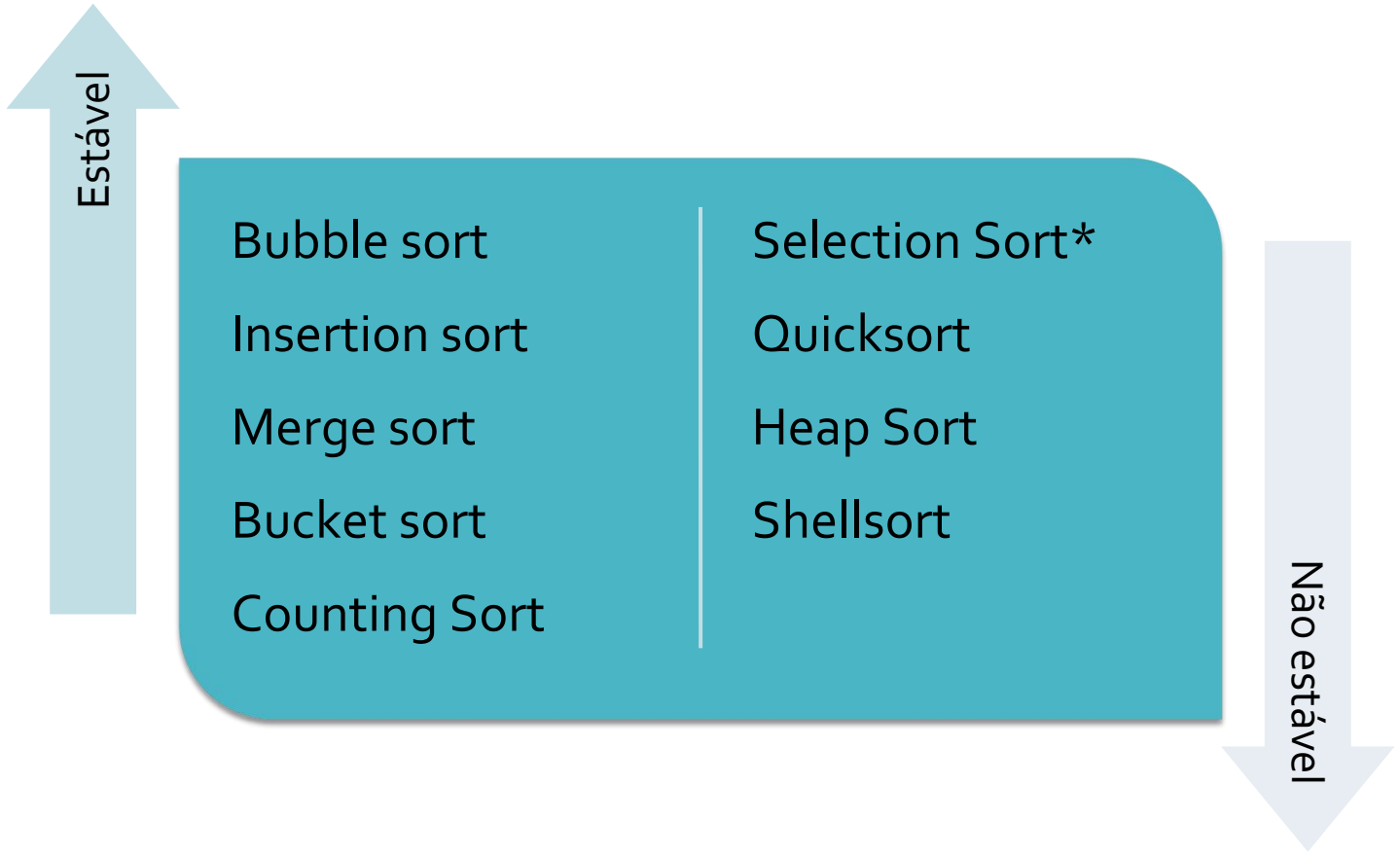
- Um algoritmo de **ordenação estável** devolve o vetor como:

v	4	5	6	6	9	9

- Um algoritmo de **ordenação não estável** devolve o vetor como:

v	4	5	6	6	9	9

Exemplos de algoritmos

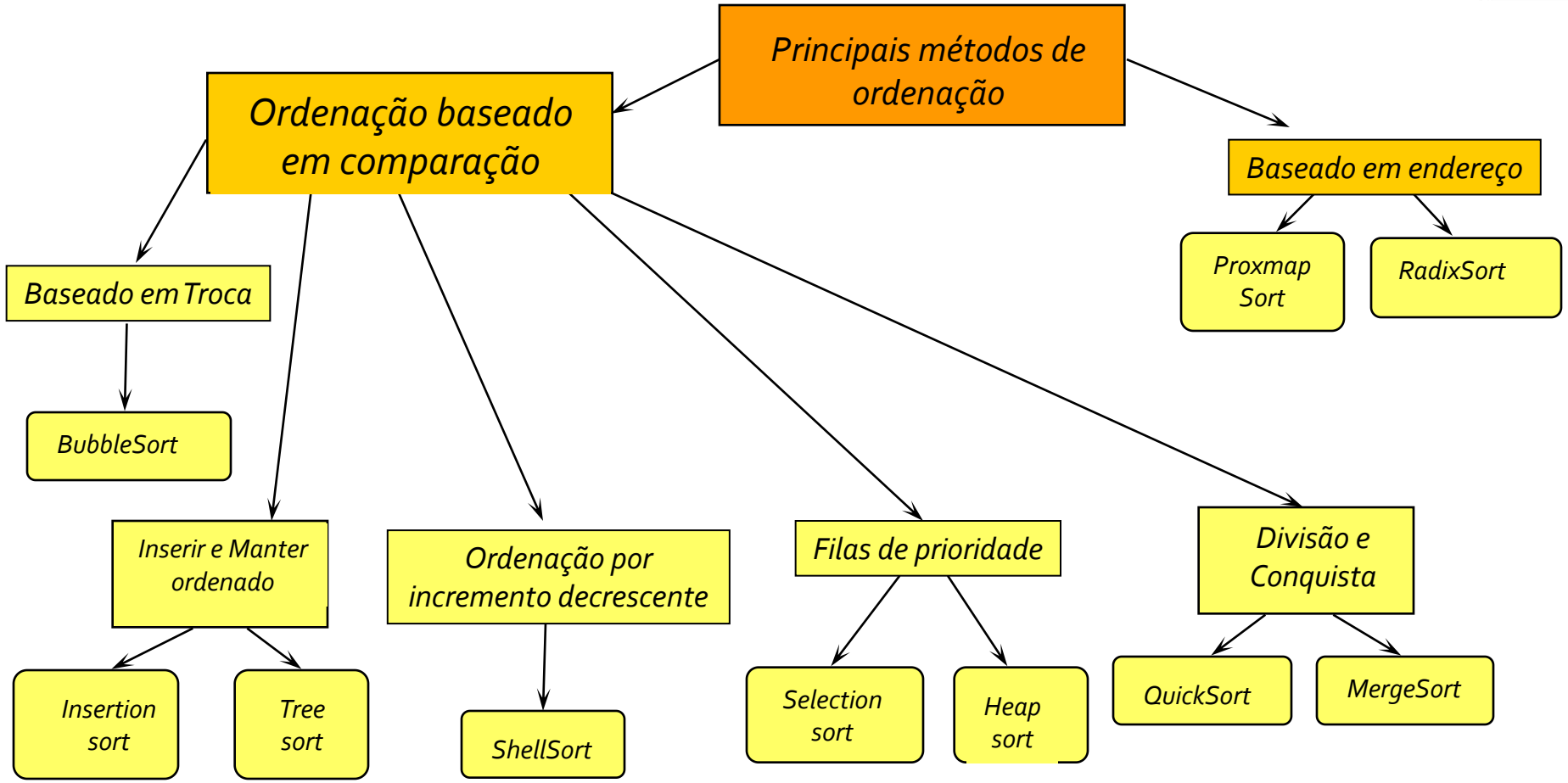


* Depende do algoritmo.

Tipos de ordenação

- Existem diversos algoritmos de ordenação para vetores.
- Eles variam em relação à dificuldade de implementação e desempenho.
- Usualmente algoritmos mais fáceis de serem implementados apresentam desempenho inferior.

Tipos de ordenação



Tipos de ordenação

ifce.edu.br

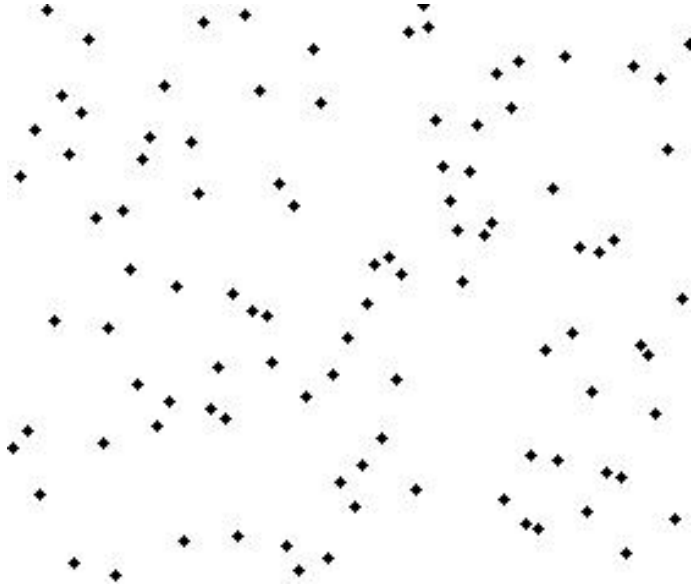
Algoritmo	Comparações			Movimentações			Espaço	Estável	In situ
	Melhor	Médio	Pior	Melhor	Médio	Pior			
Bubble	$O(n^2)$			$O(n^2)$			$O(1)$	**Sim**	**Sim**
Selection	$O(n^2)$			$O(n)$			$O(1)$	**Não***	**Sim**
Insertion	$O(n)$	$O(n^2)$		$O(n)$	$O(n^2)$		$O(1)$	**Sim**	**Sim**
Merge	$O(n\ log\ n)$			–			$O(n)$	**Sim**	**Não**
Quick	$O(n\ log\ n)$		$O(n^2)$	–			$O(n)$	**Não***	**Sim**
Shell	$O(n^{1.25})$ ou $O(n\ (ln\ n)^2)$			–			$O(1)$	**Não**	**Sim**

* Existem versões estáveis.

BUBBLE SORT

Definição

- É um algoritmo simples, útil para ordenação de vetores pequenos (desempenho ruim).
- Conhecido também como algoritmo da bolha ou intercalação ou troca.



- Sua movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

Ideia Básica

INÍCIO

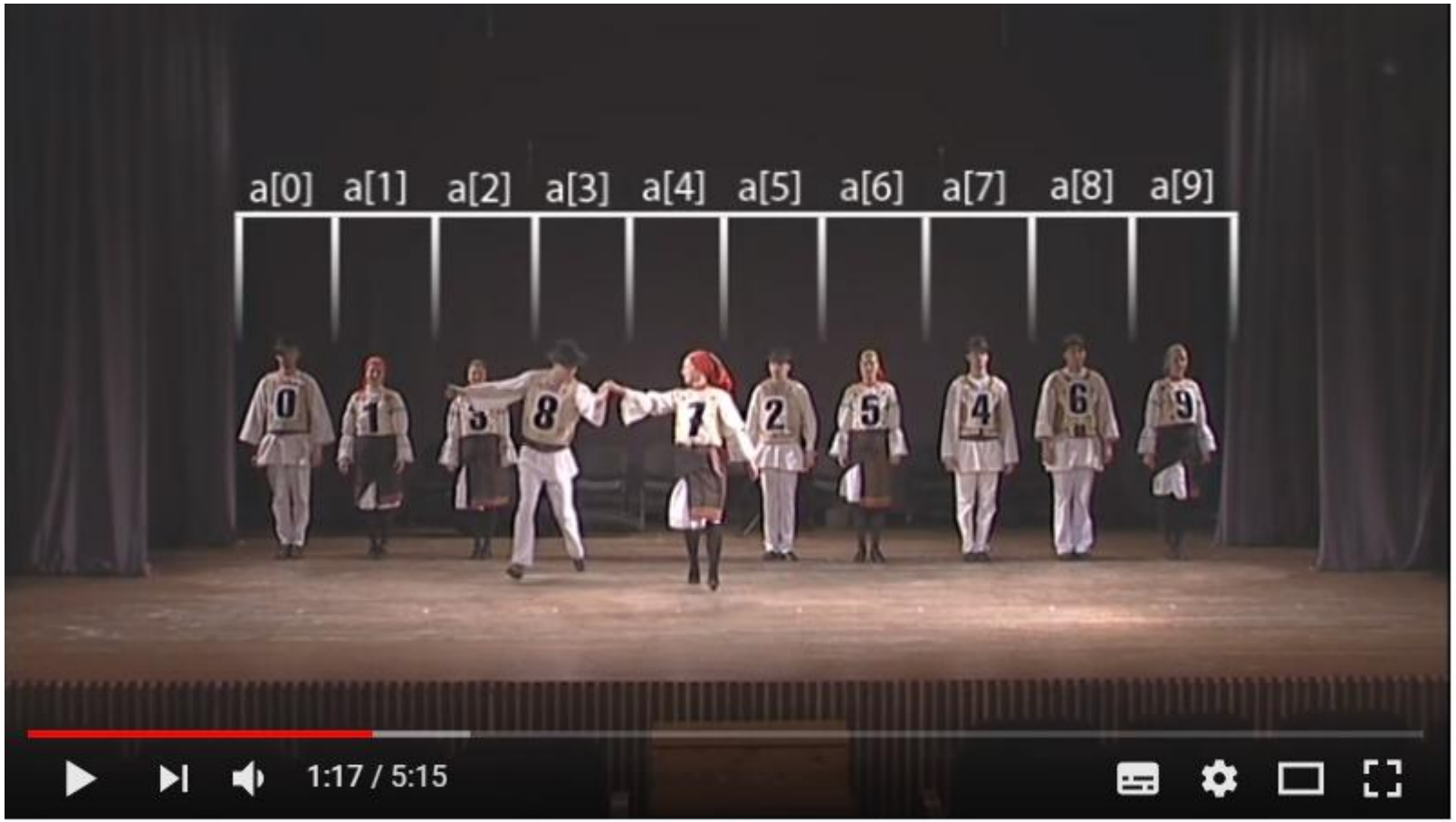
Repita até que o penúltimo elemento seja comparado com o último.

- Compare o 1º elemento com o 2º. Se estiverem desordenados, então efetue a troca de posição.
- Compare o 2º elemento com o 3º e efetue a troca de posição, se necessário;

Ao final desta repetição o elemento de maior valor estará em sua posição correta, a n-ésima posição do vetor;

- Continue a ordenação posicionando o segundo maior elemento, o terceiro,..., até que todo o vetor esteja ordenado;
- Em cada iteração "empurra" o maior elemento para última posição;

AlgoRythmics



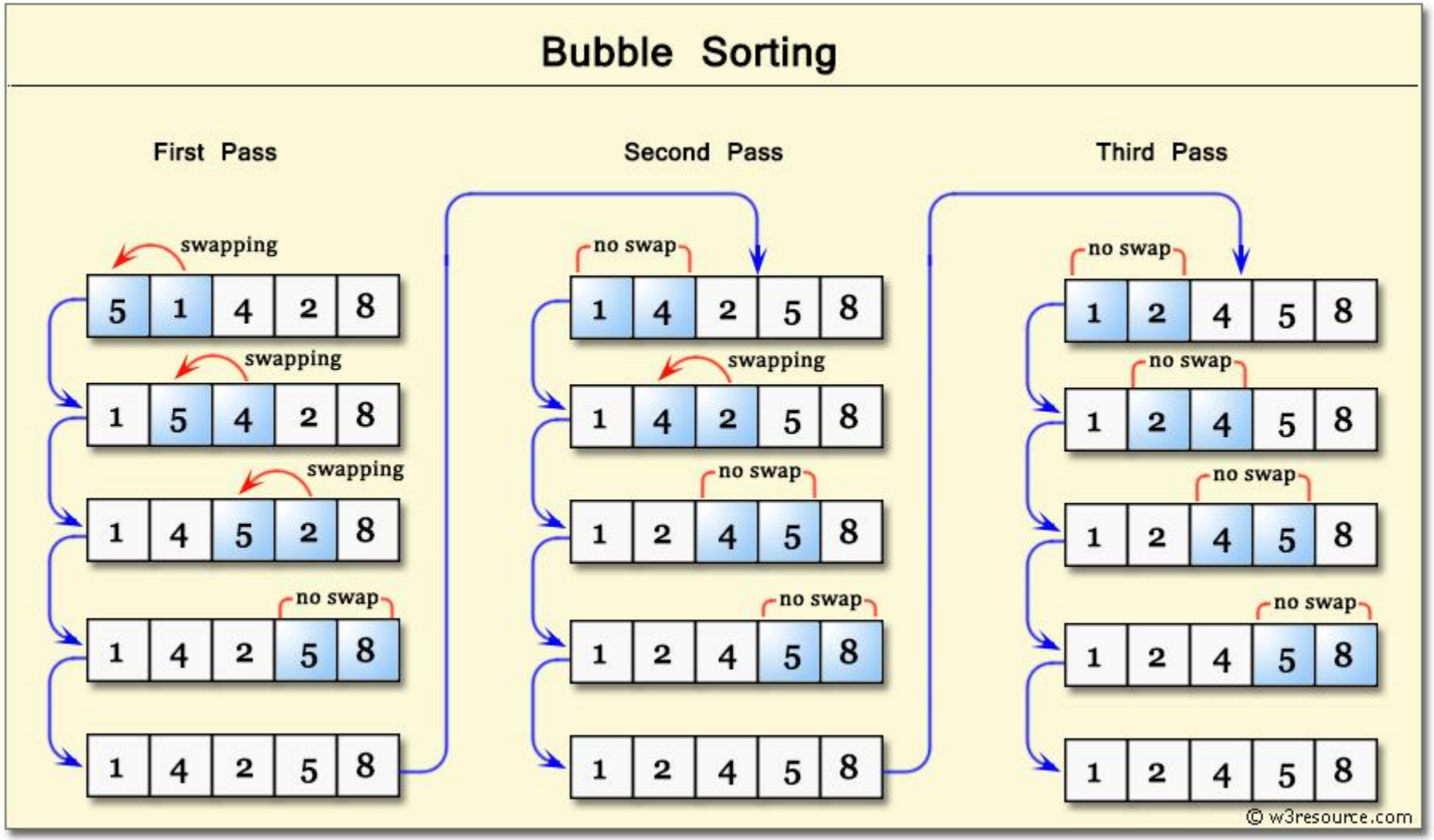
Bubble-sort with Hungarian ("Csángó") folk dance

https://youtu.be/lyZQPjUT5B4

Animação

6 5 3 1 8 7 2 4

Ilustração


Bubble Sorting
First Pass
Second Pass
Third Pass
swapping
no swap
5 1 4 2 8
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
1 4 2 5 8
1 4 2 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
© w3resource.com

Pseudocódigo

- PSEUDOCÓDIGO: suponha um vetor v de tamanho n.

```
DECLARE i, j, aux, n, v[n] NUMÉRICO;
PARA i = n-1 ATÉ i > 0 FAÇA
INÍCIO
  PARA j = 0 ATÉ j < i FAÇA
  INÍCIO
    SE v[j]> v[j+1]
      aux = v[j]; v[j] = v[j+1]; v[j+1] = aux;
  FIM
FIM
```

Exercício de Fixação

- Implemente na linguagem C o algoritmo de ordenação bubble sort. Utilize uma função auxiliar para implementar a ordenação.

Solução

```c
#include <stdlib.h>

void bubbleSort(int v[], int n)
{
    int i, j, aux;
    for(i = n-1; i > 0; i--) {
        for(j = 0; j < i; j++) {
            if(v[j] > v[j+1]) {
                aux = v[j]; v[j] = v[j+1]; v[j+1] = aux; //troca
            }
        }
    }
}

int main()
{
    int v[] = {5, 6, -9, 9, 0, 4 };
    int n = 6, i;

    bubbleSort(v, n);

    printf("\n\nVetor ordenado:\n");
    for(i = 0; i < n; i++)
        printf("%d\t", v[i]);
    printf("\n");
    return 0;
}
```

ifce.edu.br

Análise de Complexidade

- No **Pior Caso**, as operações de comparações e de troca de posição de elementos são executadas mais vezes.
- O algoritmo realizará $n - 1$ troca para a primeira iteração, depois $n - 2$ trocas para o segundo elemento e assim sucessivamente.
- $\boldsymbol{Trocas} = n - 1 + n - 2 + n - 3 \ldots + 2 + 1$ aproximadamente $\boldsymbol{n^2}$ trocas.
- No **Melhor Caso**, nenhuma troca será realizada, pois em ambos os casos o algoritmo faz da ordem $\boldsymbol{n}$ comparações.

Análise de Complexidade

- **Complexidade no tempo:** Comportamento do algoritmo no tempo, em função do tamanho da entrada.
- **Complexidade no espaço:** Consumo de memória do algoritmo, em função do tamanho da entrada.
- O tempo gasto na execução do algoritmo varia em ordem quadrática em relação ao número de elementos a serem ordenados.
- $T = O(n^2)$ **Notação "Big O"**

Análise de Complexidade

- **Atividades mais custosas:**
  - Comparações
  - Troca de Posição (swap)

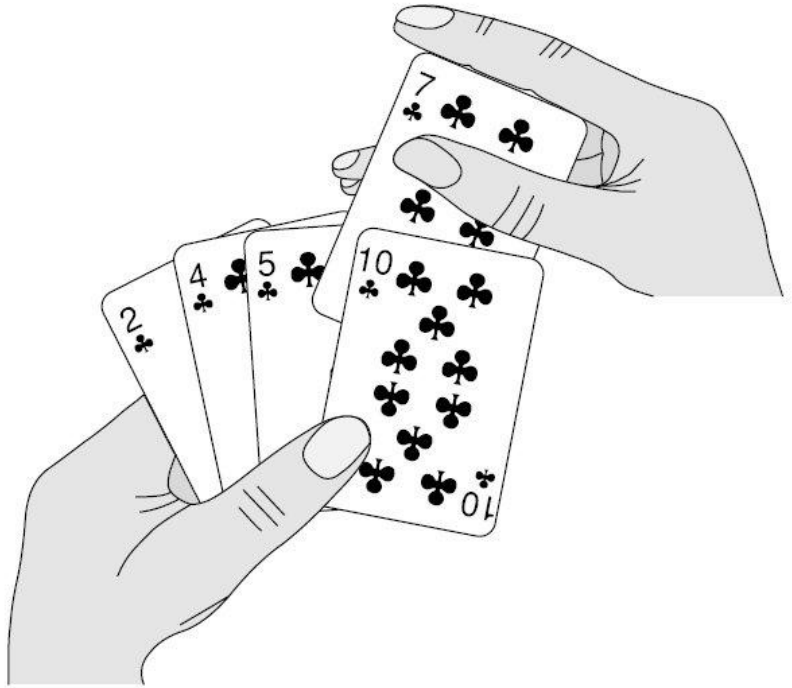
- **Melhor caso:** Vetor Ordenado.
- **Pior caso:** Vetor invertido

INSERTION SORT

Definição

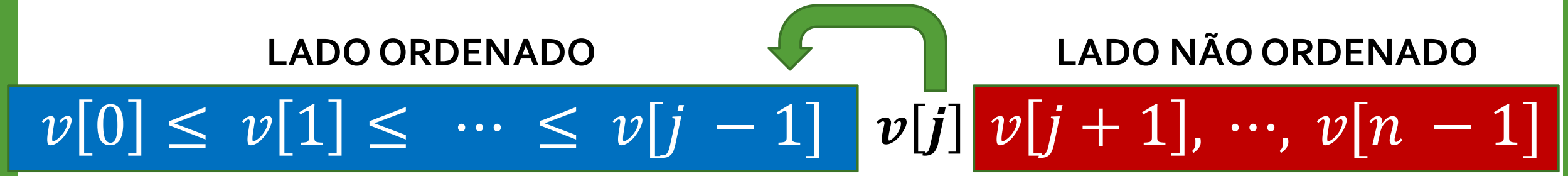
- Também conhecida como **Ordenação por Inserção, esse métodos** consiste em inserir um elemento n num vetor já ordenado de $n - 1$ elementos.

- Método de ordenação semelhante ao que usamos para ordenar as cartas de um baralho.



Ideia Básica

- Compare $v[j]$ com os elementos à sua esquerda, deslocando para direita cada elemento maior do que a chave;



- Insira a chave na posição correta à sua esquerda, onde os elementos já estão ordenados;
- Repita os passos anteriores atualizando a chave para a próxima posição à direita até o fim do vetor.

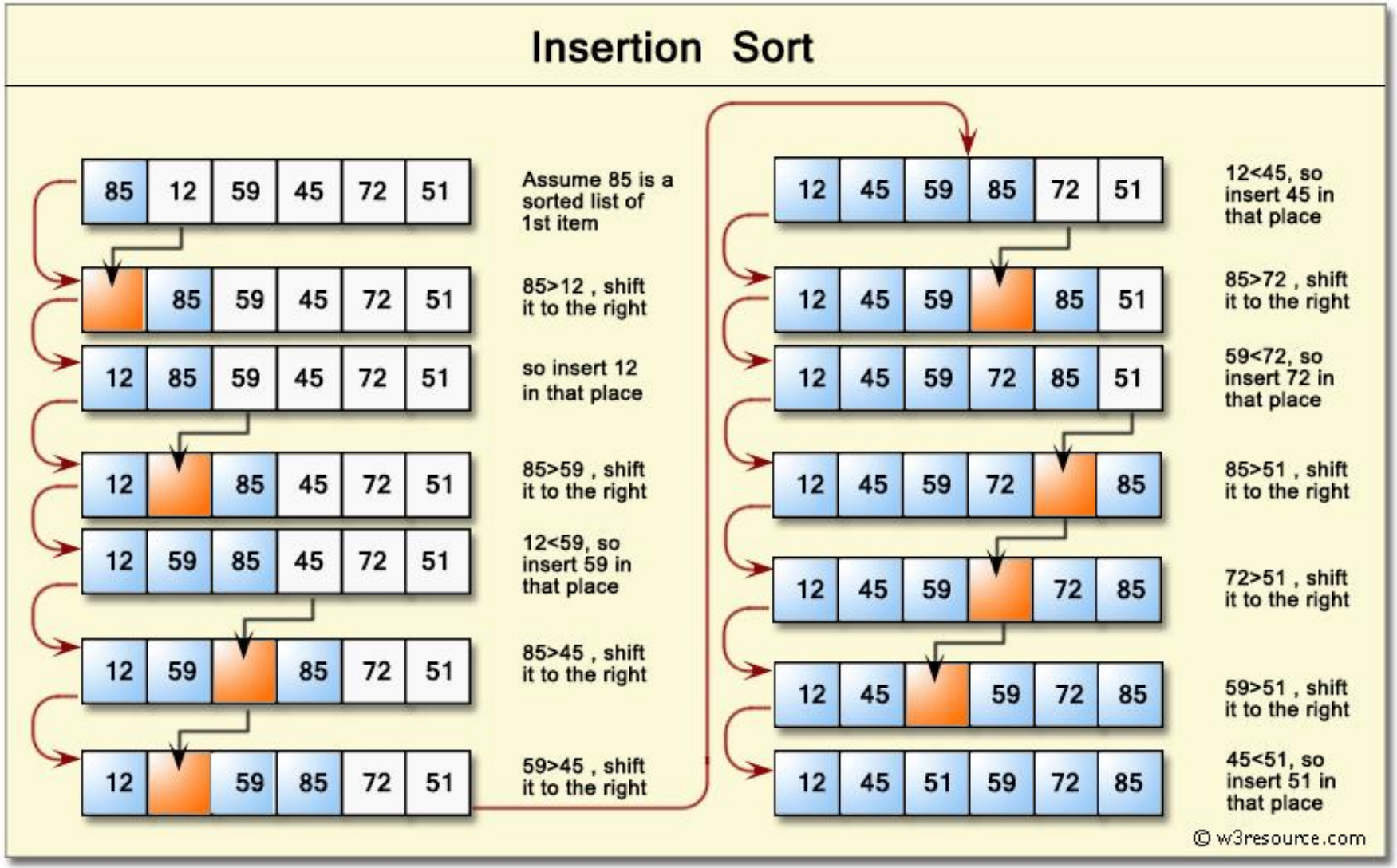
AlgoRythmics

https://youtu.be/ROalU379l3U

Animação

6 5 3 1 8 7 2 4

Ilustração


Insertion Sort
85 12 59 45 72 51
Assume 85 is a sorted list of 1st item
85 59 45 72 51
85>12 , shift it to the right
12 85 59 45 72 51
so insert 12 in that place
12 85 45 72 51
85>59 , shift it to the right
12 59 85 45 72 51
12<59, so insert 59 in that place
12 59 85 72 51
85>45 , shift it to the right
12 59 85 72 51
59>45 , shift it to the right
12 45 59 85 72 51
12<45, so insert 45 in that place
12 45 59 85 51
85>72 , shift it to the right
12 45 59 72 85 51
59<72, so insert 72 in that place
12 45 59 72 85
85>51 , shift it to the right
12 45 59 72 85
72>51 , shift it to the right
12 45 59 72 85
59>51 , shift it to the right
12 45 51 59 72 85
45<51, so insert 51 in that place
© w3resource.com

Pseudocódigo

- PSEUDOCÓDIGO: suponha um vetor v de tamanho n.

```
DECLARE i, j, x, n, v[n] NUMERICO;
PARA i = 1 ATÉ i < n FAÇA
INICIO
  x = v[i];
  j = i - 1;
  ENQUANTO j >= 0 e v[j] > x FAÇA
  INÍCIO
    v[j+1] = v[j];
    j = j-1;
  FIM
  v[j+1] = x;
FIM
```

Exercício de Fixação

- Implemente na linguagem C o algoritmo de ordenação Insertion sort. Utilize uma função auxiliar para implementar a ordenação.

Solução

```c
#include <stdio.h>
#include <stdlib.h>

void insertionSort(int v[], int n)
{
    int i, j, x;
    for(i = 1; i < n; i++) {
        x = v[i];
        j = i - 1;

        while(j >= 0 && v[j] > x) {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = x;
    }
}

int main()
{
    int v[] = {5, 6, -9, 9, 0, 4 };
    int n = 6, i;

    insertionSort(v, n);

    printf("\n\nVetor ordenado:\n");
    for(i = 0; i < n; i++)
        printf("%d\t", v[i]);
    printf("\n");
    return 0;
}
```

ifce.edu.br

Análise de Complexidade

- O menor número de comparações e trocas ocorre exatamente quando os elementos estão originalmente em ordem, e o número máximo de comparações e trocas ocorre quando os itens estão originalmente na ordem oposta.

- Para arquivos já ordenados o algoritmo descobre a um custo $\boldsymbol{O(n)}$ que cada item já está em seu lugar.

- É um método bom para adicionar um pequeno conjunto de dados a um arquivo já ordenado, originando um outro arquivo ordenado, pois neste caso o custo pode ser considerado linear.

Análise de Complexidade

- Se o valor a ordenar possui n elementos, então o algoritmo realizará n –1 etapas.

- **Quantas comparações e trocas serão realizadas?**
  - **No melhor caso:** vetor ordenado, serão realizadas 1 comparação e 1 troca por etapa, um total de $(n-1)$ comparações e $(n-1)$ trocas.

Análise de Complexidade

- **No pior caso**: vetor em ordem inversa, serão realizadas sucessivamente $1,\ 2,\ 3,\ \ldots,\ n-1$ comparações e trocas.
- A soma dos termos dessa progressão aritmética será $\boldsymbol{n^2/2}$.
- Pode ser demonstrado que para um vetor aleatório, o número aproximado de comparações e trocas é $\boldsymbol{n^2/4}$.
- Portanto, a complexidade deste algoritmo é quadrática: $\boldsymbol{T = O(n^2)}$.

Análise de Complexidade

- **No anel mais interno, na i-ésima iteração, o valor de C_i é:**
  - **Melhor caso:** $C_i(número\ de\ comparações) = 1$
  - **Pior caso:** $C_i(número\ de\ comparações) = i$
  - **Caso médio:** $C_i(número\ de\ comparações) = 1/i(1 + 2 + \ldots + i) = (i + 1)/2$

- **Se todas as permutações de n são igualmente prováveis para o caso médio, então, o número de comparações é igual a:**
  - **Melhor caso:** $C(n) = (1 + 1 + \ldots + 1) = n - 1$
  - **Pior caso:** $C(n) = (2 + 3 + \ldots + n) = n^2/2 + n/2 - 1$
  - **Caso médio:** $C(n) = 1/2(3 + 4 + \ldots + n + 1) = n^2/4 + 3n/4 - 1$

Análise de Complexidade

- **O número de movimentações na i-ésima iteração é igual a Mi(n) = Ci(n) – 1 + 3 = Ci(n) + 2 logo, o número de movimentos é igual a:**
  - **Melhor caso:** $M(número\ de\ elementos\ do\ arquivo) = (3 + 3 + \ldots + 3) = 3(n-1)$
  - **Pior caso:** $M(número\ de\ elementos\ do\ arquivo) = (4 + 5 + \ldots + n + 2) = n^2/2 + 5n/2 - 3$
  - **Caso médio:** $M(número\ de\ elementos\ do\ arquivo) = 1/2(5 + 6 + \ldots + n + 3) = n2/4 + 11n/4 - 3$

- **Deste modo podemos concluir que:**
  - **Melhor caso:** $O(n)$
  - **Pior caso:** $O(n^2)$
  - **Caso médio:** $O(n^2)$

SELECTION SORT

Definição

- Conhecido também como **ordenação por seleção**, este método consiste em:
  - Selecione o menor elemento do vetor;
  - Troque esse elemento com o elemento da primeira posição do vetor;
  - Repita as duas operações anteriores considerando apenas os $n-1$ elementos restantes, em seguida repita com os $n-2$ elementos restantes; e assim sucessivamente até que reste apenas um elemento no vetor a ser considerado.

AlgoRythmics

Select-sort with Gypsy folk dance

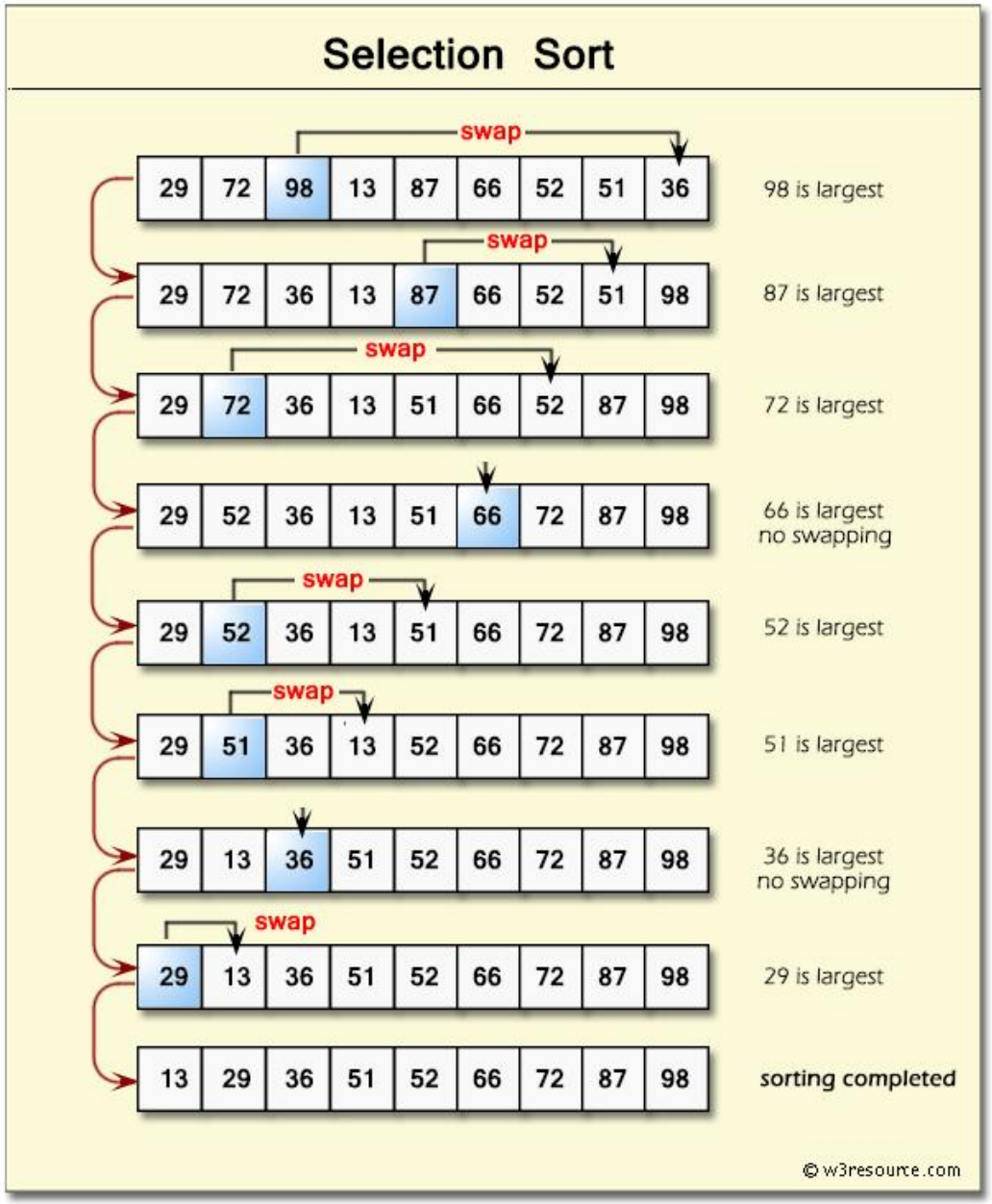
https://youtu.be/Ns4TPTC8whw

Animação

Ilustração

Esta figura ilustra outra maneira de implementar o Selection sort:

Seleciona o maior elemento do vetor e coloca na última posição.



Pseudocódigo

```
DECLARE i, j, aux, n, min, v[n] NUMERICO;
PARA i = 0 ATÉ i < n-1 FAÇA
INICIO
  min = i;
  PARA j = i+1 ATÉ j < n FAÇA
  INICIO
    SE v[j] < v[min] ENTAO
      min = j;
  FIM
  aux = v[i]; v[i] = v[min]; v[min] = aux;
FIM
```

Exercício de Fixação

- Implemente na linguagem C o algoritmo de ordenação Selection sort. Utilize uma função auxiliar para implementar a ordenação.

Solução

```c
#include <stdio.h>
#include <stdlib.h>

void selectionSort(int v[], int n)
{
    int i, j, aux, min;

    for(i = 0; i < n-1; i++) {
        min = i;
        for(j = i+1; j < n; j++)
            if(v[j] < v[min])
                min = j;
        aux = v[i]; v[i] = v[min]; v[min] = aux; //troca
    }
}

int main()
{
    int v[] = {5, 6, -9, 9, 0, 4 };
    int n = 6, i;

    selectionSort(v, n);

    printf("\n\nVetor ordenado:\n");
    for(i = 0; i < n; i++)
        printf("%d\t", v[i]);
    printf("\n");
    return 0;
}
```

ifce.edu.br

Análise de Complexidade

- A operação entre as chaves é feita no loop k, para cada valor de i são realizadas $(i-1)$ comparações no loop, como i varia de 2 até n, o número total de comparações para ordenar a lista toda é que para qualquer valor de i existe no máximo uma troca, se no caso a lista já estiver ordenada não ocorre troca.

- Pior caso existe uma troca para cada loop de k $(n-1)$ para cada troca exige três movimentos.

- O algoritmo de seleção é considerado um dos mais simples, além disso, possui uma característica eficiente quanto à quantidade de movimentações de registros, com um tempo de execução linear no tamanho de entrada.

Análise de Complexidade

- Geralmente é utilizado para arquivos de registros maiores com até 1000 registros.

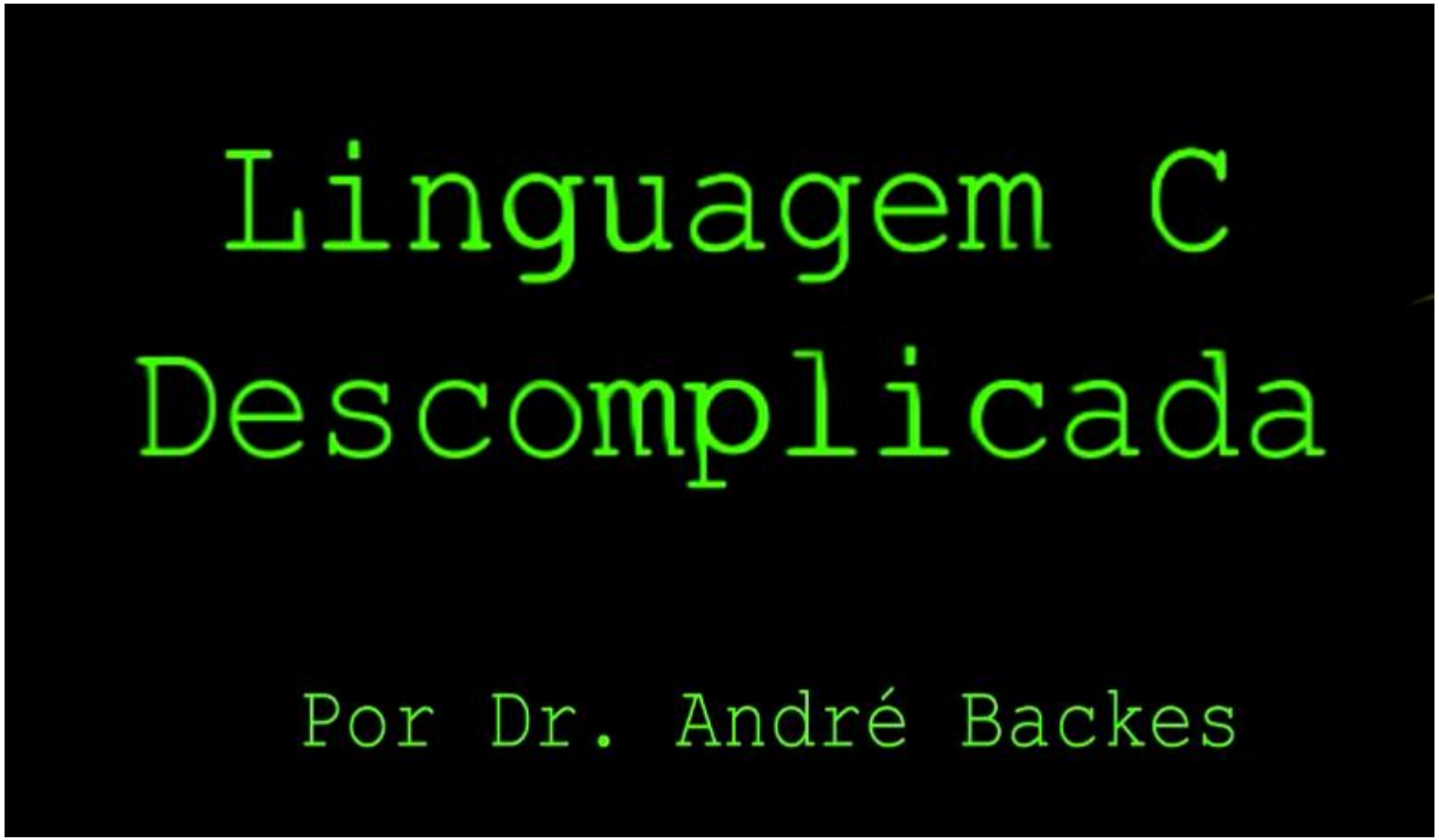
$$T = O(n^2)$$

- O fato de o conjunto já estar ordenado não ajuda em nada.
- O algoritmo não é estável, isto é, os registros com chaves iguais nem sempre irão manter a mesma posição relativa de antes do início da ordenação.

Dúvidas?

Vídeo-Aulas

- Aulas 47 até 50



Vídeo-Aulas

- Bubble Sort - Ordenação em Bolha - Canal do Código

Vídeo-Aulas

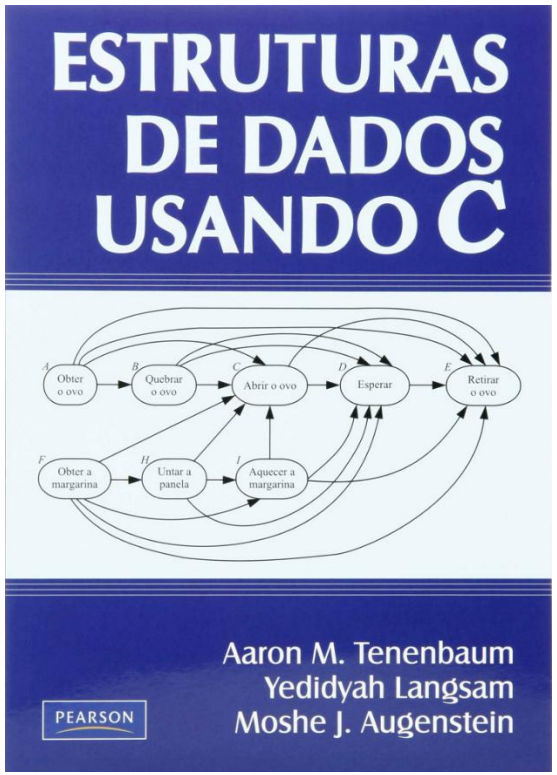
- Insertion Sort - Ordenação por Inserção - Canal do Código

Vídeo-Aulas

- Selection Sort (ordenação por seleção) - Canal do Código

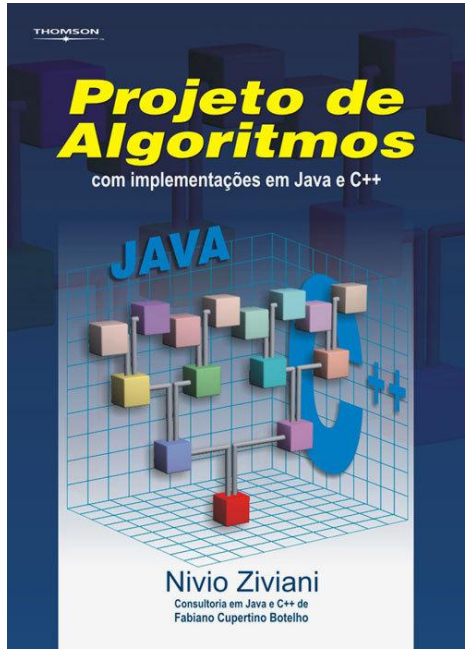
Bibliografia

- SHILDT, Herbert. **C, Completo e Total.** 3ª edição. São Paulo: Makron Books, 1996.
- **Capítulo 19**

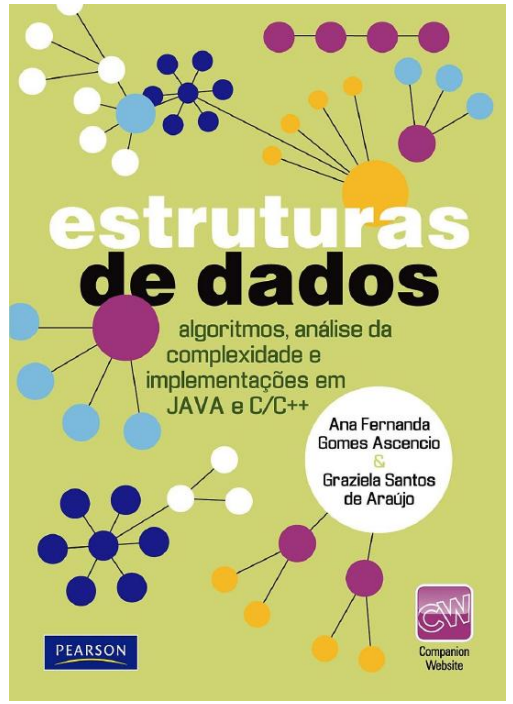


- TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. **Estruturas de dados usando C.** São Paulo : MAKRON *Books,* 1995.
- **Capítulo 06**

Bibliografia



- ZIVIANI, Nivio. **Projeto de Algoritmos: com implementações em Java e C++.** 1ª edição. São Paulo: Cengage Learning, 2013.
- **Capítulo 04**



- ASCENCIO, Ana Fernanda Gomes. Estruturas de dados: análise da complexidade e implementações em Java e C/C++ . São Paulo, Pearson Prentice Hall, 2010.
- **Capítulo 02**