

Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br

OBJETIVO

- Capacitar o aluno na **análise e seleção de algoritmos de classificação de dados e de pesquisa de dados**
- Específicos:
 - Caracterizar, analisar e selecionar estruturas de dados e suas representações;
 - **Conhecer os principais conceitos e técnicas para implementação de algoritmos de classificação e pesquisa de dados existentes e comparar a sua eficácia em diferentes conjuntos de dados;**
 - **Compreender os aspectos introdutórios do processo de análise de complexidade de algoritmos com base nos algoritmos estudados na disciplina, sabendo determinar os que apresentam o melhor e o pior custo em termos de tempo e espaço;**
 - **Implementar algoritmos de manipulação de dados e analisar sua complexidade para solucionar problemas específicos.**

CONTEÚDO PROGRAMÁTICO:

- MÉTODOS DE CLASSIFICAÇÃO DE DADOS
 1. FAMÍLIAS DE MÉTODOS DE CLASSIFICAÇÃO DE DADOS
 2. MÉTODO DA BOLHA (BUBBLESORT) E SUAS VARIAÇÕES
 3. MÉTODO DE ORDENAÇÃO POR SELEÇÃO (SELECTIONSORT)
 4. MÉTODO DE ORDENAÇÃO POR INSERÇÃO DIRETA (INSERTIONSORT)
 5. MÉTODO DA PARTIÇÃO E TROCA (QUICKSORT)
 6. ...
- PESQUISA DE DADOS
 1. FAMÍLIAS DE MÉTODOS DE PESQUISA DE DADOS
 2. PESQUISA SEQUENCIAL (PESQUISA LINEAR)
 3. PESQUISA BINÁRIA
 4. PESQUISA DIGITAL
 5. ÁRVORES DE BUSCA
 6. ...
- INTRODUÇÃO À ANÁLISE DA COMPLEXIDADE DE ALGORITMOS

Avaliação

- Provas escritas, exercícios, estudos de casos, **participação**.
- N1 = Listas de exercícios (20%), participação (10%), Trabalho 1 (30%) e Prova 1(40%);
- N2 = Listas de exercícios (20%), participação (10%), Trabalho 2 (30%) e Prova 2 (40%);
- Critérios de avaliação:
 - Média final = $(N1 + N2) / 2$
 - Média final $\geq 7,0 \rightarrow$ Aprovado
 - Média final $< 7,0 \rightarrow$ Avaliação N3 (Exame)

Parceria

- Espera-se boa conduta:
 - Não colar
 - Não trapacear
 - Chatgpt (pode ser usado para complementar, nunca para substituir o trabalho)

Importante:

- Code Race: **30/08/2024** às 20 horas



1º Lugar - R\$ 3000,00



2º Lugar - R\$ 1000,00



3º Lugar - R\$ 500,00

No ato da inscrição, cada equipe terá direito no dia do evento a:

- Uma pizza gigante para a equipe.
- Um copo personalizado do evento para cada participante.
- Acesso gratuito a café e energético para toda equipe.
- Certificado de participação com horas para toda a equipe.

The image features a white background with decorative blue lines in the corners. In the top-left corner, a blue line forms a jagged, stepped shape. In the top-right corner, a blue line forms a similar jagged, stepped shape. In the bottom-left corner, a blue line forms a jagged, stepped shape. In the bottom-right corner, a blue line forms a jagged, stepped shape.

Revisão: Estruturas de Dados



O que é estruturas de dados?

O que é estruturas de dados?

- Organização de dados
 - Acesso rápido
 - Baixo consumo de memória
 - Modificação eficiente

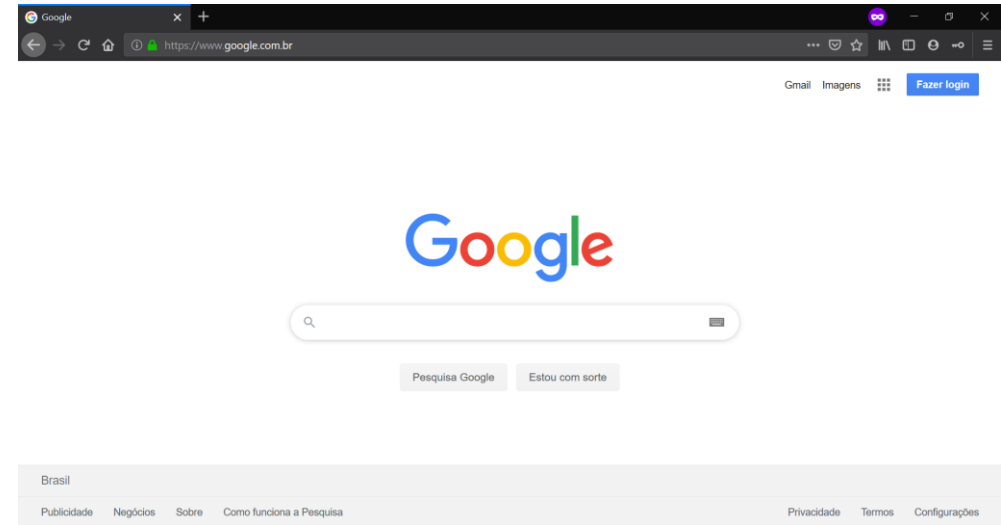
O que é estruturas de dados?

- Organização de dados
 - Acesso rápido
 - Baixo consumo de memória
 - Modificação eficiente
- Exemplos:



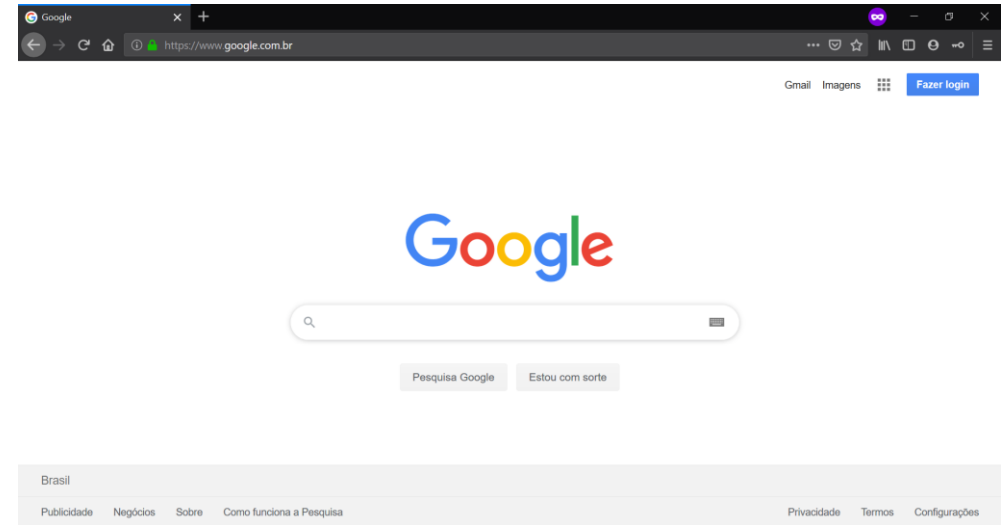
O que é estruturas de dados?

- Por que o Google é popular?
 - Por ser bonito?
 - Qual a dificuldade de se criar o Google?



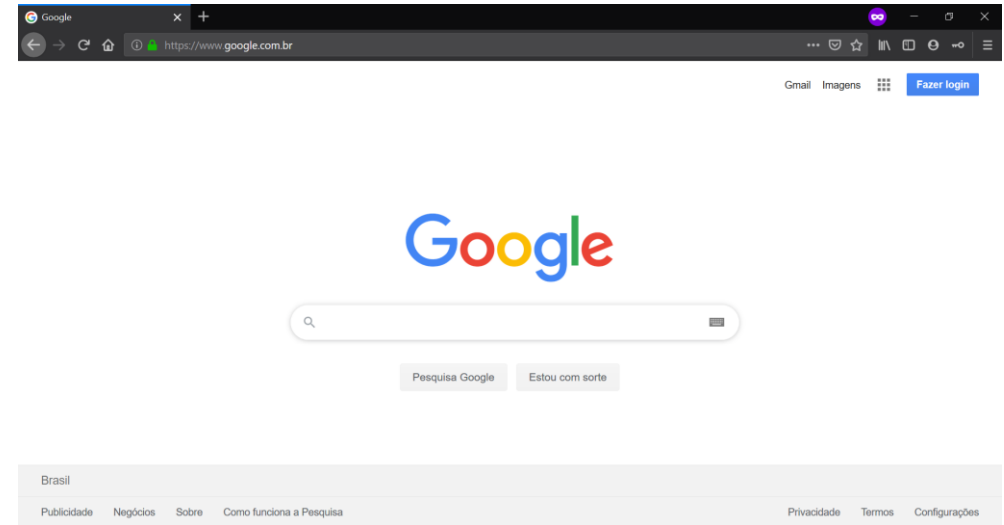
O que é estruturas de dados?

- Por que o Google é popular?
 - Por ser bonito?
 - Qual a dificuldade de se criar o Google?
 - Ele é eficiente
 - Rápido
 - Resultados esperados



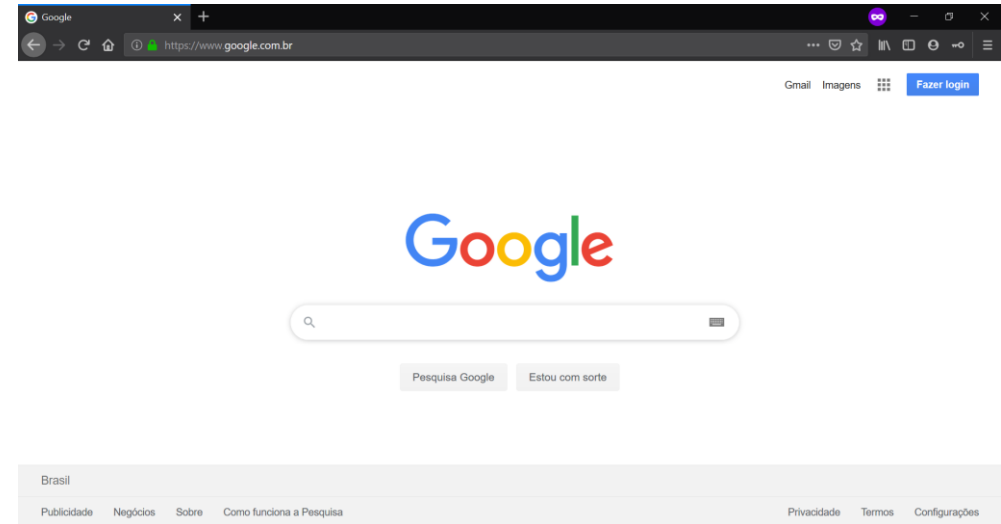
O que é estruturas de dados?

- Por que o Google é popular?
 - Por ser bonito?
 - Qual a dificuldade de se criar o Google?
 - Ele é eficiente
 - Rápido
 - Resultados esperados
- O que o faz especial?



O que é estruturas de dados?

- Por que o Google é popular?
 - Por ser bonito?
 - Qual a dificuldade de se criar o Google?
 - Ele é eficiente
 - Rápido
 - Resultados esperados
- O que o faz especial?
 - As estruturas de dados utilizadas
- É importante conhecer estruturas de dados?
 - O pessoal do google ficou rico por conhecer bem estrutura de dados



Definição: Estruturas de dados

- Uma estrutura de dados **é um meio para armazenar e organizar dados com o objetivo de facilitar o acesso e as modificações.**
- **Nenhuma estrutura de dados única funciona bem para todos os propósitos**, e assim é importante conhecer os pontos fortes e as limitações de várias delas

Ordenação

- Vamos ordenar um lista de números

 Random	 Insertion	 Selection	 Bubble	 Shell	 Merge	 Heap	 Quick	 Quick3
 Nearly Sorted								
 Reversed								
 Few Unique								

Somente para números?

- Essas técnicas só funcionam para números?
 - Se definirmos uma função que diga a partir de dois elementos qual o maior, podemos aplicar esses algoritmos para diversos tipos de dados
 - Exemplos?
 - Alfabeto
 - Cerveja
 - ...

Como saber qual método é mais eficiente?

- Ele roda mais rápido em um computador novo do que em um velho?
- Ele é mais simples?
- Depende da aplicação! **Essa é a regra mais importante dessa disciplina.**
 - Por isso precisamos saber as vantagens e desvantagens de cada técnica
 - Existe muita diferença de desempenho entre elas?

Revisão: Ponteiros

Revisão: Variáveis e endereços

- Uma **variável** é um espaço da memória principal reservado para armazenar dados.
- Variáveis possuem:
 - **Nome:**
 - Identificador usado para acessar o conteúdo.
 - **Tipo:**
 - Determina a capacidade de armazenamento.
Ex: int, char, float, ...
 - **Endereço:**
 - Posição na memória principal.

Revisão: Variáveis e endereços

Declaração

```
int num = 0;
```

Produz
→

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0

Revisão: Variáveis e endereços

Declaração

```
int num = 0;
```

```
int num2;
```

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	indefinido

?



Revisão: Variáveis e endereços

Memória Principal

End.	Valor
→ 0	
1	
2	
3	0
4	
5	
6	IND.
7	
...	...

O que acontece
quando esse
comando é
executado?

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	Int	6	indefinido

num2 = 5;

Revisão: Variáveis e endereços

Memória Principal

End. Valor

0	
1	
2	
3	0
4	
5	
6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5

num2 = 5;

Revisão: Variáveis e endereços

Memória Principal

End. Valor

0	
1	
2	
3	0
4	
5	
→ 6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5

num2 = 5;

ou

endereço 6 = 5;

Revisão: Variáveis e endereços

Memória Principal

End. Valor

0	
1	
2	
3	0
4	
5	
6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5

num2 = 5;

ou

Podemos fazer isso
manualmente?

endereço 6 = 5;

Revisão: Variáveis e endereços

Memória Principal

End. Valor

0	
→ 1	6
2	
3	0
4	
5	
6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
→ num3	*int	1	6

```
int *num3 = &num2;  
*num3 = 7;
```

Revisão: Variáveis e endereços

Memória Principal

End.	Valor
0	
→ 1	6
2	
3	0
4	
5	
6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
→ num3	*int	1	6

?

```
int *num3 = &num2;  
*num3 = 7;
```

Revisão: Variáveis e endereços

Memória Principal

End.	Valor
0	
→ 1	6
2	

Declaração de ponteiro do tipo int, ou seja, variável que armazena endereço de memória de variáveis do tipo int

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
num3	*int	1	6

?

```
int *num3 = &num2;  
*num3 = 7;
```

Revisão: Variáveis e endereços

Memória Principal

End.	Valor
0	
→ 1	6
2	
3	0
4	
5	
6	5
7	
...	...

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
→ num3	*int	1	6

Para onde vai o 7?

int *num3 = &num2;
*num3 = 7;

Deveria ir para 1, certo?

Revisão: Variáveis e endereços

Memória Principal

End. Valor

0	
1	6
2	
3	0
4	
5	
6	7
7	
...	...

Mudamos o valor
no endereço 6, o
que muda o valor
de num2

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	7
num3	*int	1	6

`int *num3 = 6;`
`*num3 = 7;`

Revisão: Variáveis e endereços

Sintaxe:

tipo *nome_da_variavel_ponteiro;

asterisco indica que a variável armazena um endereço de memória cujo conteúdo é do tipo especificado.

qualquer tipo válido em C

Revisão: Variáveis e endereços

&

operador unário, que devolve o endereço de memória de seu operando

Exemplo:

```
int num;
```

```
&num;
```

← Retorna o **endereço** de num

Revisão: Variáveis e endereços

&

operador unário, que devolve o endereço de memória de seu operando

Exemplo:

```
int num;
```

```
&num;
```



Retorna o **endereço** de num

*

operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int *num;
```

```
*num;
```



Retorna o **valor** que se encontra no **endereço** que num **aponta**

Revisão: Variáveis e endereços

&

operador
memória
Exemplo:

```
int num;  
&num;
```

Não faz o mesmo quando utilizado na declaração de uma variável!!!
Na declaração o * diz a variável é do tipo ponteiro.

Retorna o **endereço** de num

*

operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int *num;  
*num;
```

Retorna o **valor** que se encontra no **endereço** que num **aponta**

Revisão: Variáveis e endereços

float *f; // f é um ponteiro para variáveis do tipo float

int *i; // i é um ponteiro para variáveis do tipo inteiro

char a, b, *p, c, *q; // podemos declarar junto com
// variáveis de mesmo tipo

- O asterisco é o mesmo da operação de multiplicação, mas não provoca confusão porque seu significado depende do contexto em que é usado!
- Quantos usos vocês já identificaram para o *?
- Multiplicação, declaração de variável do tipo ponteiro e retorno do valor na posição de memória a qual o ponteiro aponta

Revisão: Variáveis e endereços

```
int count, q,*m;
```

```
count = 10;
```

```
m = &count; // m recebe endereço de  
            // memória da variável count
```

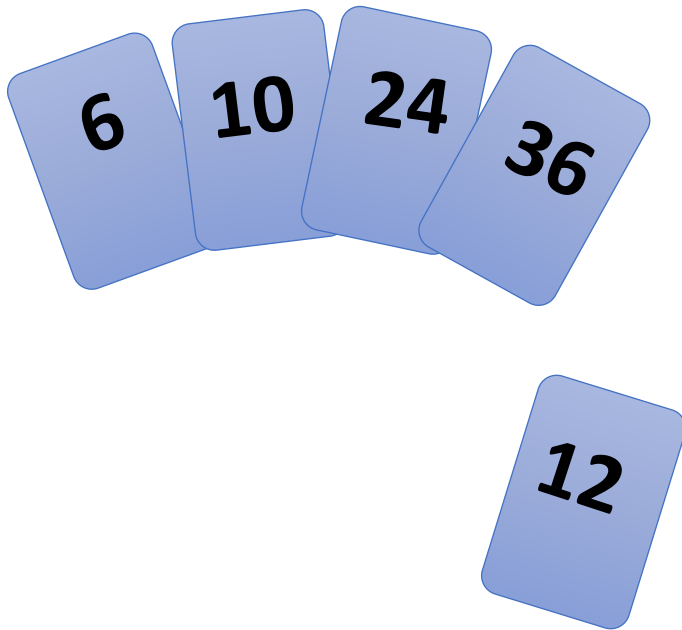
```
q = *m; // q recebe o valor armazenado no endereço apontado por m
```



Insertion Sort

Método de Inserção (Insertion Sort)

- Método do jogador de cartas



Ordenação (Exemplo)

7 2 8 5 9



2 7 8 5 9



2 7 8 5 9



2 7 8 5 9



2 7 8 5 9



2 7 5 8 9



2 5 7 8 9



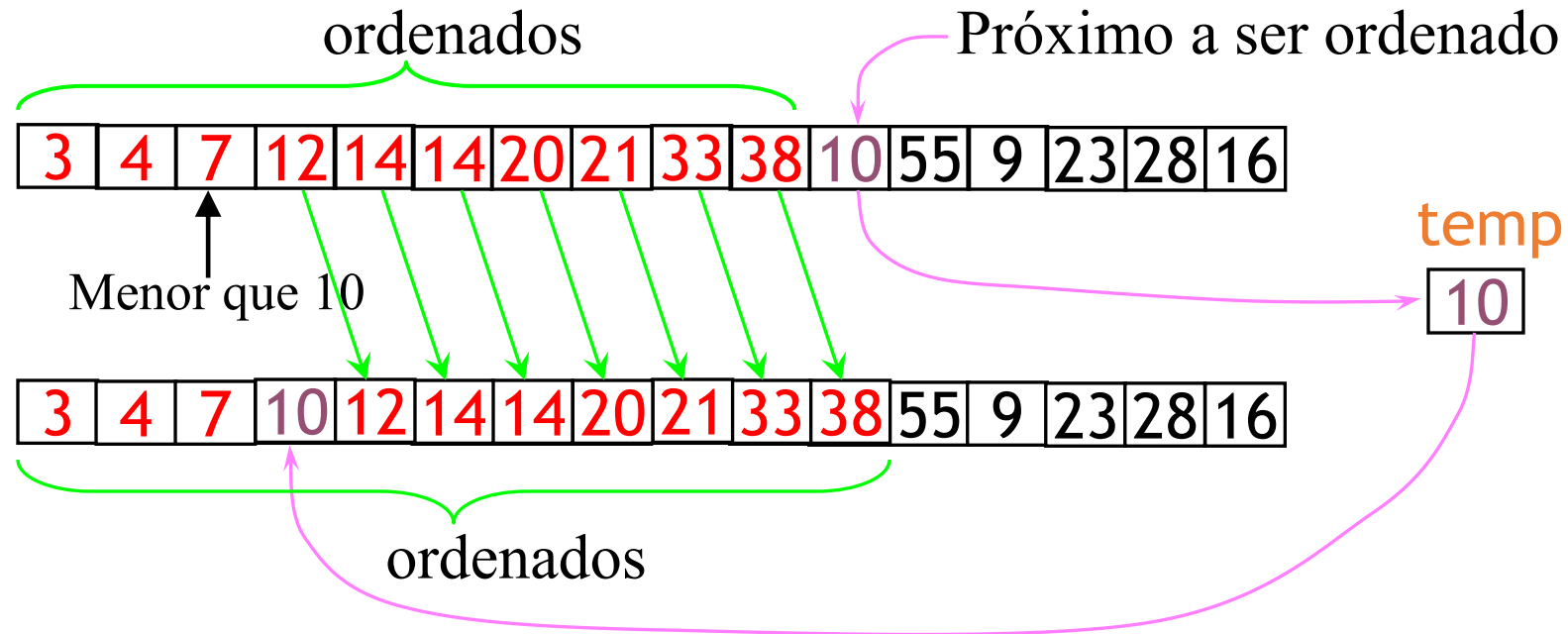
2 5 7 8 9



2 5 7 8 9



Um passo de ordenação



Algoritmo

```
Insertionsort (int data[], int n) {  
    int tmp,i,j;  
    for (j=1; j<n; j++) {  
        i =j - 1;  
        tmp = data[j];  
        while ( (i>=0) && (tmp < data[i]) ) {  
            data[i+1] = data[i];  
            i--;  
        }//while  
  
        data[i+1] = tmp;  
    }//for  
}//Insertionsort
```

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```

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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i = j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```

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



Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

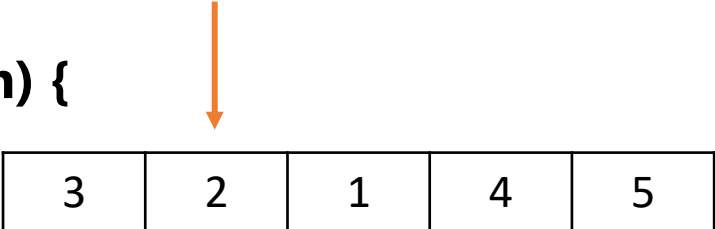
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

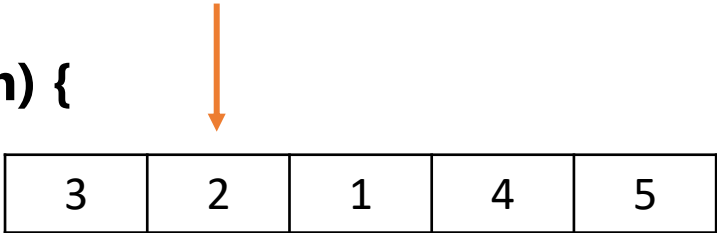
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i = j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i])) {
```

```
            data[i+1] = data[i];
```

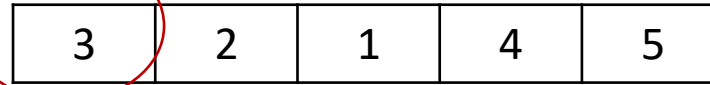
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

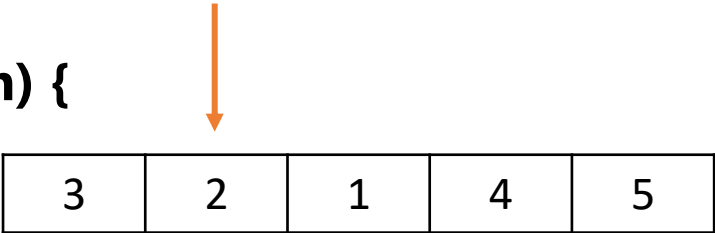
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

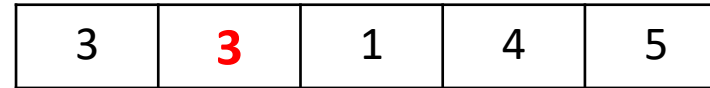
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

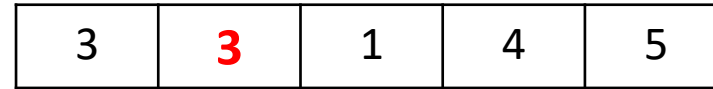
```
            i--;
```

```
        }//while
```

```
        data[i+1] = tmp;
```

```
    }//for
```

```
}//Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i = j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i >= 0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

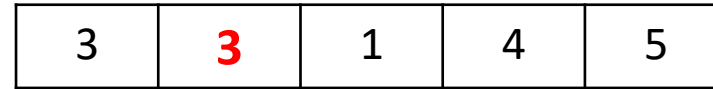
```
            i--;
```

```
        } //while
```

```
        data[i+1] = tmp;
```

```
    } //for
```

```
} //Insertionsort
```



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

Algoritmo

```
Insertionsort (int data[], int n) {
```

```
    int tmp,i,j;
```

```
    for (j=1; j<n; j++) {
```

```
        i =j - 1;
```

```
        tmp = data[j];
```

```
        while ( (i>=0) && (tmp < data[i]) ) {
```

```
            data[i+1] = data[i];
```

```
            i--;
```

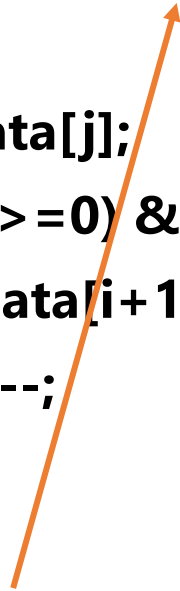
```
        }//while
```

```
        data[i+1] = tmp; // -1 +1 =0
```

```
    }//for
```

```
}//Insertionsort
```

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



Estudo de caso

- Qual o melhor caso?
 - Os dados já estarem ordenados

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

- Qual o pior?
 - Os dados estarem na ordem inversa

9	8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---	---

Exercício

- Implementar um algoritmo que implemente o *Insertion Sort* modo *easy* depois modo divertido (lista encadeada)
- Testar com elementos já ordenados
- Testar com elementos ordenados na ordem inversa
- Testar com elementos duplicados
- Testar com elementos aleatórios sem repetição



Selection Sort

Selection Sort

- Selection Sort ou Ordenação por Seleção
 - Cada passo, procura o menor valor do vetor e o coloca na primeira posição
 - Descarta a primeira posição do vetor e repete o processo para a segunda posição.
 - Isso é feito para todas as posições

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

i=2

-8	4	67	23	90	54	21
-8	4	21	23	90	54	67

Troca

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=3

-8	4	21	23	90	54	67
-8	4	21	23	90	54	67

OK

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

i=2

-8	4	67	23	90	54	21
-8	4	21	23	90	54	67

Troca

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=3

-8	4	21	23	90	54	67
-8	4	21	23	90	54	67

OK

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

i=4

-8	4	21	23	90	54	67
-8	4	21	23	54	90	67

Troca

i=2

-8	4	67	23	90	54	21
-8	4	21	23	90	54	67

Troca

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=3

-8	4	21	23	90	54	67
-8	4	21	23	90	54	67

OK

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

i=4

-8	4	21	23	90	54	67
-8	4	21	23	54	90	67

Troca

i=2

-8	4	67	23	90	54	21
-8	4	21	23	90	54	67

Troca

i=5

-8	4	21	23	90	90	67
-8	4	21	23	54	67	90

Troca

Selection Sort

Sem Ordenação

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

i=0

23	4	67	-8	90	54	21
-8	4	67	23	90	54	21

Troca

i=3

-8	4	21	23	90	54	67
-8	4	21	23	90	54	67

OK

i=1

-8	4	67	23	90	54	21
-8	4	67	23	90	54	21

OK

i=4

-8	4	21	23	90	54	67
-8	4	21	23	54	90	67

Troca

i=2

-8	4	67	23	90	54	21
-8	4	21	23	90	54	67

Troca

i=5

-8	4	21	23	90	90	67
-8	4	21	23	54	67	90

Troca

Ordenado

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

Selection Sort

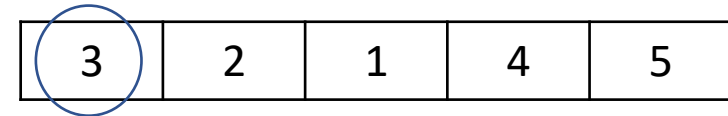
```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```



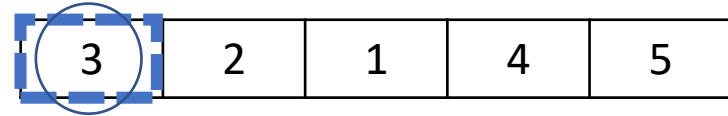
3	2	1	4	5
---	---	---	---	---

Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

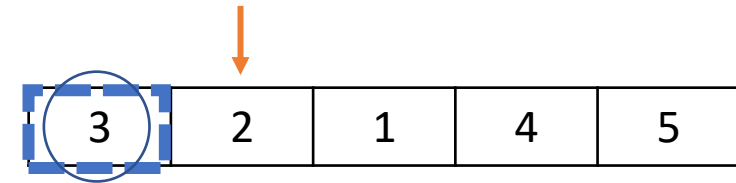


Procura menor elemento

Troca os valores da posição atual com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

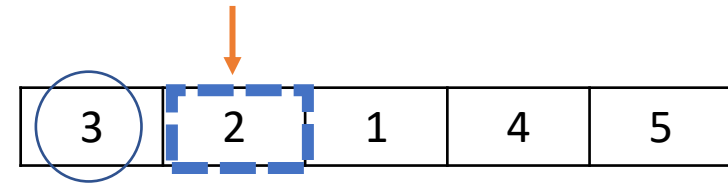


Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

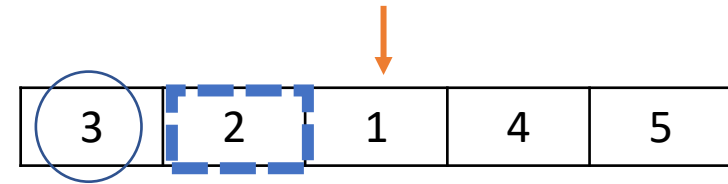


Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

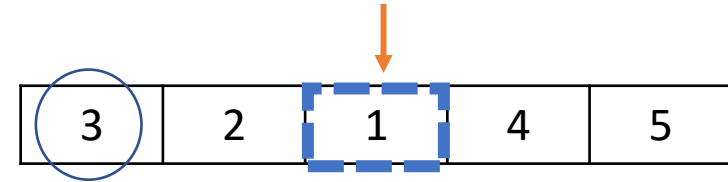


Procura menor elemento

Troca os valores da posição atual com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

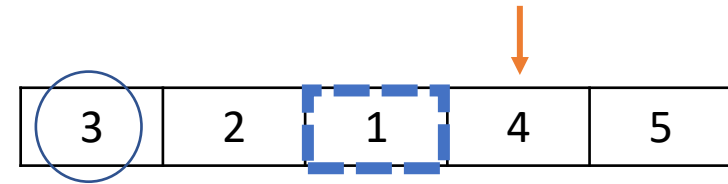


Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

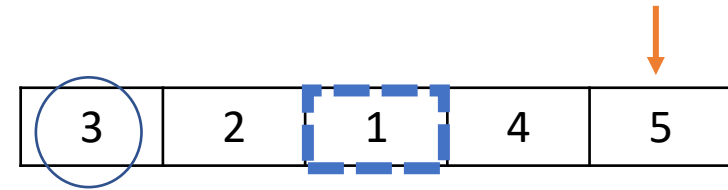


Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

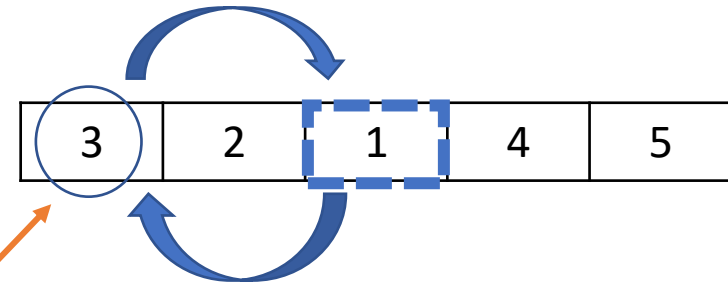


Procura menor elemento

Troca os valores da posição atual com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```

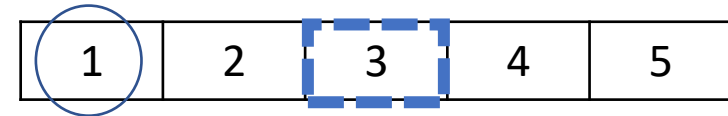


Procura menor elemento

Troca os valores da posição atual
com o menor

Selection Sort

```
Selectsort (int data[],int n) {  
    int menor,troca,i,j,menor_id;  
    for (i=0; i<n-1; i++) {  
        menor = data[i];  
        for (j=i+1; j<n; j++)  
            if (data[j] < menor)  
            {  
                menor = data[j];  
                menor_id = j;  
            }  
        troca = data[i];  
        data[i] = data[menor_id];  
        data[menor_id] = troca;  
    }  
}
```



Procura menor elemento

Troca os valores da posição atual com o menor

Exercício

- Implementar um algoritmo que implemente o *Selection Sort* modo *easy* depois modo divertido (lista encadeada)
- Testar com elementos já ordenados
- Testar com elementos ordenados na ordem inversa
- Testar com elementos duplicados
- Testar com elementos aleatórios sem repetição

Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br