Introdução aos Métodos de Ordenação

Prof. Dr. Eleandro Maschio
Tecnologia em Sistemas para Internet
Câmpus Guarapuava
Universidade Tecnológica Federal do Paraná (UTFPR)

Ordenação

- No mundo da computação, talvez as tarefas mais fundamentais e extensivamente analisadas sejam **ordenação** e **pesquisa**.
- Essas rotinas são utilizadas em praticamente todos os programas de banco de dados, bem como em compiladores, interpretadores e sistemas operacionais.
- Como o objetivo de ordenar os dados geralmente é **facilitar e acelerar o processo de pesquisa** no conjunto, usa-se discutir primeiro a ordenação.
- Embora seja extenso abordar em profundidade estudos sobre o tema, cabe a disciplina introduzir conceitos e métodos básicos de ordenação.

Conceito de Ordenação

- Ordenação é o processo de **arranjar** um conjunto de informações semelhantes numa ordem crescente ou decrescente. Dada uma lista ordenada ***i*** de ***n*** elementos, considerando a ordem crescente, então:

$$i_1 <= i_2 <= \ldots <= i_n$$

- Geralmente, quando a informação é ordenada, apenas uma porção dessa informação é usada como **chave da ordenação**. Essa chave é utilizada nas comparações, mas, quando uma troca se torna necessária, toda a estrutura de dados é transferida.

- Por exemplo, em uma lista postal, o código de endereçamento (CEP) poderia ser usado como chave. Entretanto os nomes do logradouro, do bairro, da cidade e do estado acompanhariam o CEP quando uma troca é feita.

Conceito de Ordenação

Possíveis chaves de ordenação e conjuntos de dados relacionados:

- CPF: lista de funcionários.
- CNPJ: lista de empresas.
- Nome completo: lista de chamada.
- Palavra: dicionários.
- Pontuação: ranking.
- Pontuação, depois idade: classificações em concursos.
- Data e hora da transação: movimentações bancárias.

Atividade Prática

(**1**) Realizem em duplas. Sem auxílio de computador.

(**2**) Separem as cartas de **1 a 10** de um naipe do baralho. Considerem o **Ás** como o número 1.

(**3**) **Embaralhem-as**.

(**4**) Criem um **método** para ordenar o conjunto de cartas. Independentemente da ordem.

(**5**) Registrem, em linguagem natural, a **sequência de passos** do método concebido.

Atividade Prática

(**6**) Agora, pensem na implementação deste método em uma linguagem de programação.

Quais são as **estruturas de dados** e **técnicas de programação** empregados?

(**7**) **Implementem** o método concebido. Ordenem um conjunto de **10 inteiros** aleatórios.

Estruturas de Dados e Técnicas de Programação

Métodos de ordenação podem empregar diversas estruturas de dados e técnicas de programação:

- Iteração.
- Recursividade.
- Matrizes unidimensionais.
- Tipos específicos de listas encadeadas.
- Tipos específicos de árvores.

Tipos de Métodos de Ordenação

Existem três métodos gerais para ordenar matrizes unidimensionais:

- Por **seleção**.
- Por **troca**.
- Por **inserção**.

Ordenação por Seleção

(**1**) Espalhe as cartas na mesa com os números voltados para cima. Os números que se deseja ordenar são as chaves

(**2**) Selecione a carta de **menor valor**, retire-a do baralho e segure-a em sua mão.

(**3**) Das **cartas restantes** na mesa, selecione a carta de **menor valor**, retire-a do baralho e coloque-a na sua mão após a **última**.

(**4**) Repita o **passo 3** até que **não haja mais cartas na mesa**.

(**5**) Observe que, em todo momento, as cartas da mão se encontram ordenadas. Ao final do processo, todas as cartas estarão ordenadas na mão de quem executar o método.

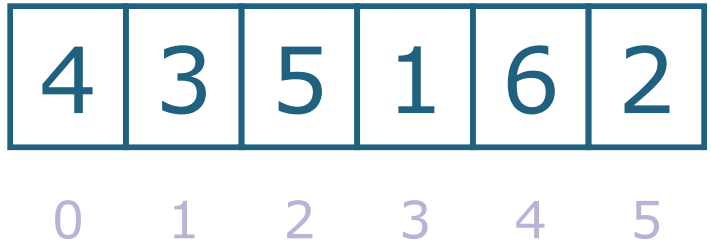
Selection Sort

Lógica de Implementação

- **Selecione** o elemento de **menor valor** e troque-o pelo **primeiro** elemento do conjunto.
- Dos **elementos restantes**, encontre o **menor valor** (segundo menor valor do todo) e troque pelo **segundo elemento**.
- Dos **elementos restantes**, encontre encontrado o **menor valor** (terceiro menor valor do todo) e troque pelo **terceiro elemento**.
- Continue a troca até restar um único elemento. Será o elemento de **maior valor**, ou seja, o último na ordem.

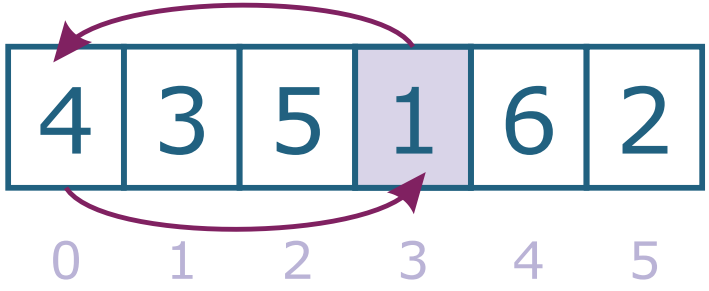
Dinâmica de Funcionamento

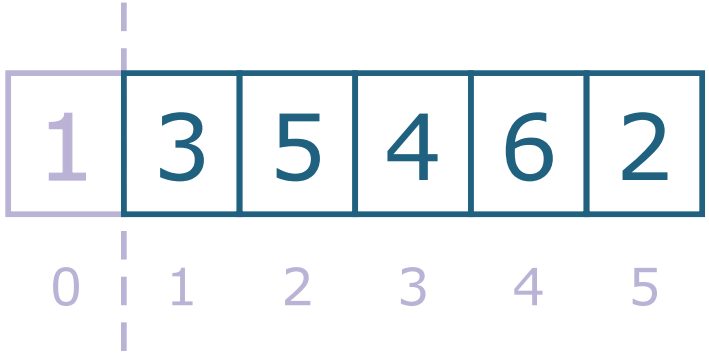
Considere a matriz unidimensional.



Dinâmica de Funcionamento

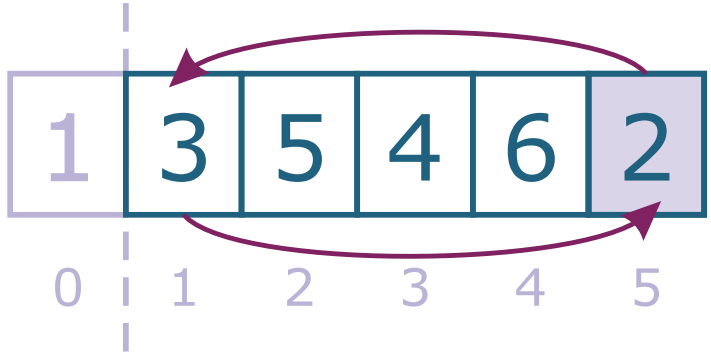
passo 1

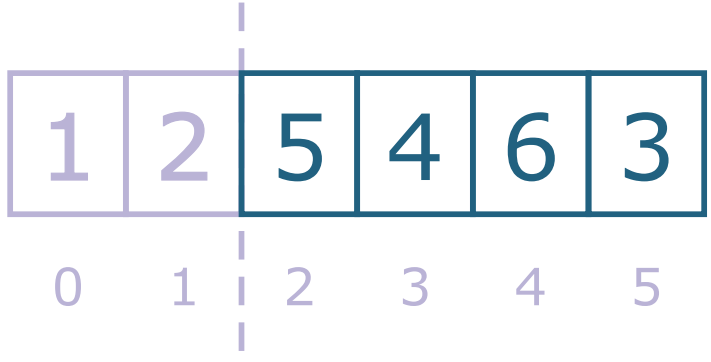




Dinâmica de Funcionamento

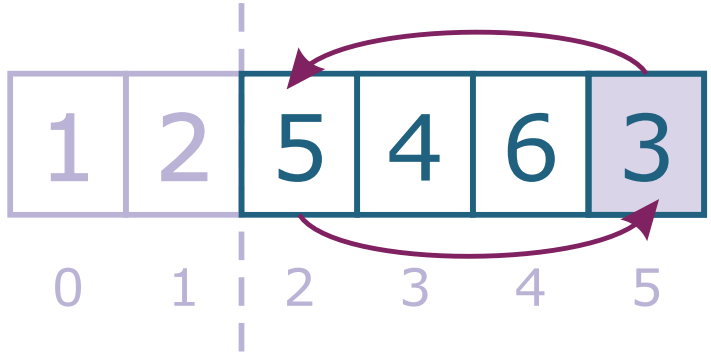
passo 2

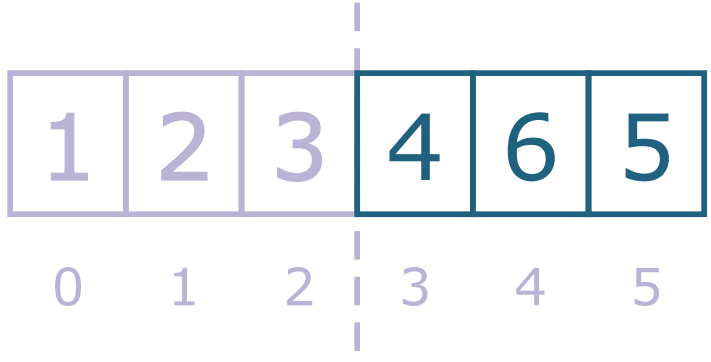




Dinâmica de Funcionamento

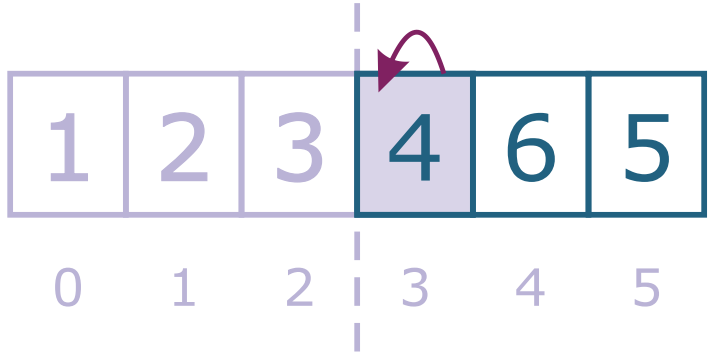
passo 3

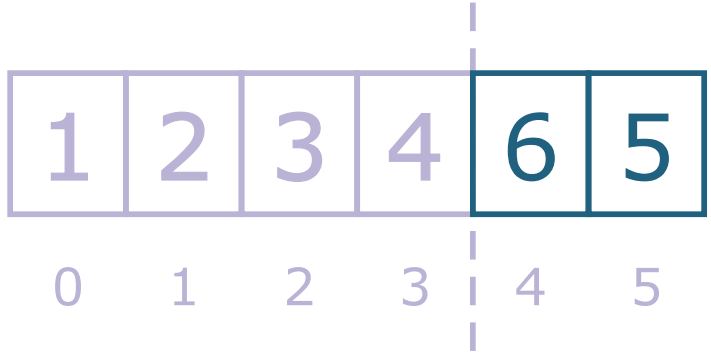




Dinâmica de Funcionamento

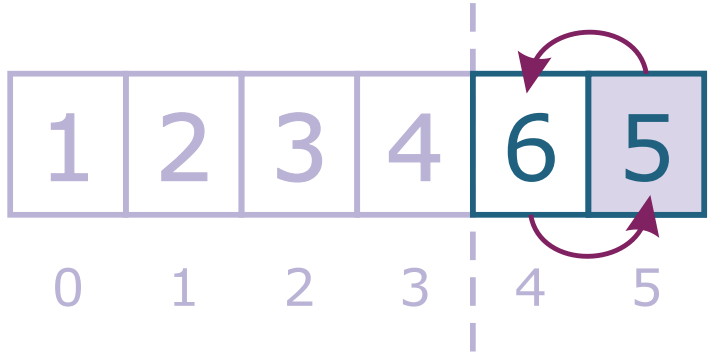
passo 4

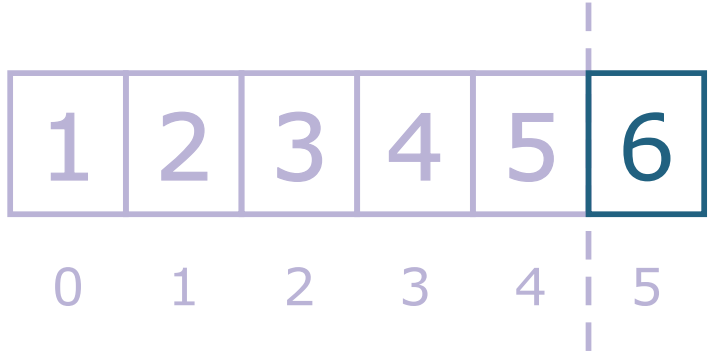




Dinâmica de Funcionamento

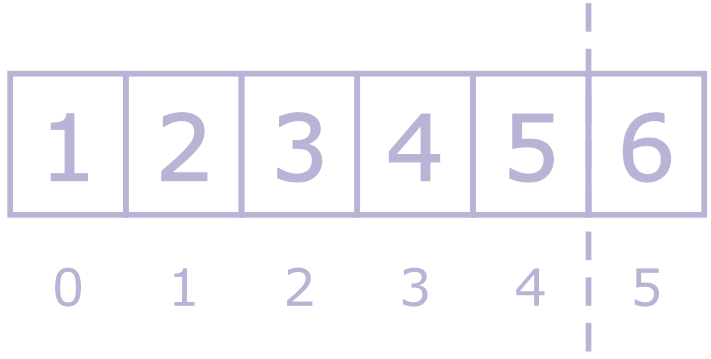
passo 5





Dinâmica de Funcionamento

concluído



Recursos

- **AlgoRythmics**
  Projeto da Sapientia University (Romênia) que mistura arte, cultura e tecnologia para potencializar o ensino de computação.
  http://www.youtube.com/user/AlgoRythmics

- **Sorting Algorithm Animations**
  Dispõe de simulações que possibilitam o entendimento e a comparação de 8 métodos de ordenação.
  http://www.sorting-algorithms.com/

- **Sorting Algorithms (Smith College)**
  Promove a simulação parametrizada de 9 métodos de ordenação.
  http://cs.smith.edu/~thiebaut/java/sort/

Implementação

Laço externo

Laço interno

Encontrar o *i*-ézimo menor elemento.

Trocar o *i*-ézimo menor elemento com aquele da posição *i*.

Implementação

Antes do método de ordenação

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TAM 10
```

Implementação

Antes do método de ordenação (...)

```
int main()
{
    // Para não gerar sempre a mesma sequencia.
    srand(time(NULL));

    printf("Selection Sort\n");
    printf("==============\n\n\n");

    int dados[TAM], i = 0;
    printf("Sequencia Gerada\n");
    printf("----------------\n");
    for (i = 0; i < TAM; i++)
    {
        dados[i] = rand() % 100;
        printf("%2d ", dados[i]);
    }
    //...
```

Implementação

Depois do método de ordenação

```
    //...
    printf("\n\n");
    printf("Sequencia Ordenada\n");
    printf("------------------\n");
    for (i = 0; i < TAM; i++)
    {
        printf("%2d ", dados[i]);
    }

    return(0);
}
```

Implementação

Encontrar a posição do menor elemento

```
//...
// Dizemos que o menor está na posição 0.
int menor = 0;

// Compara-se com as demais posições.
for (i = 1; i < TAM; i++)
{
    if (dados[i] < dados[menor])
       menor = i;
    // Se encontrar alguém menor do que dados[menor],
    // faz MENOR assumir aquela posição.
}

printf("\n\n");
printf("Posição do menor elemento: %d.\n", menor);
printf("Menor elemento: %d.", dados[menor]);

return(0);
```

Atividades

(**1**) Realize um teste-de-mesa com a sua implementação do Selection Sort. Determine a quantidade de **trocas** e **comparações** feitas pelo algoritmo. Utilize a sequência **4**, **3**, **5**, **1**, **6**, **2**.

Use o seguinte recurso para declarar a matriz (sendo TAM = 6):

```
int dados[TAM] = {4, 3, 5, 1, 6, 2};
```

(**2**) Faça os mesmo com a implementação sugerida do Selection Sort.

Selection Sort

Implementação Sugerida

```
int menor, j, aux;
for (i = 0; i < TAM-1; i++)
{
    menor = i;
    for (j = i + 1; j < TAM; j++)
    {
        if (dados[j] < dados[menor])
           menor = j;
    }

    if (i != menor)
    {
       aux = dados[i];
       dados[i] = dados[menor];
       dados[menor] = aux;
    }
}
```

Discussão

(**1**) Por que a implementação não funcionaria com outra estrutura de repetição além do `for`.

(**2**) Se o algoritmo levasse os maiores **elementos para o final**, ao invés dos menores para o início, isso descaracterizaria o Selection Sort?

(**3**) Quais alterações seriam necessárias para que o algoritmo colocasse os elementos em ordem **descendente**?.

(**4**) Qual seria o impacto, no algoritmo, caso a troca ocorresse toda vez que um elemento menor do que aquele na posição `i` fosse encontrado? Observe que, na implementação dada, a troca ocorre somente depois de encontrar o menor valor, considerando todo o restante da matriz.