Algoritmos de ordenação

Programação II – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

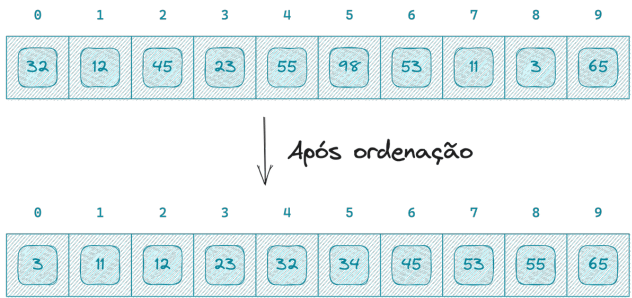
Licenciamento

Por que ordenar?

- **Por que ordenar?**
  - Gerar relatórios para usuários (i.e. lista de alunos ordenada pela matrícula)
  - Para reduzir a complexidade de um problema (i.e. busca binária)
- **Onde podem estar os dados que deseja-se ordenar?**
  - Memória principal (i.e. memória RAM)
  - Memória auxiliar (i.e. disco rígido)

Exercício

- Desenvolva um aplicativo em C que gere um vetor de inteiros de tamanho *n*, preenchido de forma aleatória e não ordenada
- Imprima o conteúdo do vetor no dispositivo de saída padrão, faça a ordenação dos elementos (ordem crescente) e depois imprima o conteúdo do vetor no dispositivo de saída padrão



Algoritmos de ordenação por comparação

Compara elementos de uma lista (ex: usando operador relacional $<$)

- **Troca e seleção**
  - *bubble sort*
  - *insertion sort*
  - *selection sort*
  - *heap sort*
- **Divisão e conquista**
  - *merge sort*
  - *Quicksort*

Classificação

Um algoritmo de ordenação é considerado como **estável** se preservar a ordem relativa de elementos que possuam valores idênticos

Algoritmos diferem na **complexidade em tempo** e **complexidade em espaço** (memória usada)

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

Método bolha (*bubble sort*)

- Compare elementos consecutivos de um vetor e faça a permuta caso estejam fora de ordem
- A cada iteração os elementos mais leves (número menor) sobem para a “parte mais alta” do vetor e os elementos mais pesados descem

E assim por diante...

Método bolha (*bubble sort*)

Análise e pseudocódigo

- $\Theta(n)$ – melhor caso
- $O(n^2)$ – pior caso
- $O(n^2)$ – caso médio
- Faz um grande número de permutas
  - $O(n^2)$
- É um algoritmo **estável**

Algoritmo 1: Bubble sort

Entrada: vetor[1..n]
Saída: vetor ordenado

```
1 para i de 1 até n faça
2     para j de 1 até n − i faça
3         se vetor[j] < vetor[j + 1] então
4             aux ← vetor[j];
5             vetor[j] ← vetor[j + 1];
6             vetor[j + 1] ← aux;
7         fim
8     fim
9 fim
```

Ordenação por inserção (*insertion sort*)

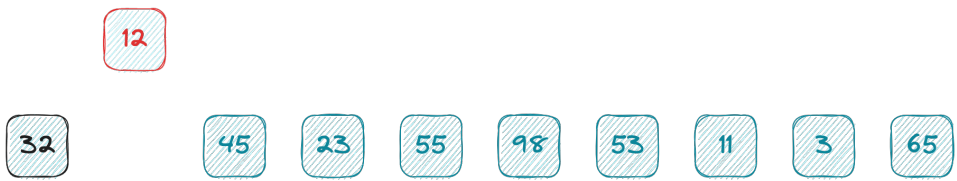
- Algoritmo eficiente para ordenar um número pequeno de elementos
- Semelhante a forma que as pessoas ordenam as cartas em jogo de buraco (canastra ou biriba)
- Elementos em um vetor são divididos em subestruturas
  - elementos já ordenados (no lado esquerdo)
  - elementos ainda por ordenar (no lado direito)

Fonte: (CORMEN et al., 2012)

Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)

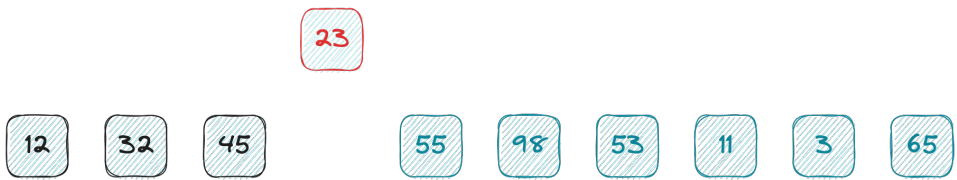
Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

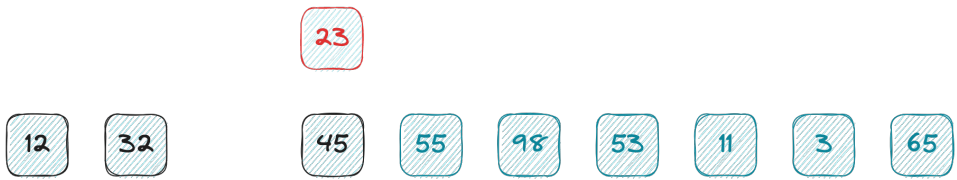
Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

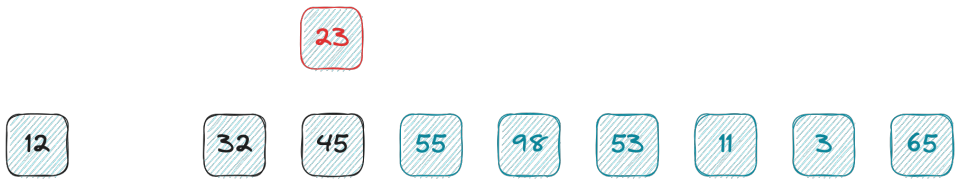
Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

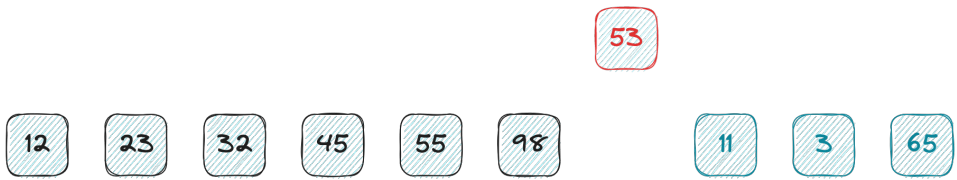
Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

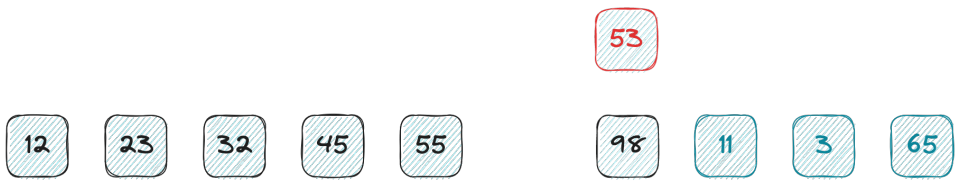
Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

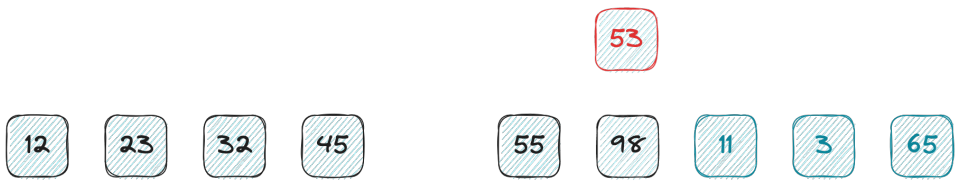
Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)



Ordenação por inserção (*insertion sort*)

Ordenação por inserção (*insertion sort*)

E assim continua...

Ordenação por inserção (*insertion sort*)

Análise e pseudocódigo

- $\Theta(n)$ – melhor caso
  - elementos ordenados
- $\Theta(n^2)$ – pior caso
  - elementos na ordem inversa
- $O(n^2)$ – caso médio
  - geralmente é quase tão ruim quanto o pior caso
- É um algoritmo **estável**

Algoritmo 2: Insertion sort

Entrada: vetor[1..n]
Saída: vetor ordenado

```
1 para i de 1 até n − 1 faça
2     chave ← vetor[i];
3     j ← i − 1;
4     enquanto j ≥ 0 E vetor[j] > chave faça
5         vetor[j + 1] ← vetor[j];
6         j ← j − 1;
7     fim
8     vetor[j + 1] ← chave
9 fim
```

Adequado para problemas com poucos elementos ou quando a entrada está quase totalmente ordenada

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

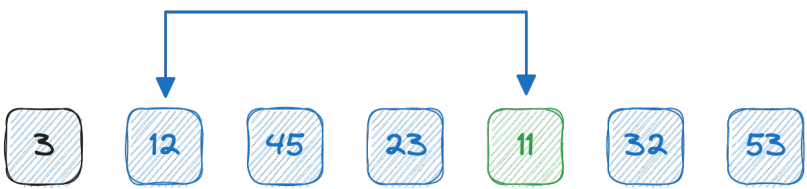
- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante



Ordenação por seleção (*selection sort*)

- Selecione o menor elemento do vetor e troque este com o elemento da primeira posição e assim por diante

E assim continua...

Ordenação por seleção (*selection sort*)

Análise e pseudocódigo

- $O(n^2)$ – melhor caso
  - Mesmo em um vetor ordenado
- $O(n^2)$ – pior caso
- $O(n^2)$ – caso médio
- Faz pouca permuta de elementos
  - $O(n)$
- É um algoritmo **não estável**

Algoritmo 3: Selection sort

Entrada: vetor[1..n]
Saída: vetor ordenado

```
1  para i de 1 até n − 1 faça
2      min ← i;
3      para j de (i+1) até n faça
4          se vetor[j] < vetor[min] então
5              min ← j;
6          fim
7      fim
8      se i ≠ min então
9          aux ← vetor[i];
10         vetor[i] ← vetor[min];
11         vetor[min] ← aux;
12     fim
13 fim
```

Curiosidades

- Em 2007, o ex CEO da Google pergunta ao candidato Barack Obama sobre a melhor forma de ordenar um vetor de inteiros com 1 milhão de elementos
  - https://youtu.be/m4yVlPqeZwo?t=1370
- **Comparação interativa de algoritmos de ordenação**
  - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
  - https://www.toptal.com/developers/sorting-algorithms
- **Quinze algoritmos de ordenação em 6 minutos**
  - https://www.youtube.com/watch?v=kPRA0W1kECg
- **Danças folclóricas**
  - *bubble sort* – https://www.youtube.com/watch?v=lv3vgjM8Pv4
  - *insertion sort* – https://www.youtube.com/watch?v=ROalU379l3U
  - *selection sort* – https://www.youtube.com/watch?v=0-W8OEwLebQ

Exercício 1

- Evolua a biblioteca `libprg`, criada por você na aula de lista de sequencial, para ofertar diferentes funções para ordenar um vetor de inteiros por meio dos algoritmos
  a. método bolha, de forma crescente e decrescente
  b. ordenação por inserção
  c. ordenação por seleção, de forma crescente ou decrescente
- As definições das funções devem ser feitas obrigatoriamente no arquivo de cabeçalho `libprg.h`
- As implementações das funções obrigatoriamente no arquivo com o nome `alg_ord_troca.c`

Exercício 2

- Crie um projeto com CMake de uma aplicação em C que dependa da biblioteca `libprg`, criada no exercício anterior
  - Faça uso do `FetchContent` no CMake para baixar a dependência
- Faça um programa que crie um vetor de inteiros de tamanho n, informado pelo usuário, e faça o povoamento desse vetor com números aleatórios (0 ... `INT_MAX`)
  - `INT_MAX` é definida na biblioteca `limits.h`
- Ordene o mesmo vetor usando cada um dos algoritmos de ordenação que implementou em sua biblioteca `libprg`
  - Dica: Se iniciar a função `srand` sempre com uma mesma semente, então conseguirá reproduzir a ordem dos números sorteados. Ex: `srand(1)`
- Deve-se imprimir na tela o tempo gasto (relógio de parede e de CPU) por cada um dos algoritmos

Exercício 2 (continuação)

Exemplo de saída esperada após execução

```
Tamanho do vetor: 1000
-------------------------------------
Algoritmo   Relógio (seg)  CPU (seg)
-------------------------------------
bolha       0.001827       0.001828
inserção    0.000357       0.000357
seleção     0.000723       0.000722
-------------------------------------
```

Referências

CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. LTC, 2012. Disponível em: <https://app.minhabiblioteca.com.br/reader/books/9788595158092>.