# Estruturas de Dados 2

Prof. Silvia Brandão

2024.1

## ATENÇÃO

Na aula 4, do dia 13/03, falaremos a respeito dos próximos slides (avaliação).

**Boa calourada!**

# Avaliações

- 1ª avaliação: 03/04, 20pts
  - Projeto: ordenação e análise – Bubble, Selection e Insertion – Artigo
- 2ª avaliação: 22/05, 20pts
  - Projeto: ordenação e análise – Quick, Merge, Heap e Shell – Artigo
- 3ª avaliação: 25/06, 20pts
  - Projeto c/ Menu – inserção, remoção, alteração, relatório com dados ordenados (escolher o método conforme a base dados) e busca (pesquisa e análise).

O uso de banco de dados no projeto com MENU seria opcional.

- Os 5pts, de cada momento, seriam de trabalho e avaliação contínua, durante as aulas práticas, com a participação do aluno.

# Estudo de Caso – apresentação no formato de artigo

- **Título**
- **Introdução** – justificar o problema estudado e informar os objetivos do trabalho realizado
- **Desenvolvimento** – exposição ordenada e detalhada sobre o assunto
  - **Metodologia**
    - Escolha uma linguagem de programação, um computador e 3 arquivos de dados (strings ou numéricos – escolher apenas 1 deles)
    - Utilize os seguintes algoritmos de ordenação interna: *Bubble Sort, Selection Sort e Insertion Sort* para a realização prática do estudo.
    - Faça os testes necessários para verificar o comportamento dos algoritmos em relação ao tempo, movimentações de trocas e comparações.
  - **Referencial teórico** – citar e referenciar as obras utilizadas.
- **Resultados** – devem, à luz do aporte teórico utilizado na pesquisa, evidenciar análise e discussão dos dados obtidos.
  - Registre em uma tabela os resultados encontrados para os três testes de ordens de listas com 3 tamanhos diferentes (100, 1000, 10000) cada:
    - Ordem 1: lista ordenada em ordem crescente – arquivo1.txt ou arquivo1.dat
    - Ordem 2: lista ordenada em ordem decrescente – arquivo2.txt ou arquivo2.dat
    - Ordem 3: lista desordenada com números aleatórios – arquivo3.txt ou arquivo3.dat
  - Trace as curvas de desempenho dos métodos pra n x tempo de execução;
- **Conclusão** – considerar os objetivos explicitados e os resultados indicados no texto anterior
- **Referências** – citar apenas autores e obras mencionados no texto, obedecendo às normas da ABNT.

# Estudo de Caso – apresentação no formato de artigo

- O trabalho deve ser apresentado em formato eletrônico **(.doc ou .docx;)** configurando a página para o tamanho de papel A4, com orientação retrato, margem superior e esquerda igual a (3cm), inferior e direita igual a (2cm). Deve ser utilizada a fonte Times New Roman, corpo 12, espaçamento 1,5 entre linhas em todo o texto, parágrafo de 1,25 cm, alinhamento justificado, à exceção do título. A numeração da página deve constar à direita na parte inferior da folha, em algarismos arábicos.
- Máximo de 3 alunos
- Data de entrega e apresentação: 03/04 – 20PTOS (Avaliação A1)
- DIÁRIO DE BORDO: artigo + arquivos → basta 1 aluno do grupo postar

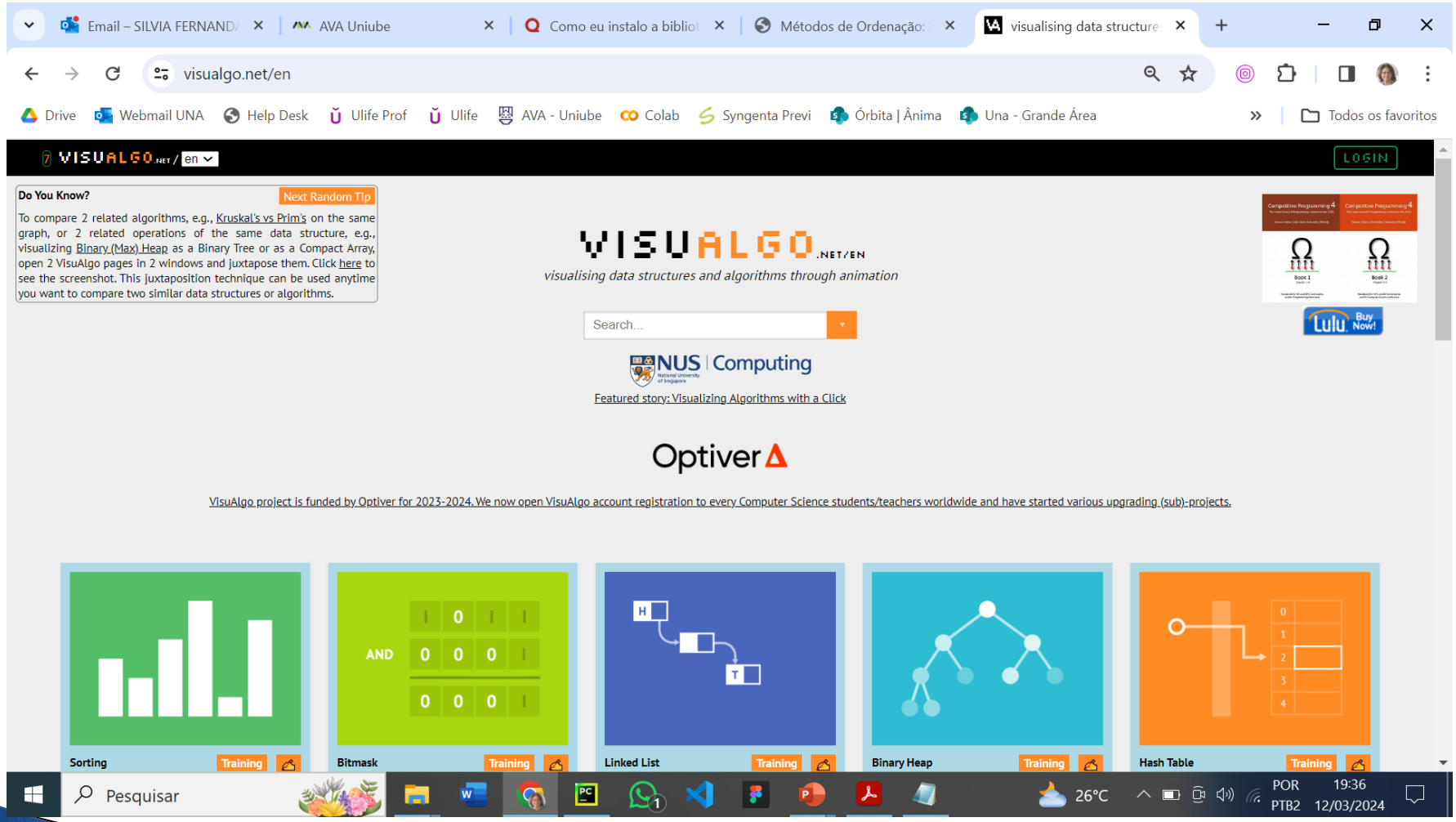
- **Modelo de tabela para apresentação dos resultados:**

| Ordem x | Tamanho do conjunto de dados = xxx | | |
|---|---|---|---|
| Algoritmo | Tempo(ms) | Comparações | Movimentações |
| Bubble sort | | | |
| Selection sort | | | |
| Insertion sort | | | |

# Método *Selection Sort* (ordenação por seleção)

- Algoritmo de ordenação bastante simples.
- A cada passo, ele seleciona o melhor elemento para ocupar aquela posição do array.
  - Maior ou menor, dependendo do tipo de ordenação.
  - Na prática, possui um desempenho quase sempre superior quando comparado com o *bubble sort*.
- Funcionamento
  - A cada passo, procura o menor valor do array e o coloca na primeira posição do array.
    - Divide o array em duas partes: a parte ordenada, a esquerda do elemento analisado, e a parte que ainda não foi ordenada, a direita do elemento.
  - Descarta-se a primeira posição do array e repete-se o processo para a segunda posição
  - Isso é feito para todas as posições do array

# Bora lá brincar?...

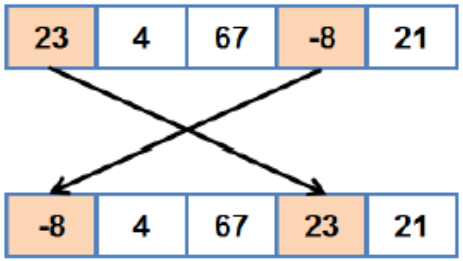



https://visualgo.net/en/sorting

# *Selection Sort* – algoritmo

```
//ENTRADA: UM VETOR V COM N POSIÇÕES
//SAÍDA: O VETOR V EM ORDEM CRESCENTE

PARA i = 0 até n - 2 FAÇA:
    menor = i
    PARA j = i + 1 até n - 1 FAÇA:
        SE V[j] < V[menor] ENTÃO
            menor = j
        FIM_SE
    FIM_PARA
    SE (i != menor) ENTÃO
        troca = V[i]
        V[i] = V[menor]
        V[menor] = troca
    FIM_SE
FIM_PARA
```

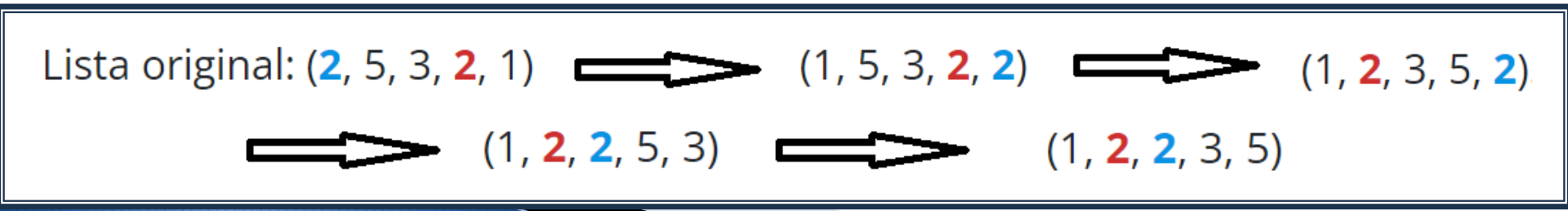



Procura o menor elemento em relação a “i”

Troca os valores da posição atual com a “menor”

# *Selection Sort* – algoritmo

- **Vantagens:**
  - Simples de ser implementado.
  - Não requer memória adicional.
- **Desvantagens:**
  - Sua eficiência diminui drasticamente a medida que o número de elementos no array aumenta.
  - Não é recomendado para aplicações que envolvam grandes quantidade de dados ou que precisem de velocidade.
  - O algoritmo de ordenação por seleção não é estável, pois ele escolhe o menor elemento da lista e o coloca na primeira posição, sem levar em consideração a ordem original dos elementos iguais. Isso pode resultar em mudanças na ordem original dos elementos iguais. Veja:



# *Selection Sort* – algoritmo

- **Complexidade:**
  - Considerando um array com N elementos, o tempo de execução é sempre de ordem O($n^2$).
  - A eficiência do *selection sort* não depende da ordem inicial dos elementos.
  - Melhor do que o *bubble sort*.
    - Apesar de possuírem a mesma complexidade no caso médio, na prática o *selection sort* quase sempre supera o desempenho do *bubble sort*, pois envolve um número menor de comparações.

| Algoritmo | Tempo | | |
|---|---|---|---|
| | Melhor | Médio | Pior |
| Bubble sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Insertion sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Selection sort | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Merge sort | $O(n \log_2 n)$ | $O(n \log_2 n)$ | $O(n \log_2 n)$ |
| Quick sort | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ |

# Recursos

- Sorting –
  - https://visualgo.net/en/sorting
  - https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
- Arquivos em Python:
  - https://awari.com.br/python-a-leitura-de-arquivos-txt/

# Prática

- Implemente o método *Selection Sort* junto ao código do *Bubble Sort*. Compare o tempo de cada ordenação para cada estrutura de dados.

# Próxima aula

- Métodos Simples – *Insertion Sort* (com análise de custo)