ANÁLISE E DESENVOLVIMENTO DE SISTEMAS- UNIFIP
ESTRUTURA DE DADOS

EQUIPE:
- JAYR MARTINS
- TIAGO RAMALHO
- RAFAEL

SELECTION SORT & MERGE SORT

Comparativo entre Algoritmo de Ordenação!

UNIFIP
CENTRO UNIVERSITÁRIO

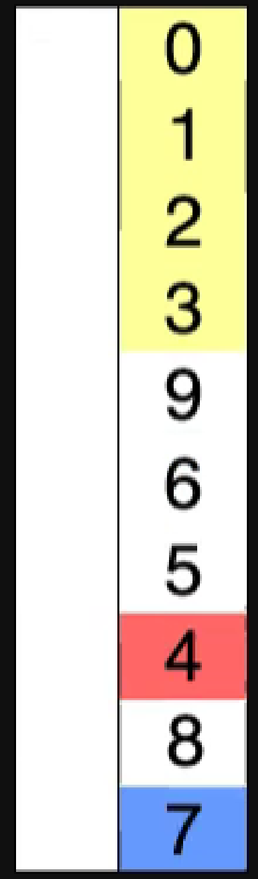
REPOSITÓRIO DO PROJETO

```
Selection e Merge/
├── .venv/
├── Selection e Merge/
│   ├── __pycache__/
│   ├── graficos/
│   ├── main.py
│   ├── merge.py
│   ├── selection.py
│   └── utils.py
```

UNIFIP
CENTRO UNIVERSITÁRIO

SELECTION SORT

A ordenação por seleção é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição.

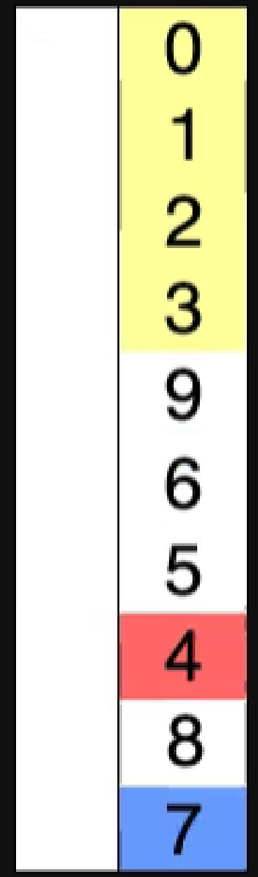


UNIFIP
CENTRO UNIVERSITÁRIO

FUNCIONAMENTO

Sempre O(n2)!

O selection sort compara a cada interação um elemento com os outros, visando encontrar o menor.

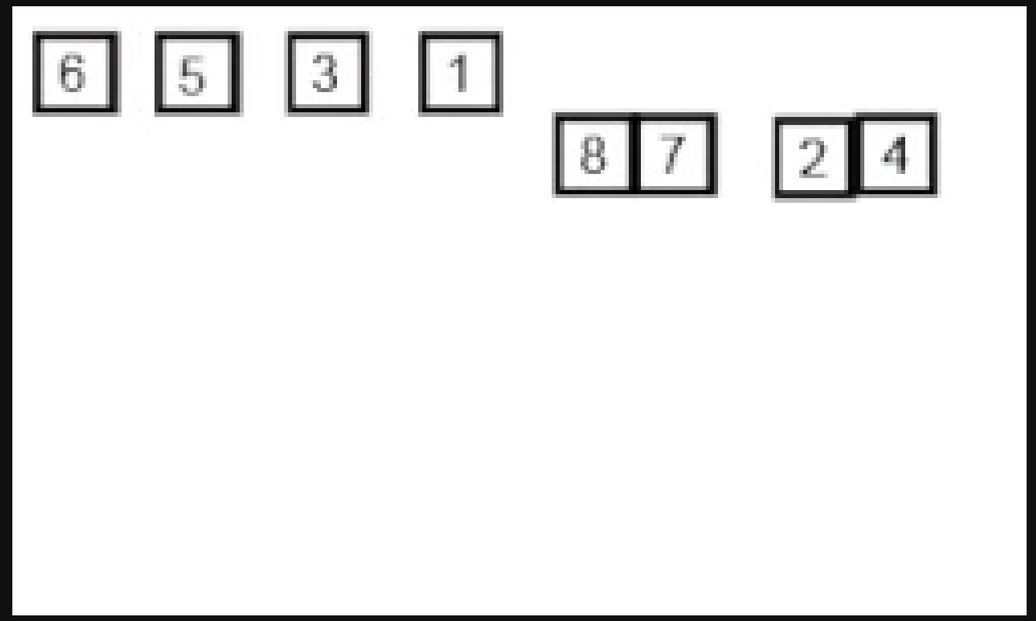


CENÁRIO DE USO SELECTION

- Em ambientes embarcados com microcontroladores de baixa capacidade (como Arduino)

- Em softwares pequenos (como em um CRUD local ou interface simples em Java Swing)

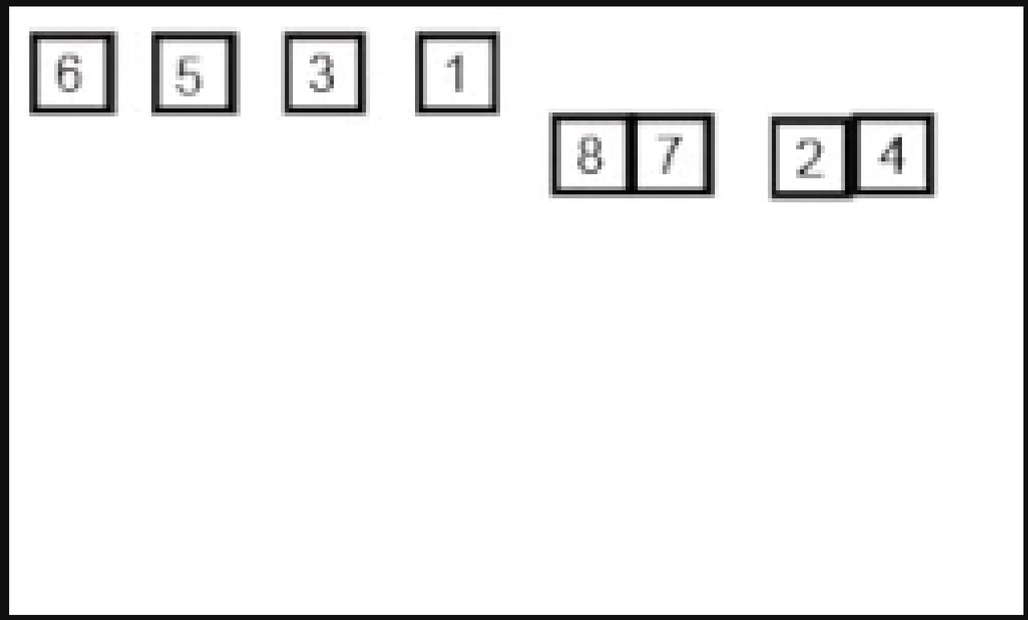
MERGE SORT



Ordenação por Mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar.

FUNCIONAMENTO

Θ(nlog2n) em todos os casos.



1- Dividir o Problema.

2- Resolver os Subproblemas.

3- Unir as soluções.

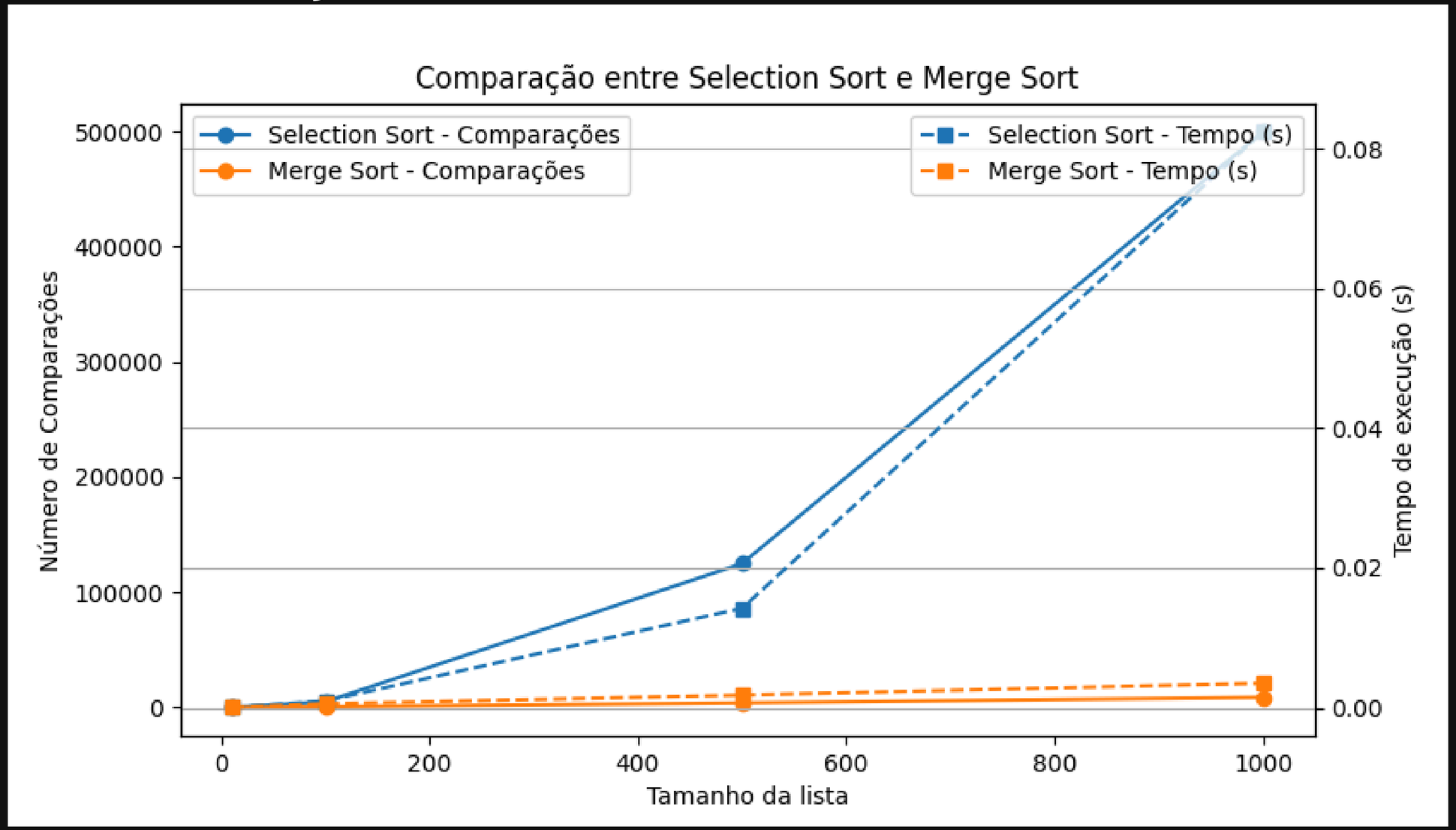
CENÁRIO DE USO MERGE

- Ordenação de grandes arquivos em disco (external sort) Ex.: Log PPPoE Redes.

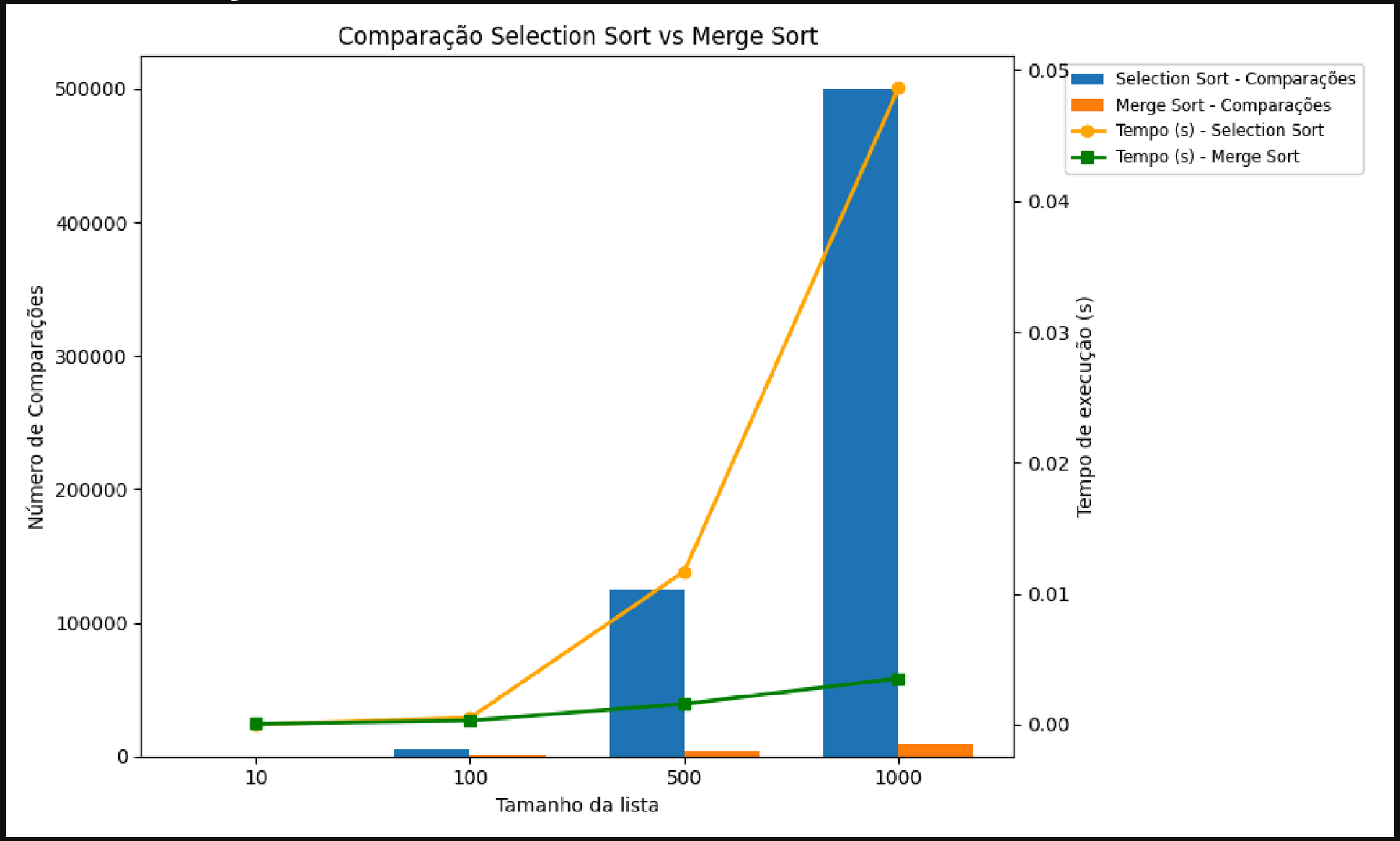
- Serviços de nuvem ou banco de dados
  - Exemplo: ordenar um SELECT com ORDER BY nome ASC que retorna milhões de registros.

UNIFIP CENTRO UNIVERSITÁRIO

COMPARAÇÕES



COMPARAÇÕES



CONCLUSÃO

Ao comparar Selection Sort e Merge Sort, observamos um forte contraste em suas características de desempenho.

Selection Sort apresenta uma complexidade de tempo de O(n^2) em todos os casos. Isso o torna relativamente ineficiente para conjuntos de dados maiores, visto que o número de comparações e trocas cresce quadraticamente com o tamanho da entrada.

UNIFIP
CENTRO UNIVERSITÁRIO

CONCLUSÃO

O **Merge Sort**, por outro lado, emprega uma estratégia de divisão e conquista, resultando em uma complexidade temporal de O(n log n), independentemente da ordem inicial dos dados de entrada. Essa escalabilidade superior permite que o Merge Sort manipule conjuntos de dados significativamente maiores com maior eficiência.