**ICMC - Instituto de Ciências Matemáticas e da Computação**

# Disciplina: SCC0201 - Introdução à Ciência de Computação II
# Projeto 2

**Aluno:** Alec Campos Aoki (15436800)
**Aluno:** Fernando Valentim Torres (NUSP)

## 1. Problema e Solução

O trabalho consistia de comparar os métodos de ordenação BubbleSort, SelectionSort, InsertionSort, ShellSort, QuickSort, HeapSort, MergeSort, Contagem de Menores e RadixSort utilizando vetores ordenados, inversamente ordenados e aleatórios. Foram feitas comparações de tempo de execução, quantidade de trocas e quantidade de comparações.

Para tanto, implementamos cada método de ordenação e utilizamos uma struct chamada Data para armazenar a quantidade de trocas e comparações de cada algoritmo. A cada troca ou comparação, incrementamos o campo correspondente dessa struct. Além disso, nos casos teste, utilizamos somente elementos pertences ao intervalo entre 0 e a quantidade de elementos do teste (no teste de 100 elementos, por exemplo, os números no teste vão de 0 a 100), para melhor padronizar os resultados obtidos.

Para melhor organizar as informações, agrupamos os algoritmos por complexidade. Sendo assim, temos os grupos:

1. $O(n^2)$;
   (a) BubbleSort;
   (b) Contagem de Menores;
   (c) InsertionSort;
   (d) SelectionSort;
   (e) ShellSort;
2. $O(n \log(n))$;
   (a) HeapSort;
   (b) MergeSort;
   (c) QuickSort;
3. $O(nk)$;
   (a) RadixSort.

## 2. Tempo de Execução

### 2.1 $O(n^2)$

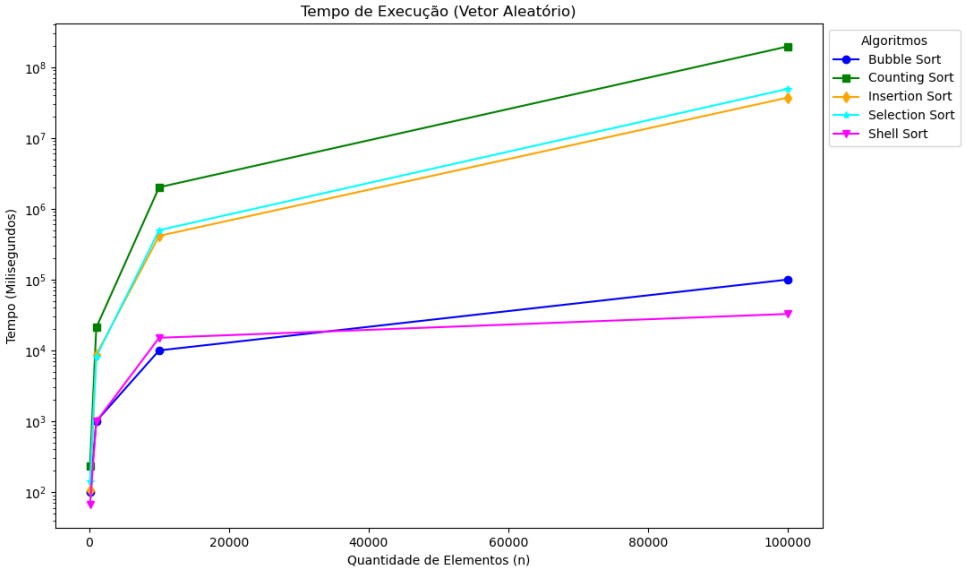


Figura 1: Gráfico do tempo de execução dos algoritmos $O(n^2)$

### 2.2 $O(n \log(n))$

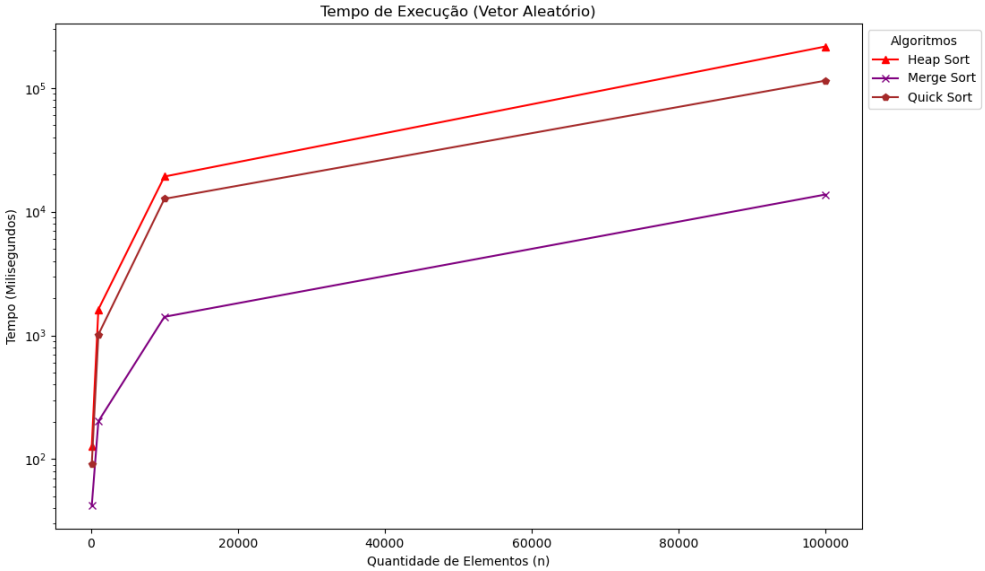


Figura 2: Gráfico do tempo de execução dos algoritmos $O(n \log(n))$

### 2.3 $O(nk)$

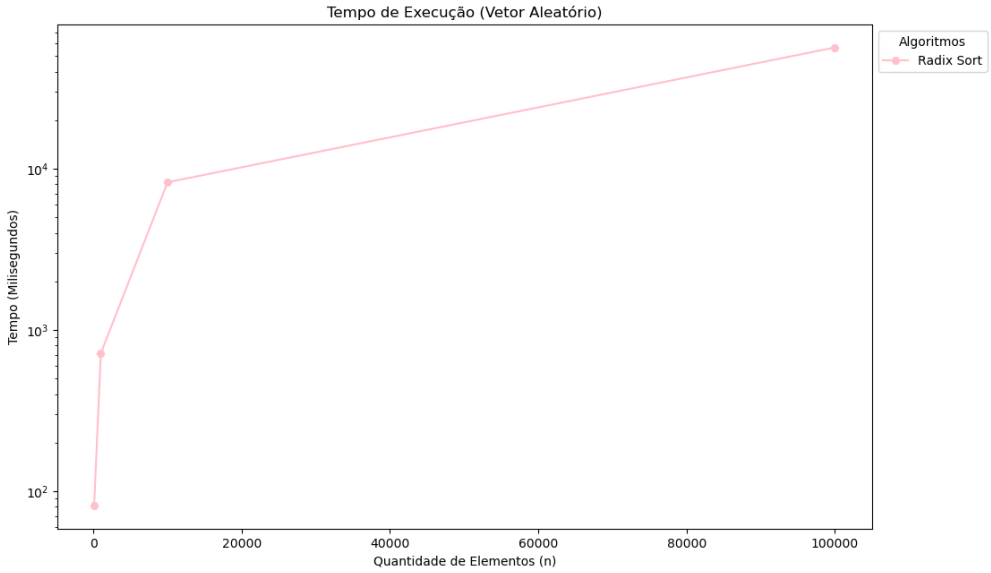


Figura 3: Gráfico do tempo de execução dos algoritmos $O(nk)$

## 3. Quantidade de Comparações

### 3.1 $O(n^2)$

Tabela 1: Tabela de comparações BubbleSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 99 | 4950 | 4882,1 |
| 1000 | 999 | 499500 | 498700,7 |
| 10000 | 9999 | 49995000 | 49987304,1 |
| 100000 | 99999 | 4999950000 | 4999798132 |

Tabela 2: Tabela de comparações Contagem de Menores

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 4950 | 4950 | 4950 |
| 1000 | 499500 | 499500 | 499500 |
| 10000 | 49995000 | 49995000 | 49995000 |
| 100000 | 4999950000 | 4999950000 | 4999950000 |

Tabela 3: Tabela de comparações InsertionSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 99 | 4950 | 2636,3 |
| 1000 | 999 | 499500 | 249352,9 |
| 10000 | 9999 | 49995000 | 24986288,7 |
| 100000 | 99999 | 4999950000 | 2498975066 |

Tabela 4: Tabela de comparações SelectionSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 4950 | 4950 | 4950 |
| 1000 | 499500 | 499500 | 499500 |
| 10000 | 49995000 | 49995000 | 49995000 |
| 100000 | 4999950000 | 4999950000 | 4999950000 |

Tabela 5: Tabela de comparações ShellSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 0 | 414 | 573,7 |
| 1000 | 0 | 5188 | 11295,2 |
| 10000 | 0 | 211340 | 252463,5 |
| 100000 | 0 | 18184250 | 11965729,8 |

**3.2** $O(n \log(n))$

Tabela 6: Tabela de comparações HeapSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 1380 | 1132 | 1262,2 |
| 1000 | 20416 | 17632 | 19143,2 |
| 10000 | 273912 | 243392 | 258448,2 |
| 100000 | 3401708 | 3094868 | 3249879,6 |

Tabela 7: Tabela de comparações MergeSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 99 | 102 | 188,9 |
| 1000 | 999 | 1001 | 1981,3 |
| 10000 | 9999 | 10005 | 19977,4 |
| 100000 | 99999 | 100006 | 199975,4 |

Tabela 8: Tabela de comparações QuickSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 756 | 1417 | 873,9 |
| 1000 | 10814 | 102188 | 12786,1 |
| 10000 | 141660 | 9744569 | 178437,5 |
| 100000 | 1747098 | 968505521 | 2232338,6 |

**3.3** $O(nk)$

Tabela 9: Tabela de comparações RadixSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 100 | 100 | 100 |
| 1000 | 1000 | 1000 | 1000 |
| 10000 | 10000 | 10000 | 10000 |
| 100000 | 100000 | 100000 | 100000 |

**4. Movimentações**

**4.1** $O(n^2)$

Tabela 10: Tabela de trocas BubbleSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 0 | 4950 | 2542,9 |
| 1000 | 0 | 499500 | 248360,4 |
| 10000 | 0 | 49995000 | 24976298,5 |
| 100000 | 0 | 4999950000 | 2271704616 |

Tabela 11: Tabela de trocas Contagem de Menores

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 100 | 100 | 100 |
| 1000 | 1000 | 1000 | 1000 |
| 10000 | 10000 | 10000 | 10000 |
| 100000 | 100000 | 100000 | 100000 |

Tabela 12: Tabela de trocas InsertionSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 99 | 5049 | 2641,9 |
| 1000 | 999 | 500499 | 249359,4 |
| 10000 | 9999 | 50004999 | 24986297,5 |
| 100000 | 99999 | 5000049999 | 2498975077 |

Tabela 13: Tabela de trocas SelectionSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 0 | 50 | 94,7 |
| 1000 | 0 | 500 | 993,6 |
| 10000 | 0 | 5000 | 9991,5 |
| 100000 | 0 | 50000 | 99987,8 |

Tabela 14: Tabela de trocas ShellSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 291 | 705 | 864,7 |
| 1000 | 4610 | 9798 | 15905,2 |
| 10000 | 49610 | 260950 | 302073,5 |
| 100000 | 499610 | 18683860 | 12465339,8 |

### 4.2 $O(n \log(n))$

Tabela 15: Tabela de trocas HeapSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 640 | 516 | 581,1 |
| 1000 | 9708 | 8316 | 9071,6 |
| 10000 | 131956 | 116696 | 124224,1 |
| 100000 | 1650854 | 1497434 | 1574939,8 |

Tabela 16: Tabela de trocas MergeSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 201 | 201 | 201 |
| 1000 | 2000 | 2000 | 2000 |
| 10000 | 20004 | 20004 | 20004 |
| 100000 | 200005 | 200005 | 200005 |

Tabela 17: Tabela de trocas QuickSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 424 | 1179 | 482 |
| 1000 | 5840 | 99206 | 6731,8 |
| 10000 | 75580 | 9709257 | 93351,2 |
| 100000 | 919107 | 968096337 | 1164047,7 |

### 4.3 $O(nk)$

Tabela 18: Tabela de trocas RadixSort

| N | Ordenado | Inverso | Aleatório |
|---|---|---|---|
| 100 | 300 | 300 | 300 |
| 1000 | 4000 | 4000 | 4000 |
| 10000 | 50000 | 50000 | 50000 |
| 100000 | 600000 | 600000 | 600000 |

**5. Análise dos Resultados**

Na necessidade de dicidir qual o melhor e o pior algoritmo, vamos nos aproximar o máximo possível de uma situação real, onde a propriedade que mais importa é o **tempo de execução**, que pode ser o gargalo para outros serviços, e não a quantidade de trocas ou comparações, tendo em vista a alta capacidade computacional das máquinas atuais. Além disso, o vetor que mais se aproxima de situações reais é um **aleatório**. Portanto, vamos analisar o gráfico a seguir.

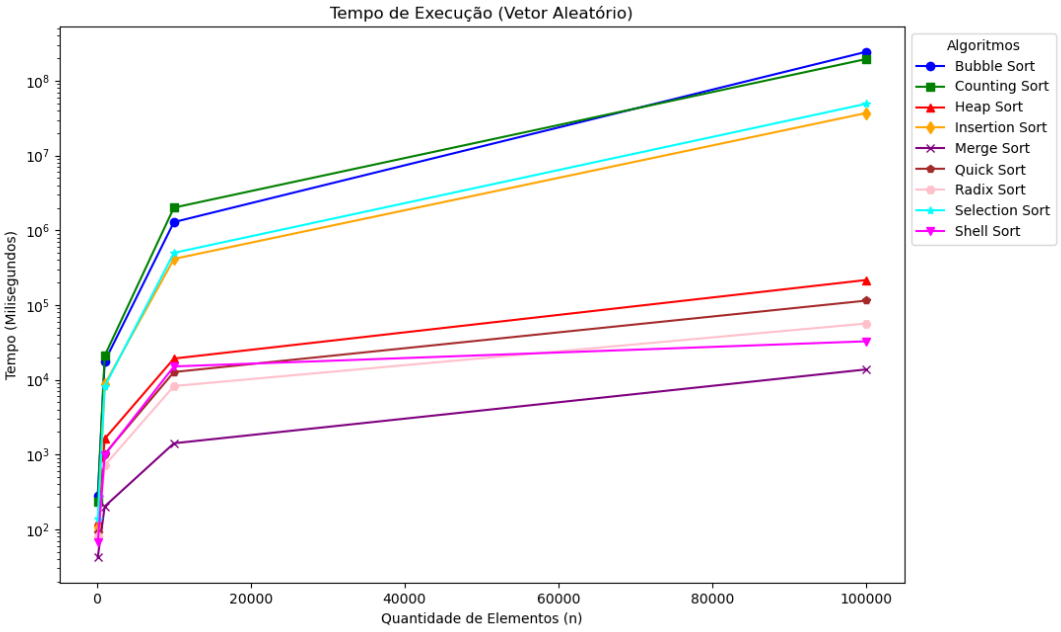


Figura 4: Gráfico do tempo de execução de todos algoritmos

Percebemos, no gráfico, que o melhor algoritmo foi o **MergeSort**, enquanto os piores foram o **BubbleSort** e o **Contagem de Menores**. Esse resultado faz sentido quando comparado com as complexidades dos algoritmos, visto que o MergeSort tem complexidade temporal $O(n \log(n))$. Note, contudo, que ele possui uma complexidade de espaço $O(n)$. Os piores algoritmos foram aqueles com complexidade temporal $O(n^2)$, com complexidades espaciais de $O(n)$.