Faculdade de Informática e Administração Paulista
FIAP Paulista

Henrique Martins Oliveira – RM563620
Henrique Texeira Cesar – RM563088

Computational Thinking Using Python

Análise de Desempenho de Algoritmos de Ordenação

São Paulo
2025

CONFIGURAÇÃO DA MÁQUINA UTILIZADA

- **Tipo:** Notebook Dell
- **Processador (CPU):** AMD Ryzen 5
- **Memória RAM:** 8 GB
- **Armazenamento (HD/SSD):** 256 GB SSD
- **Placa de Vídeo (GPU):** AMD Radeon Graphics

ALGORITMO BUBBLE SORT

1. ### BREVE DESCRIÇÃO

   O Bubble Sort é um algoritmo de ordenação que percorre repetidamente a lista, compara elementos um do lado do outro e os troca se estiverem na ordem errada. As passagens pela lista são repetidas até que nenhuma troca seja necessária. Possui uma complexidade de tempo média e de pior caso de $O(n^2)$, sendo ineficiente para grandes conjuntos de dados.

2. ### CÓDIGO DO ALGORITMO

```python
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        trocou = False
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                trocou = True
        if not trocou:
            break
    return lista
```

3. ### TABELA COM OS TEMPOS OBTIDOS

```
-=≡ Testando o Bubble Sort ≡=-
Testando com N = 1000...
  Amostra 1: 0.082 segundos
  Amostra 2: 0.080 segundos
  Amostra 3: 0.088 segundos
  Média de tempo: 0.083 segundos

Testando com N = 5000...
  Amostra 1: 2.014 segundos
  Amostra 2: 1.976 segundos
  Amostra 3: 2.084 segundos
  Média de tempo: 2.025 segundos

Testando com N = 10000...
  Amostra 1: 8.452 segundos
  Amostra 2: 7.991 segundos
  Amostra 3: 8.004 segundos
  Média de tempo: 8.149 segundos

Testando com N = 25000...
  Amostra 1: 51.582 segundos
  Amostra 2: 54.072 segundos
  Amostra 3: 53.417 segundos
  Média de tempo: 53.024 segundos

Testando com N = 50000...
  Amostra 1: 236.581 segundos
  Amostra 2: 240.480 segundos
  Amostra 3: 256.412 segundos
  Média de tempo: 244.491 segundos
```

5. ANÁLISE DA SIMULAÇÃO

A simulação do Bubble Sort demonstra ineficiência com o aumento do volume de dados. Conforme a maior quantidade de dados maior o tempo, confirmando que este algoritmo não é prático para aplicações que manipulam grandes quantidades de dados.

ALGORITMO SELECTION SORT

6. BREVE DESCRIÇÃO

O Selection Sort é um algoritmo que divide a lista em duas partes: uma sublista ordenada e uma sublista não ordenada. A cada iteração, ele encontra o menor elemento na sublista não ordenada e o move para o final da sublista ordenada. Sua complexidade de tempo é de $O(n^2)$, o que o torna inadequado para listas muito grandes.

7. CÓDIGO DO ALGORITMO

```python
def selection_sort(lista):
    n = len(lista)

    for i in range(n):
        indice_minimo = i

        for j in range(i + 1, n):
            if lista[j] < lista[indice_minimo]:
                    indice_minimo = j

        lista[i], lista[indice_minimo] = lista[indice_minimo], lista[i]
    return lista
```

8. TABELA COM OS TEMPOS OBTIDOS

```
-=≡ Testando o Selection Sort ≡=-
Testando com N = 1000...
  Amostra 1: 0.036 segundos
  Amostra 2: 0.023 segundos
  Amostra 3: 0.031 segundos
  Média de tempo: 0.030 segundos

Testando com N = 5000...
  Amostra 1: 0.608 segundos
  Amostra 2: 0.546 segundos
  Amostra 3: 0.571 segundos
  Média de tempo: 0.575 segundos

Testando com N = 10000...
  Amostra 1: 2.176 segundos
  Amostra 2: 2.234 segundos
  Amostra 3: 2.463 segundos
  Média de tempo: 2.291 segundos

Testando com N = 25000...
  Amostra 1: 14.535 segundos
  Amostra 2: 15.261 segundos
  Amostra 3: 14.511 segundos
  Média de tempo: 14.769 segundos

Testando com N = 50000...
  Amostra 1: 67.078 segundos
  Amostra 2: 75.701 segundos
  Amostra 3: 66.536 segundos
  Média de tempo: 69.772 segundos
```

9. ANÁLISE DA SIMULAÇÃO

Os testes com o Selection Sort revelam um crescimento gigante no tempo de processamento. Embora tenha se mostrado consistentemente mais rápido que o Bubble Sort, a escalabilidade ainda é um problema grave. O tempo para

ordenar 50.000 itens foi de mais de um minuto, ainda é considerado lento para grandes volumes de dados.

ALGORITMO INSERTION SORT

10. BREVE DESCRIÇÃO

O Insertion Sort constrói a lista ordenada final um item de cada vez. Ele itera sobre a lista de entrada, e a cada iteração, remove um elemento e o insere na posição correta da parte já ordenada da lista. A complexidade de tempo média é $O(n^2)$, ele é eficiente para conjuntos de dados pequenos e para listas que já estão quase ordenadas.

11. CÓDIGO DO ALGORITMO

```python
def insertion_sort(lista):
    for i in range(1, len(lista)):
        chave = lista[i]
        j = i - 1
        while j >= 0 and chave < lista[j]:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = chave
    return lista
```

12. TABELA COM OS TEMPOS OBTIDOS

```
-=≡ Testando o Insertion Sort ≡=-
Testando com N = 1000...
  Amostra 1: 0.048 segundos
  Amostra 2: 0.037 segundos
  Amostra 3: 0.043 segundos
  Média de tempo: 0.043 segundos

Testando com N = 5000...
  Amostra 1: 1.010 segundos
  Amostra 2: 0.973 segundos
  Amostra 3: 0.960 segundos
  Média de tempo: 0.981 segundos

Testando com N = 10000...
  Amostra 1: 4.805 segundos
  Amostra 2: 4.151 segundos
  Amostra 3: 4.925 segundos
  Média de tempo: 4.627 segundos

Testando com N = 25000...
  Amostra 1: 26.833 segundos
  Amostra 2: 31.780 segundos
  Amostra 3: 27.987 segundos
  Média de tempo: 28.867 segundos

Testando com N = 50000...
  Amostra 1: 133.084 segundos
  Amostra 2: 113.384 segundos
  Amostra 3: 138.174 segundos
  Média de tempo: 128.214 segundos
```

13. ANÁLISE DA SIMULAÇÃO

O Insertion Sort apresentou um desempenho intermediário entre o Selection Sort e o Bubble Sort em nossos testes para listas maiores. Com um tempo médio de mais de dois minutos para ordenar 50.000 números, o algoritmo confirma sua inadequação para dados de grande porte.

ALGORITMO MERGE SORT

14. BREVE DESCRIÇÃO

O Merge Sort é um algoritmo de ordenação eficiente. Ele divide a lista ao meio até que cada sublista contenha um único elemento. Em seguida, combina essas sublistas de forma ordenada até que a lista inteira esteja ordenada. Sua grande vantagem é a complexidade de tempo de $O(n \log n)$ em todos os casos (pior, médio e melhor).

15. CÓDIGO DO ALGORITMO

```python
def merge_sort(lista):
    if len(lista) > 1:

        meio = len(lista) // 2

        L = lista[:meio]
        R = lista[meio:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                lista[k] = L[i]
                i += 1
            else:
                lista[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            lista[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            lista[k] = R[j]
            j += 1
            k += 1
    return lista
```

16. TABELA COM OS TEMPOS OBTIDOS

```
-=≡ Testando o Merge Sort ≡=-
Testando com N = 1000...
  Amostra 1: 0.007 segundos
  Amostra 2: 0.004 segundos
  Amostra 3: 0.004 segundos
  Média de tempo: 0.005 segundos

Testando com N = 5000...
  Amostra 1: 0.030 segundos
  Amostra 2: 0.024 segundos
  Amostra 3: 0.018 segundos
  Média de tempo: 0.024 segundos

Testando com N = 10000...
  Amostra 1: 0.044 segundos
  Amostra 2: 0.044 segundos
  Amostra 3: 0.048 segundos
  Média de tempo: 0.045 segundos

Testando com N = 25000...
  Amostra 1: 0.117 segundos
  Amostra 2: 0.105 segundos
  Amostra 3: 0.115 segundos
  Média de tempo: 0.112 segundos

Testando com N = 50000...
  Amostra 1: 0.222 segundos
  Amostra 2: 0.231 segundos
  Amostra 3: 0.237 segundos
  Média de tempo: 0.230 segundos
```

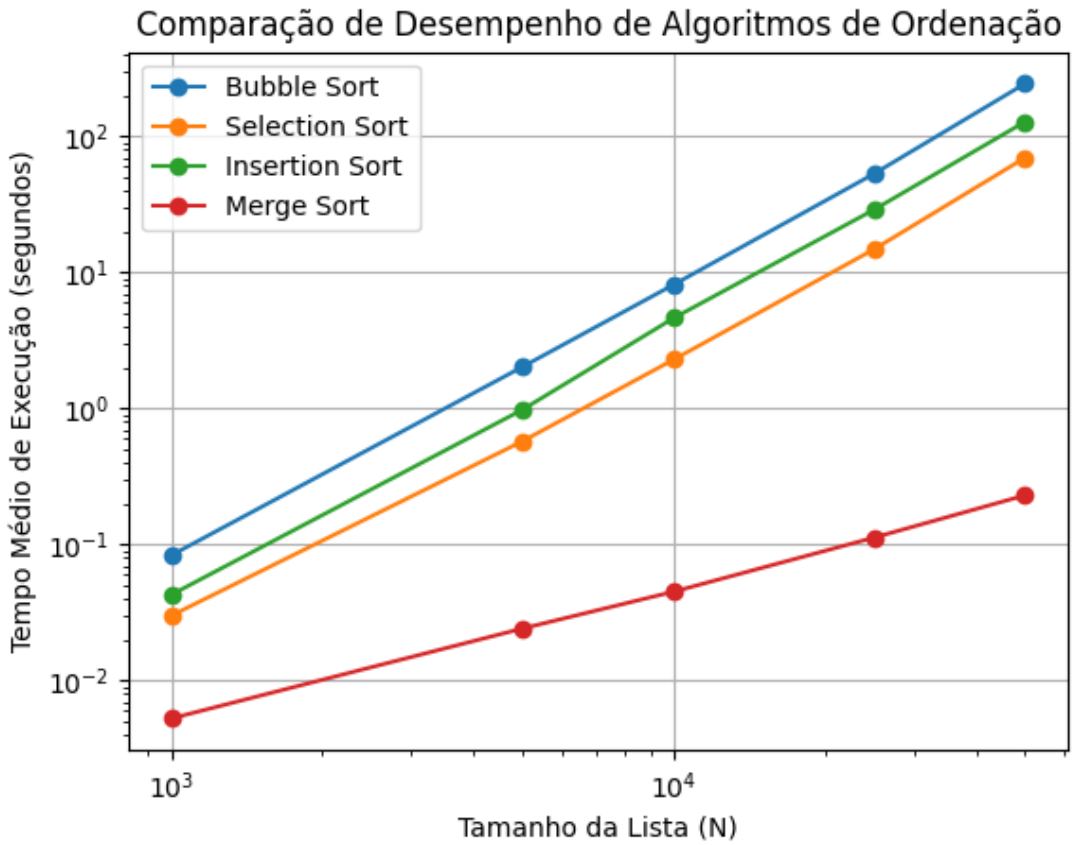
17. ANÁLISE DA SIMULAÇÃO

O Merge Sort é superior em relação aos outros algoritmos. O tempo de execução para ordenar 50.000 elementos foi de apenas 0.230 segundos, O crescimento do tempo é muito mais controlado, provando que sua complexidade $O(n \log n)$ o torna uma escolha extremamente eficiente e escalável para grandes volumes de dados.

GRÁFICOS COMPARATIVOS

O gráfico abaixo mostra a diferença de desempenho entre os algoritmos. Ele utiliza uma escala entre o Tamanho da Lista e Tempo de Execução para permitir a visualização de todos os algoritmos na mesma imagem.

A linha do Merge Sort (vermelha) se destaca na parte inferior do gráfico, mostrando um crescimento lento e controlado. Em contraste, as linhas do Bubble Sort, Selection Sort e Insertion Sort sobem drasticamente, confirmando visualmente seu crescimento quadrático e sua ineficiência para listas maiores.



CONCLUSÃO

Os algoritmos com complexidade $O(n^2)$ (Bubble, Selection e Insertion Sort) mostraram-se extremamente lentos, com tempos de execução que crescem exponencialmente com o tamanho da entrada, tornando-os inviáveis para lidar com grandes volumes de informação.

Por outro lado, o Merge Sort, com sua complexidade $O(n \log n)$, demonstrou uma performance e escalabilidade superior. Seu tempo de execução, mesmo para a maior lista de 50.000 elementos, foi inferior a um segundo, destacando-o como uma solução robusta e eficiente.

A conclusão final é que a escolha do algoritmo correto tem um impacto fundamental no desempenho de um software, e a análise de complexidade é uma ferramenta essencial para prever o comportamento de um sistema à medida que os dados aumentam.