Relatório Técnico: Aprofundamento Teórico Comparativo de Algoritmos de Ordenação

Disciplina: Estrutura de Dados

Autores: * Alexsander R. dos Santos * Lucas Silva * Pedro Babier

Data: 14 de Outubro de 2025

---

1. Introdução

Este relatório técnico tem como objetivo apresentar uma análise comparativa dos seis principais algoritmos de ordenação baseados em comparação: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort e Heap Sort. A comparação será fundamentada na **Complexidade de Tempo** (Time Complexity) em seus cenários de Melhor Caso (Best Case), Caso Médio (Average Case) e Pior Caso (Worst Case), conforme a notação Big O. A eficiência de um algoritmo é crucial para o desenvolvimento de sistemas escaláveis, e a compreensão de como diferentes arranjos de entrada afetam o desempenho é fundamental [1].

2. Complexidade de Tempo dos Algoritmos de Ordenação

A Complexidade de Tempo, expressa pela notação Big O, descreve o crescimento do tempo de execução de um algoritmo em função do tamanho da entrada (n). Para algoritmos de ordenação, a análise é tipicamente dividida em três cenários:

- **Melhor Caso (Best Case):** O cenário de entrada que resulta no menor tempo de execução.

- **Caso Médio (Average Case):** O tempo de execução esperado para uma entrada aleatória.
- **Pior Caso (Worst Case):** O cenário de entrada que resulta no maior tempo de execução.

A tabela a seguir resume a complexidade de tempo para os seis algoritmos analisados [1]:

Algoritmo	Melhor Caso (Best Case)	Caso Médio (Average Case)	Pior Caso (Worst Case)
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

3. Análise dos Cenários de Entrada

A variação na complexidade de tempo para cada algoritmo é determinada pela forma como a entrada (o vetor a ser ordenado) está organizada.

3.1. Algoritmos Quadráticos ($O(n^2)$)

Bubble Sort

- **Melhor Caso ($O(n)$):** Ocorre quando o vetor já está **completamente ordenado**. O algoritmo realiza apenas uma passagem completa pelo vetor, detectando que nenhuma troca é necessária e encerrando a execução prematuramente.
- **Pior Caso ($O(n^2)$):** Ocorre quando o vetor está **ordenado em ordem inversa**. Cada elemento precisa ser trocado com o seu vizinho em quase todas as

iterações, maximizando o número de comparações e trocas.

Selection Sort

- **Melhor, Médio e Pior Caso ($O(n^2)$):** O Selection Sort sempre realiza o mesmo número de comparações, independentemente da ordem inicial do vetor. Em cada iteração, ele deve percorrer o restante do vetor para encontrar o menor elemento. Embora o número de **trocas** seja minimizado (apenas uma por iteração), o número de **comparações** permanece constante, resultando em $O(n^2)$ em todos os cenários.

Insertion Sort

- **Melhor Caso ($O(n)$):** Ocorre quando o vetor já está **completamente ordenado**. O algoritmo percorre o vetor, e em cada passo, o elemento atual é comparado apenas com o seu vizinho anterior, pois já está na posição correta, minimizando as comparações e deslocamentos.
- **Pior Caso ($O(n^2)$):** Ocorre quando o vetor está **ordenado em ordem inversa**. Cada elemento inserido deve ser comparado e deslocado até o início da sublista já ordenada, maximizando o número de operações.

3.2. Algoritmos Eficientes ($O(n \log n)$)

Merge Sort

- **Melhor, Médio e Pior Caso ($O(n \log n)$):** O Merge Sort é um algoritmo de divisão e conquista que divide o vetor em metades recursivamente e depois as combina (merge) de forma ordenada. A complexidade é sempre $O(n \log n)$ porque o processo de divisão e a subsequente fusão (merge) de subvetores requerem um número previsível de operações, independentemente da ordem inicial dos dados.

Heap Sort

- **Melhor, Médio e Pior Caso ($O(n \log n)$):** O Heap Sort constrói uma estrutura de dados *heap* (montículo) e extrai o elemento máximo (ou mínimo) repetidamente. A construção inicial do *heap* leva $O(n)$, e cada uma das n extrações leva $O(\log n)$. Portanto, a complexidade total é $O(n + n \log n)$, que simplifica para

$O(n \log n)$ em todos os casos, pois a estrutura do *heap* garante um desempenho consistente.

Quick Sort

- **Melhor e Caso Médio ($O(n \log n)$):** O Quick Sort também é um algoritmo de divisão e conquista, mas seu desempenho depende da escolha do **pivô**. O Melhor Caso e o Caso Médio ocorrem quando o pivô escolhido divide o vetor em duas sublistas de tamanhos aproximadamente iguais. Isso garante que a profundidade da recursão seja $\log n$, e o trabalho em cada nível seja $O(n)$, resultando em $O(n \log n)$.
- **Pior Caso ($O(n^2)$):** Ocorre quando o pivô escolhido é sempre o menor ou o maior elemento do vetor. Isso acontece, por exemplo, quando o vetor já está **ordenado** ou **ordenado em ordem inversa** e o pivô é sempre o primeiro ou o último elemento. Nesse cenário, o particionamento resulta em uma sublista de tamanho $n - 1$ e outra de tamanho 0, levando a uma profundidade de recursão de n, e a complexidade degenera para $O(n^2)$.

4. Conclusão

A análise comparativa demonstra que os algoritmos de ordenação podem ser divididos em duas categorias principais de eficiência assintótica: os algoritmos quadráticos ($O(n^2)$) e os algoritmos eficientes ($O(n \log n)$).

Categoria	Algoritmos	Cenário de Uso Recomendado
Quadrático	Bubble, Selection, Insertion	Vetores muito pequenos ou vetores quase ordenados (para Bubble e Insertion).
Eficiente	Merge, Quick, Heap	Vetores grandes, onde a velocidade de processamento é crítica.

O **Merge Sort** e o **Heap Sort** oferecem a garantia de $O(n \log n)$ em todos os casos, sendo ideais para aplicações que exigem desempenho previsível. O **Quick Sort** é frequentemente o mais rápido na prática (devido a fatores como localidade de referência), mas sua complexidade de Pior Caso $O(n^2)$ exige cuidado na seleção do pivô.

---

5. Referências

[1] GeeksforGeeks. *Time Complexities of all Sorting Algorithms*. Disponível em: https://www.geeksforgeeks.org/dsa/time-complexities-of-all-sorting-algorithms/. Acesso em: 14 out. 2025.