# Explorando a Eficiência dos Algoritmos de Ordenação: Avaliação Teórica e Experimental de Desempenho

**Thiago Henrique Gomes Feliciano**

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUC-Minas)
Belo Horizonte – MG – Brasil

***Resumo.*** *Este artigo apresenta uma análise comparativa entre quatro algoritmos de ordenação: Selection Sort, Insertion Sort, Bubble Sort e Quicksort. A avaliação foi realizada utilizando vetores de números inteiros gerados aleatoriamente com tamanhos variando entre 100, 1000, 10000 e 100000 elementos. Os algoritmos foram implementados e executados, e os resultados foram comparados com base no tempo de execução, número de comparações e movimentações realizadas. Os resultados confirmam a ineficiência dos algoritmos quadráticos para grandes volumes de dados, enquanto o Quicksort demonstrou desempenho muito melhor.*

***Abstract.*** *This paper presents a comparative analysis between four sorting algorithms: Selection Sort, Insertion Sort, Bubble Sort, and Quicksort. The evaluation was carried out using randomly generated integer vectors with sizes ranging from 100, 1000, 10000 and 100000 elements. The algorithms were implemented and executed, and the results were compared based on the execution time, number of comparisons, and movements performed. The results confirm the inefficiency of quadratic algorithms for large volumes of data, while Quicksort demonstrated much better performance..*

## 1. Introdução

A ordenação desempenha um papel fundamental na ciência da computação, sendo essencial para a eficácia e o funcionamento de diversos algoritmos e sistemas. Em muitos casos, ela é um pré-requisito para operações como busca binária, análise de dados, algoritmos gráficos e árvores balanceadas. Diante disso, é fundamental que um profissional da área compreenda os diferentes algoritmos de ordenação, suas complexidades e os contextos em que cada um é mais apropriado.

Algoritmos como Selection Sort, Insertion Sort, Bubble Sort e Quick Sort estão entre os mais clássicos e amplamente utilizados na computação. Neste artigo, analisaremos a complexidade desses algoritmos a partir de métricas como tempo de execução, número de movimentações e comparações realizadas. Dessa forma, será possível comparar o desempenho de cada um em cenários com diferentes quantidades de dados a serem ordenados.

## 2. Fundamentação dos Métodos de Ordenação

### 2.1. Selection Sort

O Selection Sort é um dos algoritmos de ordenação mais simples. Ele funciona iterando sobre a lista e, a cada iteração, selecionando o menor elemento e trocando-o com o elemento da posição atual. O processo é repetido até que a lista esteja completamente ordenada.

```java
public static void selectsort(int[] array, int n) {
    for (int i = 0; i < n - 1; i++) {
        int menor = i;
        for (int j = i + 1; j < n; j++) {
            if (array[menor] > array[j]) {
                menor = j;
            }
        }
        swap(array, menor, i);
    }
}

public static void swap(int[] array, int a, int b) {
    int temp = array[a];
    array[a] = array[b];
    array[b] = temp;
}
```

**Figure 1. Implementação do Select Sort**

O Selection Sort possui complexidade de O(n²) para todos os casos.

## 2.2. Insertion Sort

```java
public static void insertionsort(int[] array, int n) {
    for (int i = 1; i < n; i++) {
        int temp = array[i];
        int j = i - 1;
        while ((j >= 0) && (array[j] > temp)) {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = temp;
    }
```

**Figure 2. Implementação do Insertion Sort**

O Insertion Sort é um algoritmo de ordenação que organiza os elementos de uma lista criando uma sequência ordenada aos poucos. A cada passo, ele seleciona um elemento da lista e o insere na posição correta entre os elementos já ordenados. Isso se repete até que toda a lista esteja ordenada. O Insertion Sort tem complexidade O(n²) no pior caso (quando a lista está decrescente). No melhor caso, quando a lista já está ordenada, a complexidade é O(n), pois ele só percorre a lista sem fazer trocas.

## 2.3. Bubble Sort

```java
public static void bubblesort(int[] array, int n) {
    for (int i = n - 1; i > 0; i--) {
        for (int j = 0; j < i; j++) {
            if (array[j] > array[j + 1]) {
                int aux = array[j];
                array[j] = array[j + 1];
                array[j + 1] = aux;
            }
        }
    }
}
```

**Figure 3. Implementação do Bubble Sort**

O Bubble Sort é um algoritmo simples que percorre a lista várias vezes, comparando elementos adjacentes e trocando-os de posição se estiverem fora de ordem. Isso se repete até que toda a lista esteja ordenada. O Bubble Sort tem complexidade de O(n²) no pior caso.

### 2.4. Quicksort

O Quicksort é um algoritmo de ordenação baseado no paradigma de dividir e conquistar. Ele escolhe um elemento do vetor (chamado de pivô) e divide a lista em duas partes: uma com elementos menores que o pivô e outra com elementos maiores. Esses subvetores são então ordenados recursivamente. O Quicksort tem uma complexidade média de O(n log n). No pior caso, quando o pivô é o menor ou o maior elemento, a complexidade pode ser O($n^2$).

```java
public static void quicksort(int[] array, int esq, int dir) {
    int i = esq, j = dir, pivo = array[(esq + dir) / 2];

    while (i <= j) {
        while (array[i] < pivo) {
            i++;
        }
        while (array[j] > pivo) {
            j--;
        }
        if (i <= j) {
            swap(array, i, j);
            i++;
            j--;
        }
    }

    if (esq < j) {
        quicksort(array, esq, j);
    }
    if (i < dir) {
        quicksort(array, i, dir);
    }
}

public static void swap(int[] array, int a, int b) {
  int temp = array[a];
  array[a] = array[b];
  array[b] = temp;
}
```

**Figure 4. Implementação do Quicksort**

## 3. Metodologia

Os testes de desempenho dos algoritmos foram realizados em um notebook Samsung 550XDA, utilizando o sistema operacional Ubuntu 24.04.2 LTS Desktop (amd64). O processador da máquina é um Intel Core i3 de 11ª geração, com frequência de aproximadamente 2.99 GHz. O equipamento possui 4 GB de memória RAM, o que limita a execução eficiente de algoritmos com grandes volumes de dados. A arquitetura do sistema é 64 bits, e os testes foram feitos diretamente no terminal, utilizando o Vim para desenvolvimento e execução dos códigos. Todos os algoritmos foram implementados utilizando a linguagem de programação Java. Cada algoritmo foi testado com quatro vetores contendo 100, 1.000, 10.000 e 100.000 inteiros aleatórios. Os critérios de avaliação utilizados serão o tempo de execução (em milissegundos), número de comparações e movimentações. Cada algoritmo será testado 30 vezes com cada tamanho de vetor, e o resultado final será a média simples entre os resultados de cada execução para cada tamanho de vetor.

## 4. Resultados e Discussão

Com os gráficos a seguir sobre o tempo de execução, percebe-se que o Bubble Sort apresenta a curva mais acentuada, indicando que é o algoritmo que mais demora para ser executado. Isso pode ser relacionado à sua complexidade de $O(n^2)$. Por outro lado, o Quicksort, o mais eficiente em termos de tempo de execução, tem uma curva imperceptível quando comparada às dos outros algoritmos.

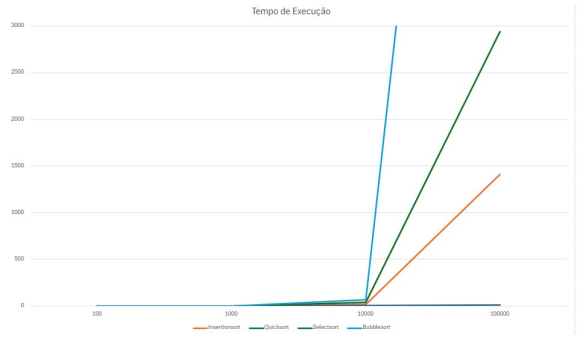


**Figure 5. Gráfico do Tempo de Execução**

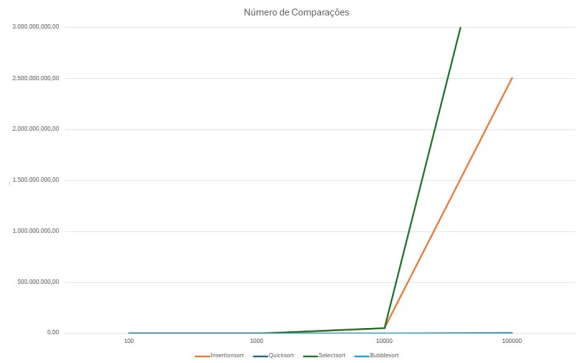


**Figure 6. Gráfico do Número de Comparações**

Com a análise dos gráficos e dos resultados, percebe-se que os algoritmos Insertion Sort e Selection Sort realizam o maior número de comparações, o que é evidente pelas suas curvas acentuadas. Por outro lado, as curvas do Quicksort e do Bubble Sort mal aparecem no gráfico, indicando que esses algoritmos realizaram um número muito menor de comparações.

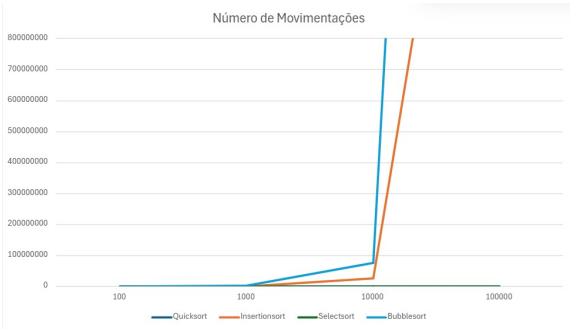


**Figure 7. Gráfico do Número de Movimentações**

Com a análise dos resultados e dos gráficos, é visível que o Bubble Sort e o Insertion Sort realizam um número de movimentações significativamente maior do que o Selection Sort e o Quicksort. As curvas destes últimos mal aparecem no gráfico, evidenciando a menor quantidade de movimentações realizadas por esses algoritmos.

## 5. Conclusão

Com as análises realizadas, percebe-se que, no quesito tempo de execução, o Quicksort apresenta uma eficiência significativamente superior. Em relação ao número de comparações, tanto o Quicksort quanto o Bubble Sort demonstraram melhor desempenho. Já no critério de movimentações, o Selection Sort e o Quicksort se destacaram pela menor quantidade de operações.

Dessa forma, conclui-se que o Quicksort apresenta a melhor eficiência geral em comparação com os demais algoritmos de ordenação analisados, o que é coerente com sua complexidade média de O(nlogn). Para trabalhos futuros, seria interessante estender a análise para outros algoritmos com complexidade O(nlogn), como Merge Sort e Heap Sort, a fim de comparar de forma mais abrangente o desempenho em termos de tempo de execução, número de comparações e movimentações.

## 6. References

Alexandre da Silva Pedroso and Fausto Gonçalves Cintra. *Estudo Analítico do Desempenho de Algoritmos de Ordenação*. 2024.