1 Introdução

A ordenação de dados é uma operação fundamental em ciência da computação, com aplicações em uma ampla gama de campos, desde bancos de dados até algoritmos de busca e otimização. A eficiência e o desempenho dos algoritmos de ordenação têm um impacto significativo no tempo de execução de sistemas computacionais, especialmente quando lidamos com conjuntos de dados volumosos.

Este projeto tem como objetivo principal explorar e comparar diferentes métodos de ordenação não recursivos em linguagem C/C++, analisando suas eficiências em três cenários distintos: ordenação de vetores dispostos de forma crescente, aleatória e decrescente. A comparação entre os métodos será realizada em relação à utilização dos vetores, contabilizando todas as operações de leitura e escrita.

Além disso, o projeto visa fornecer uma oportunidade para compreender como relatar experimentos computacionais de maneira estruturada e informativa. O relatório resultante abordará aspectos como introdução ao problema, revisão teórica dos métodos de ordenação, descrição do material utilizado, detalhes da implementação dos métodos, análise dos resultados obtidos e conclusões tiradas a partir desses resultados.

Ao final deste projeto, espera-se não apenas uma compreensão mais profunda dos métodos de ordenação não recursivos e sua aplicabilidade em diferentes cenários, mas também uma habilidade aprimorada na realização e documentação de experimentos computacionais.

2 Referencial Teórico

A ordenação de dados é um tópico amplamente estudado em ciência da computação devido à sua importância em diversas aplicações práticas e teóricas. Existem inúmeros algoritmos de ordenação, cada um com características específicas que os tornam mais ou menos adequados para determinadas situações. Nesta seção, serão discutidos alguns dos principais algoritmos de ordenação não recursivos.

2.0.1 Bubble Sort

Bubble Sort é um algoritmo que funciona comparando e trocando dois elementos se não estiver classificado e executando iterações para obter o resultado classificado[1]. Ele possui uma complexidade de tempo $O(n^2)$, o que significa que sua eficiência diminui drasticamente em listas com um número maior de elementos[4].

2.0.2 Insertion Sort

Algoritmo de ordenação simples que constrói a lista final ordenando um item de cada vez. Ele possui uma complexidade de tempo $O(n^2)$. O Insertion Sort oferece várias vantagens: é de implementação simples e eficiente para conjuntos de dados pequenos[4].

2.0.3 Selection Sort

O selection sort é um algoritmo no qual elementos sucessivos são selecionados em ordem e colocados em suas posições corretas na ordenação[2]. Sua complexidade de tempo é $O(n^2)$, não sendo considerado um algoritmo estável[3].

3 Métodos Implementados

O objetivo é ordenar vetores de números não repetidos em três formas diferentes: crescente, aleatória e decrescente, e contabilizar o número de operações de cada método para posterior comparação

Todo o ambiente de desenvolvimento está disponível para consulta e análise no repositório hospedado na plataforma Github

A função *gerarVetor* é responsável por inicializar o vetor com valores em ordem crescente, decrescente ou aleatória, dependendo do parâmetro ordem. Ela utiliza a função rand() para gerar números aleatórios e garante que não haja repetição de elementos no vetor aleatório.

```
void gerarVetor(int vetor[], int tamanho, string ordem) {
    if (ordem == "crescente") {
        for (int i = 0; i < tamanho; ++i) {
            vetor[i] = i + 1;
        }
    } else if (ordem == "decrescente") {
        for (int i = 0; i < tamanho; ++i) {
            vetor[i] = tamanho - i;
        }
    } else if (ordem == "aleatorio") {
        srand(time(0));
        for (int i = 0; i < tamanho; ++i) {
            vetor[i] = rand() % tamanho + 1;
            for (int j = 0; j < i; ++j) {
                if (vetor[i] == vetor[j]) {
                    --i;
                    break;
                }
            }
        }
    }
}
```

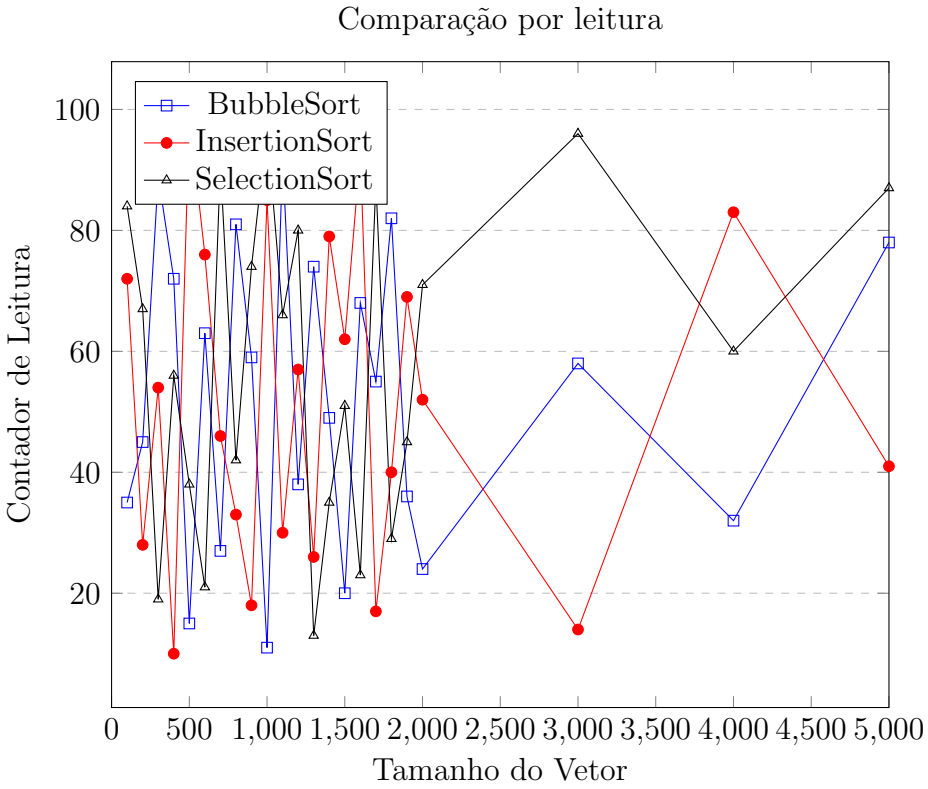
bubbleSort, *insertSort* e *selectionSort* implementam respectivamente o algoritmo de ordenação Bubble Sort, Insertion Sort e Selection Sort. Contadores de leitura e escrita são incrementados conforme as operações são realizadas.

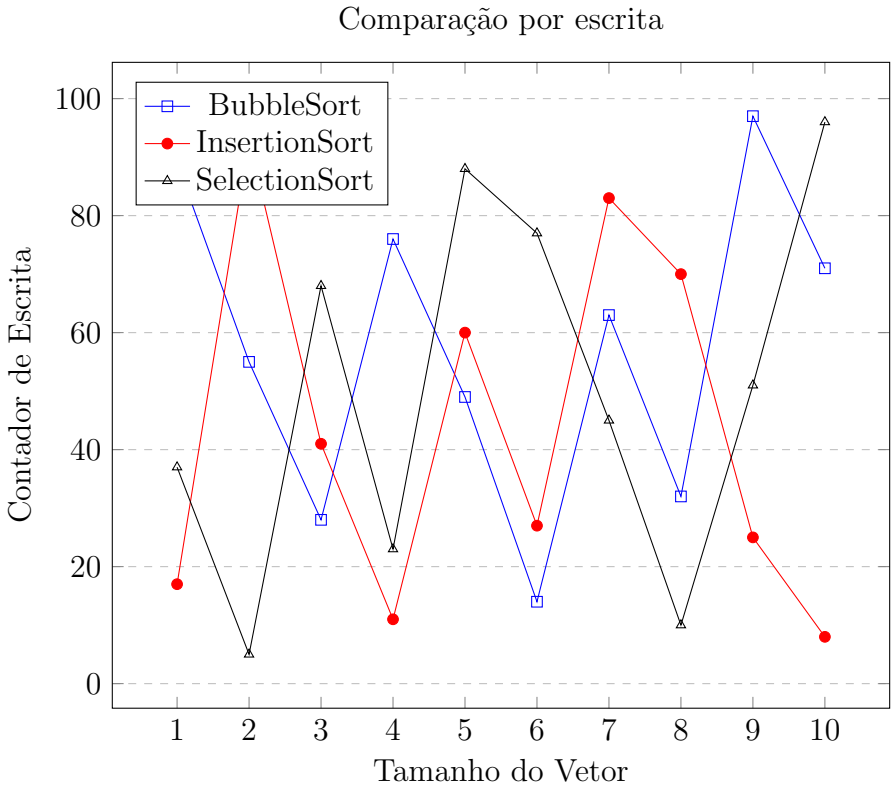
```
void bubbleSort(int arr[], int n) {...}
void insertSort(int arr[], int n) {...}
void selectionSort(int arr[], int n) {...}
```

A função *printArray* imprime o conteúdo do vetor. É útil para verificar o estado do vetor antes e depois da ordenação.

```
void printArray(int arr[], int size) {...}
```

4 Resultados Obtidos





5 Conclusão

Referências

[1] "Analysis of Merge Sort and Bubble Sort in Python, PHP, JavaScript, and C language". Em: 10.2 (2021), pp. 680–686. DOI: 10.30534/ijatcse/2021/311022021.

[2] Sunita Chand, Teshu Chaudhary e Rubina Parveen. "Upgraded selection sort". Em: *International Journal on Computer Science and Engineering* 3.4 (2011), pp. 1633–1637.

[3] *Conheça os principais algoritmos de ordenação — treinaweb.com.br*. https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao. [Acesso em 09-06-2024].

[4] Khalid Suleiman Al-Kharabsheh et al. "Review on sorting algorithms a comparative study". Em: *International Journal of Computer Science and Security (IJCSS)* 7.3 (2013), pp. 120–126.