UNIVERSIDADE FEDERAL DE ALFENAS

Aluno: Thiago Martins da Silva - RA:2024.1.08.023

Professor: Paulo Alexandre Bressan

AEDs I - Prática

16 de junho de 2024

INTRODUÇÃO

Comparação entre os Métodos de Ordenação

Neste relatório foi realizado um projeto para comparar a eficiência dos 3 métodos de ordenação: Bubble Sort, Insertion Sort e Selection Sort. Os executando em vetores de números aleatórios, crescentes e decrescentes, além de variando o tamanho dos vetores da seguinte forma: De 100 em 100 até atingir 1000, de 1000 em 1000 até atingir 10000 e de 10000 em 10000 até atingir 100000. De modo a cobrir um maior espaço amostral.

REFERENCIAL TEÓRICO

BubbleSort

- **Complexidade:** O(n^2) no pior caso e caso médio.
- **Estabilidade:** Estável.
- **Adaptação:** Não adaptativo.
- **Aplicabilidade:** Adequado para pequenos conjuntos de dados onde a simplicidade é prioritária.

InsertionSort

- **Complexidade:** O(n^2) no pior caso e caso médio, mas pode ser O(n) em ordens parciais.
- **Estabilidade:** Estável.
- **Adaptação:** Adaptativo para ordens parciais.
- **Aplicabilidade:** Eficiente para pequenos conjuntos de dados e ordens parciais.

SelectionSort

- **Complexidade:** O(n^2) no pior caso e caso médio.
- **Estabilidade:** Não estável.
- **Adaptação:** Não adaptativo.
- **Aplicabilidade:** Simples de implementar, mas menos eficiente para grandes conjuntos de dados.

Comparação Geral

- **Eficiência:** Bubble Sort e Insertion Sort são mais eficientes que SelectionSort. Insertion Sort é preferível em ordens parciais.
- **Estabilidade:** Bubble Sort e Insertion Sort são estáveis.
- **Adaptação:** Apenas Insertion Sort é adaptativo.

MATERIAL UTILIZADO

O projeto foi realizado por meio da IDE Visual Studio Code, utilizando a linguagem C++, onde foram implementadas funções contendo os três algoritmos de ordenação, bem como funções para gerar vetores aleatórios, crescentes e decrescentes. A fim de obter os resultados para elaborar um gráfico que destacasse a eficiência dos algoritmos, utilizou-se mais 3 funções para testar cada um dos métodos com todo o espaço amostral necessário, registrando sempre esses resultados em arquivos externos por meio da biblioteca fstream. Os gráficos, para realizar uma apresentação mais visual dos resultados, foram feitos plotando os resultados obtidos no programa Google Sheets.

MÉTODOS IMPLEMENTADOS

Bubble Sort: Consiste em percorrer o vetor várias vezes, levando o maior número a última posição do vetor em cada execução.

Implementação em C/C++:

```
// Algoritmo BubbleSort implementado.
void bubblesort(int v[], int size, long long &readCount, long long &writeCount){
    int i,j,aux;

    for(i = 0; i < size - 1; i++){
        for(j = 0; j < size - 1 - i; j++){
            ++readCount;
            if(v[j] > v[j+1]){
                aux = v[j+1];
                v[j+1] = v[j];
                v[j] = aux;
                writeCount+= 3;
            }
        }
    }
}
```

Insertion Sort: Consiste percorrer o vetor a fim de encontrar um algarismo desordenado e depois realizar comparações adjacentes entre eles para finalizar a ordenação.

Implementação em C/C++:

```
// Algoritmo InsertionSort implementado.
void insertionsort(int v[], int size, long long &readCount, long long &writeCount){
    int i, j, aux;

    for (i = 1; i < size; i++){
        aux = v[i];
        j = i - 1;
        ++writeCount;

        while (j >= 0 && aux < v[j])
        {
            v[j + 1] = v[j];
            writeCount++;
            readCount++;
            j--;
        }

        v[j + 1] = aux;
        writeCount++;
    }
}
```

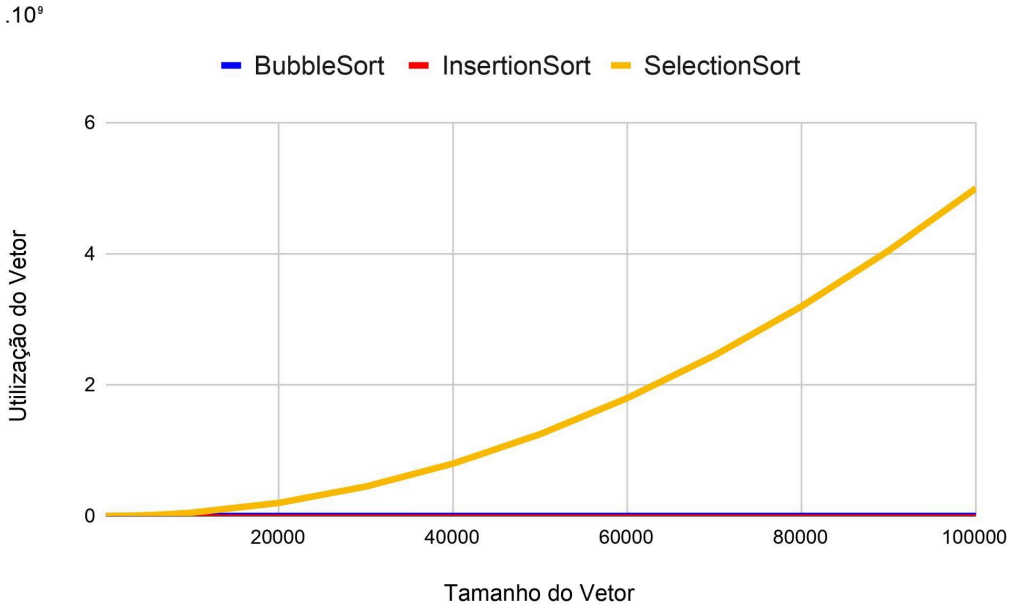
Selection Sort: Consiste em percorrer o vetor várias vezes. Em cada passagem pelo vetor, ele busca o menor elemento restante e o coloca na posição correta, realizando a ordenação.

Implementação em C/C++:

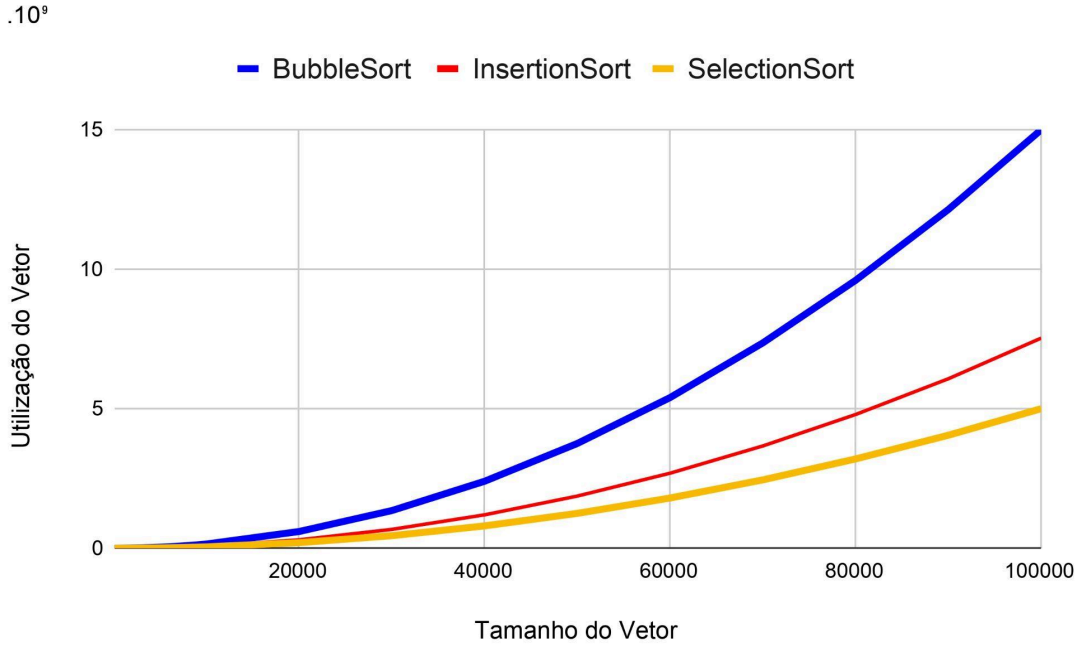
```
// Algoritmo SelectionSort implementado.
void selectionsort(int v[], int size, long long &readCount, long long &writeCount){
    int i, j, min, aux;

    for(i = 0; i < size - 1; i++){
        min = i;
        for(j = i + 1; j < size; j++){
            readCount++;
            if(v[j] < v[min]){
                min = j;
            }
        }
        if(min != i){
            aux = v[i];
            v[i] = v[min];
            v[min] = aux;
            writeCount+= 3;
        }
    }
}
```

RESULTADOS OBTIDOS

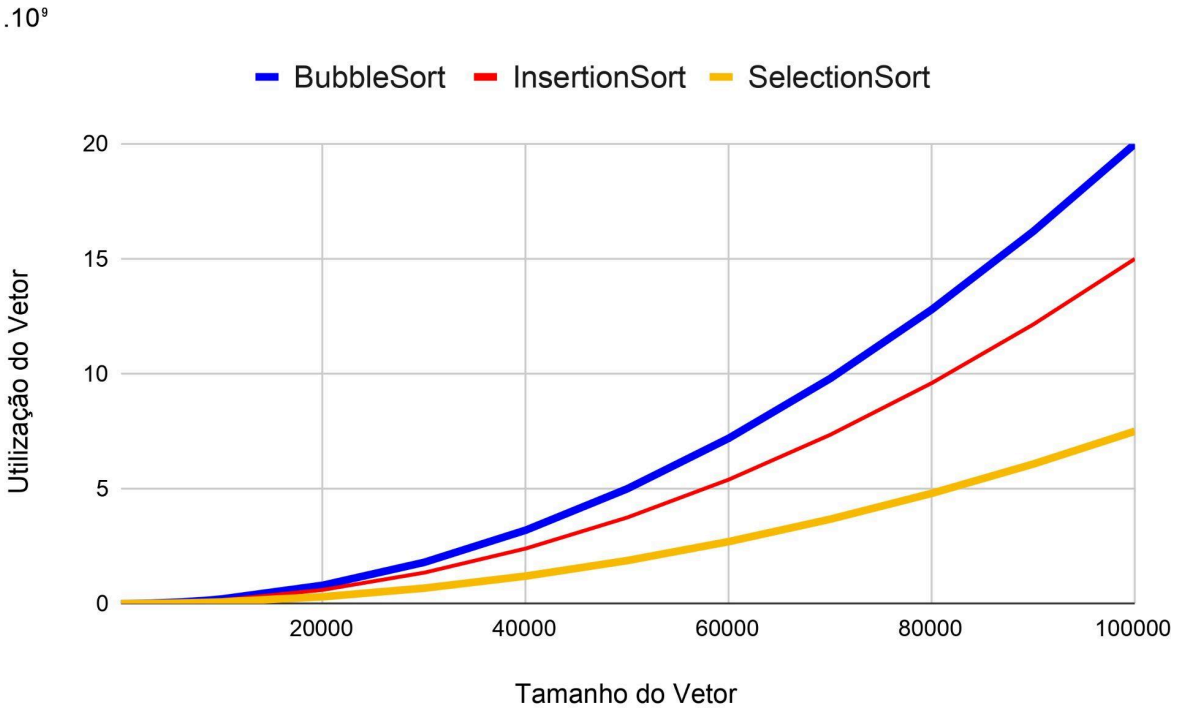
Vetores Crescentes:



Vetores Aleatórios:



Vetores Decrescentes:



CONCLUSÃO

Após a realização de todos os experimentos propostos, podemos concluir que:

BubbleSort: Apesar de ser o algoritmo com mais simplicidade dentre eles, apresentou boa performance em vetores crescentes, porém, como prescrito pela teoria, foi o método mais ineficiente para vetores aleatórios e decrescentes, precisando de mais leituras e escritas para ordenar os vetores:

InsertionSort: Apresentou um ótimo desempenho na ordenação de vetores crescentes. Ainda manteve um elevado número de leituras e escritas para lidar com os vetores aleatórios e decrescentes, porém, se mostrou superior ao BubbleSort nesses casos.

SelectionSort: Obteve o melhor desempenho entre os 3 métodos na ordenação de vetores aleatórios e decrescentes, porém, sua atuação sobre os vetores crescentes deixou a desejar, sendo o algoritmo mais ineficiente nesse caso.

Portanto, podemos concluir que o melhor algoritmo de ordenação é algo que irá variar conforme os dados que necessitam ser ordenados.

Referências

1- Sorting Algorithms Explained with Examples in Python, Java and C - FreeCodeCamp

https://www.freecodecamp.org/news/sorting-algorithms-explained-with-examples-in-python-java-and-c/

2- Método de Ordenação InsertionSort (implementação em linguagem C) - Ponto Acadêmico :

Método de ordenação Insertion Sort (implementação em linguagem C)