Algoritmos de ordenação em linguagem C: Um estudo comparativo

Mauricio Benjamin da Rocha

Abstract— **Contexto: Algoritmos de ordenação são amplamente utilizados em várias áreas da ciência da computação. Os mesmos são processos lógicos para se organizar uma determinada estrutura linear. Existem vários tipos de algoritmos de ordenação, desde os mais simples como o Bubble Sort aos mais complexos como o Quick Sort, onde cada um possui suas próprias características e desempenho. A avaliação comparativa desses algoritmos é fundamental para determinar qual é mais eficaz em diferentes cenários.**

Problema: Devido ao acúmulo de dados ser algo natural e com o decorrer do tempo, tornando-se cada vez mais um problema a ser resolvido desde, a dificuldade de encontrar determinado item em uma gama gigantesca de possibilidades, a necessidade de trabalhar com conjuntos de dados ordenados para realizar operações como pesquisa, filtragem, fusão, entre outras. Os algoritmos de ordenação tornam essas operações mais eficientes, uma vez que se beneficiam da ordem dos dados.

Resultados: Faz-se então, necessário o uso dos algoritmos de ordenação, onde neste estudo objetiva-se avaliar, por meio de experimentação em diferentes quantidades de dados desordenados, a diferença de resultados utilizando os algoritmos de ordenação Bubble Sort, Insertion Sort, Selection Sort que são tidos como os algoritmos mais simples. Para tanto, são executados testes em cada algoritmo usando vetores exatamente iguais visando identificar a diferença de performance entre os algoritmos.

Palavras-chave: Bubble Sort. Insertion Sort. Selection Sort. Algoritmos de ordenação.

I. INTRODUÇÃO

Algoritmos de ordenação desempenham um papel fundamental na ciência da computação permitindo classificar, organizar e reordenar valores em uma sequência específica de acordo com critérios específicos, facilitando o acesso eficiente aos dados posteriormente. Dados ordenados em critérios específicos são a base de áreas como Análise de Dados, Big Data, entre outras, pois revelam padrões e tendências ocultas, auxiliando em análises estatísticas, mineração de dados e tomada de decisões[1].

Devido ao acúmulo de dados ser algo natural no meio tecnológico devido a grande quantidade de dados circulando a cada instante, e com o decorrer do tempo torna-se cada vez mais um problema a ser resolvido desde.[2]Algoritmos de ordenação são fundamentais para garantir consistência de resultados em pesquisas científicas ou testes de desempenho, permitindo obter resultados repetíveis e comparáveis, independentemente da ordem inicial dos dados. Antes de começar a tratar um conjunto de dados, deve ter-se um olhar crítico para detectar este tipo de erros que podem destruir toda uma análise subsequente [3] . Dados ordenados facilitam a identificação de dados.

O objetivo deste estudo é avaliar, por meio de experimentação, análise, identificação e comparação a eficiência entre os algoritmos Bubble Sort, Insertion Sort e Selection Sort em diferentes cenários. A linguagem C foi escolhida como o ambiente de implementação devido à sua natureza de médio a baixo nível, que oferece maior eficiência de tempo em comparação com linguagens de mais alto nível. O presente trabalho identificou que dentre os algoritmos abordados para estudo, um deles mostrou-se mais eficiente .

A estrutura deste estudo é organizada da seguinte forma: na seção 2, será apresentada a metodologia utilizada durante o experimento; na seção 3, serão expostos os resultados obtidos; e, por fim, na seção 4, serão apresentadas as conclusões decorrentes da análise dos dados coletados no decorrer de todo o trabalho.

II. METODOLOGIA

Esta seção apresenta a metodologia abordada neste trabalho. A seção foi dividida em subseções visando ajudar no entendimento do trabalho. As subseções A, B e C abordam os algoritmos de ordenação codificados em linguagem C seguidos de uma breve descrição sobre os mesmos. A subseção D aborda a forma formatação dos dados usados. A subseção E detalha quais testes foram executados usando a base de dados definida. A subseção F aborda as ferramentas utilizadas e o ambiente usado para a realização do experimento.

A. BUBBLE SORT

O algoritmo de ordenação Bubble Sort compara pares de elementos adjacentes visando trocar suas posições caso estejam fora de ordem. O processo é repetido até que toda a lista esteja ordenada. O Bubble Sort percorre a lista várias vezes, movendo o elemento maior em direção ao final O mesmo é lento para grandes conjuntos de dados, pois requer muitas comparações e trocas. Pode ser facilmente entendido e implementado, sendo útil para pequenos conjuntos de dados ou como base para outros algoritmos de ordenação mais eficientes[4].

B. INSERTION SORT

O algoritmo de ordenação Insertion Sort divide a lista em duas partes, uma parte ordenada e uma parte não ordenada. O algoritmo percorre a lista da esquerda para a direita, inserindo cada elemento na posição correta da parte ordenada. Para fazer isso, compara cada elemento com os elementos anteriores da parte ordenada e os desloca para a direita até encontrar a posição correta. O processo continua até que todos os elementos estejam na posição correta. O mesmo é eficiente para listas pequenas ou quase ordenadas, mas pode ser lento para listas grandes ou inversamente ordenadas. Pode ser facilmente entendido e implementado[4].

C. SELECTION SORT

O algoritmo de ordenação Selection Sort é um método simples e eficiente para ordenar uma lista de elementos. Ele divide a lista em duas partes: uma parte ordenada e uma parte não ordenada. O algoritmo percorre a lista da esquerda para a direita, inserindo cada elemento na posição correta da parte ordenada. Para fazer isso, compara cada elemento com os elementos anteriores da parte ordenada e os desloca para a direita até encontrar a posição correta. O processo continua até que todos os elementos estejam na posição correta. O Insertion Sort é eficiente para listas pequenas ou quase ordenadas, mas pode ser lento para listas grandes ou inversamente ordenadas. No entanto, é fácil de entender e implementar[5].

D. FORMATAÇÃO DOS DADOS

Apesar de ser possível a implementação dos algoritmos de ordenação com qualquer estrutura linear, foi definido visando facilitar a geração e testes sobre os dados o uso de apenas algarismos numéricos inteiros (sequências de números) no intervalo entre 0 (zero) e 9 (nove). A ordenação escolhida foi de ordem crescente de acordo com os valores contidos na estrutura, de forma que ao final teremos acesso a uma estrutura totalmente ordenada com o primeiro elemento sendo o menor e o último elemento sendo o maior.

E. METODOLOGIA DE TESTES

Foram executados testes em cada algoritmo usando a estrutura de dados vetor, onde foram completamente preenchidos de valores totalmente aleatórios visando simular a precisão dos algoritmos em situações de aplicações reais e identificar a diferença de performance. Os vetores possuem tamanhos dez, cem, mil, dez mil, cem mil. Para cada tamanho de vetor, foram realizadas trinta tentativas de ordenação por algoritmo, onde o tempo de cada algoritmo foi salvo e a média de cada foi calculada.

Devido às limitações computacionais para execução do experimento tais quantidades de dados foram escolhidos para ajudar na identificação das situações em que cada algoritmo desempenha seu papel, seja ele mais eficiente ou menos eficiente. Foram preparados três vetores exatamente iguais para cada algoritmo, onde cada algoritmo teve acesso a seu respectivo vetor, do qual os mesmos tiveram que ordenar os valores em ordem crescente. Os resultados obtidos se encontram na tabela de resultados.

F. FERRAMENTAS E AMBIENTE

O Visual Studio Code foi escolhido como ambiente de desenvolvimento, onde o mesmo é um editor de código-fonte desenvolvido pela Microsoft para computadores com sistemas operacionais Windows, Linux e macOS. Tal ferramenta possui diversas funcionalidades e customizações que permitem facilitar e acelerar o processo de desenvolvimento e experimentação. sendo alguma delas o realce de sintaxe, complementação inteligente de código, entre outras funcionalidades que fornecem suporte para seus usuários.

A máquina usada para a realização dos testes possui as seguintes configurações de hardware conforme mostra a tabela I.

TABLE I
HARDWARE USADO

Processador	Intel(R) Core(TM) i3-9100F
Frequência	3.60 GHz;
Memória RAM	8,00 GB (DDR4 2400MHz);
Sistema Operacional	Windows 10 PRO (64 bits).

III. RESULTADOS

Esta seção apresenta os resultados obtidos através da experimentação proposta conforme foi abordado nas seções anteriores. As Tabelas II, III, IV, V e VI contem os dados coletados durante os testes realizadas , onde as mesmas são compostas pelo tempo gasto em mili segundos de cara algoritmo. A Figura VII

TABLE II
TEMPO GASTO EM UM VETOR DE TAMANHO 10

TENTATIVA	BUBBLESORT (ms)	INSERTION SORT (ms)	SELECTION SORT (ms)
1	0,0004	0,0003	0,0004
2	0,0002	0,0001	0,0003
3	0,0002	0	0,0003
4	0,0001	0	0,0001
5	0,0001	0,0001	0,0002
6	0,0001	0	0,0001
7	0,0001	0,0001	0,0001
8	0,0001	0	0,0001
9	0,0001	0,0001	0,0001
10	0,0001	0,0001	0,0002
11	0,0001	0	0,0001
12	0,0001	0,0001	0,0001
13	0,0001	0,0001	0,0002
14	0,0001	0	0,0001
15	0,0001	0	0,0001
16	0,0001	0,0001	0,0002
17	0,0001	0	0,0001
18	0,0001	0	0,0001
19	0,0001	0,0001	0,0001
20	0,0001	0,0001	0,0002
21	0,0001	0	0,0001
22	0,0001	0,0001	0,0001
23	0,0001	0,0001	0,0002
24	0,0001	0	0,0001
25	0,0001	0	0,0001
26	0,0001	0,0001	0,0001
27	0,0001	0	0,0002
28	0,0001	0	0,0001
29	0,0001	0,0001	0,0001
30	0,0001	0,0001	0,0002

TABLE III

TEMPO GASTO EM UM VETOR DE TAMANHO 100

TENTATIVA	BUBBLESORT (ms)	INSERTION SORT (ms)	SELECTION SORT (ms)
1	0,0186	0,0062	0,0125
2	0,0093	0,0003	0,0109
3	0,0093	0,0003	0,0109
4	0,0096	0,0003	0,0109
5	0,0096	0,0003	0,0109
6	0,0095	0,0003	0,0109
7	0,0093	0,0003	0,0108
8	0,0096	0,0003	0,011
9	0,0094	0,0003	0,0109
10	0,013	0,0004	0,0109
11	0,0096	0,0003	0,0108
12	0,0096	0,0003	0,0109
13	0,0095	0,0003	0,0109
14	0,0095	0,0003	0,0108
15	0,0094	0,0003	0,0108
16	0,0096	0,0004	0,0109
17	0,0102	0,0005	0,0111
18	0,0094	0,0003	0,1342
19	0,0208	0,0004	0,0111
20	0,0093	0,0004	0,0109
21	0,0093	0,0003	0,0551
22	0,0094	0,0003	0,0108
23	0,0096	0,0003	0,0108
24	0,0096	0,0003	0,0109
25	0,0096	0,0003	0,0108
26	0,0094	0,0003	0,011
27	0,0095	0,0003	0,011
28	0,0095	0,0004	0,0109
29	0,0095	0,0003	0,011
30	0,0096	0,0004	0,0109

TABLE IV

TEMPO GASTO EM UM VETOR DE TAMANHO 1000

TENTATIVA	BUBBLESORT (ms)	INSERTION SORT (ms)	SELECTION SORT (ms)
1	3,7045	0,9423	1,0303
2	0,9231	0,003	1,0035
3	0,9258	0,0031	1,0055
4	0,92	0,0029	0,9999
5	0,9442	0,0031	0,998
6	1,4462	0,0036	2,5854
7	1,1712	0,0032	1,0072
8	0,9464	0,003	1,036
9	0,9646	0,0031	0,9987
10	0,9194	0,003	0,9985
11	0,9197	0,0029	1,5527
12	2,5654	0,0032	1,0431
13	1,5227	0,0035	1,0584
14	0,9273	0,0033	1,0106
15	1,0023	0,0034	1,0051
16	0,9232	0,0038	1,0084
17	3,4834	0,0031	1,0047
18	0,9248	0,0032	1,013
19	1,0123	0,0032	1,0012
20	0,9209	0,0031	1,186
21	0,9415	0,003	1,0373
22	0,9204	0,0031	4,063
23	0,9591	0,0035	1,1952
24	1,466	0,0034	1,6732
25	1,0235	0,0031	1,0347
26	0,9729	0,003	6,5249
27	1,7911	0,0033	1,041
28	1,1735	0,0031	1,1406
29	2,2489	0,003	4,8105
30	1,3716	0,0032	1,0182

TABLE V

TEMPO GASTO EM UM VETOR DE TAMANHO 10000

TENTATIVA	BUBBLESORT (ms)	INSERTION SORT (ms)	SELECTION SORT (ms)
1	307,1293	72,28000	163,4707
2	139,3381	0,02990	140,4163
3	118,0314	0,03020	126,4625
4	120,1979	0,02970	166,5166
5	163,8374	0,02990	169,644
6	148,9195	0,03020	173,7517
7	137,2258	0,03180	158,3854
8	155,945	0,03010	150,6479
9	140,4895	0,02990	143,9908
10	186,4372	0,21360	153,0697
11	129,92	0,04290	141,8804
12	144,0432	0,03010	172,4061
13	137,9302	0,03000	153,9873
14	135,1233	0,03170	174,8061
15	144,6133	0,03090	142,7376
16	148,6445	0,03640	161,5329
17	127,7072	0,03100	137,2888
18	128,6403	0,03040	142,6593
19	121,3695	0,02980	133,8989
20	129,5507	0,04110	141,5283
21	128,0821	0,03010	146,6382
22	134,5804	0,03000	143,4727
23	128,7575	0,03310	137,1927
24	130,8067	0,03010	143,6323
25	132,8913	0,03030	139,1851
26	132,9708	0,03020	135,4358
27	131,2158	0,02990	142,0058
28	135,6341	0,03320	147,7277
29	137,1067	0,03010	146,0085
30	123,9331	0,02980	143,9492

TABLE VI

TEMPO GASTO EM UM VETOR DE TAMANHO 100000

TENTATIVA	BUBBLESORT(ms)	INSERTION SORT(ms)	SELECTION SORT(ms)
1	37044,4596	5292,342	11724,3686
2	12734,9373	0,3142	14786,2839
3	12824,4639	0,3134	13620,144
4	12918,7808	0,3019	14059,7662
5	13665,9152	0,3001	15527,1369
6	16837,1349	0,3233	18319,2621
7	16833,6109	0,2997	15267,9057
8	12056,0524	0,3073	18374,5671
9	12539,6307	0,3359	14312,4769
10	13573,0441	0,3605	13843,2595
11	13213,0105	0,3019	12897,2603
12	12315,4189	0,3572	13181,4995
13	13538,876	0,3081	13365,9554
14	12444,1995	0,3105	13567,1255
15	12725,8914	0,3781	14429,4575
16	12454,937	0,3234	12551,8451
17	12645,6928	0,3017	15360,3351
18	13080,2784	0,7478	13472,2926
19	12218,0898	0,302	13781,7939
20	13624,1313	0,303	13492,0529
21	12812,7468	0,3032	14407,4719
22	13580,6895	0,4533	13986,234
23	12926,3116	0,3017	13017,0097
24	13636,0091	3,0524	14479,0824
25	11974,9532	0,3108	13239,1272
26	12359,3845	0,3154	14728,0907
27	13030,2474	0,6046	13807,3424
28	12759,4568	0,3061	13142,559
29	12461,206	1,719	13742,2173
30	12764,2275	0,371	13477,0863

TABLE VII

MÉDIA EM MILI SEGUNDOS DOS TEMPOS OBTIDOS DURANTE O EXPERIMENTO

TAMANHO DO VETOR	BUBBLESORT(ms)	INSERTIONSORT(ms)	SELECTIONSORT(ms)
10	0,0001	0,0001	0,0001
100	0,0096	0,0003	0,0109
1000	0,96875	0,0031	1,0325
10000	134,85185	0,0302	143,97
100000	12818,60535	0,3138	13794,56815

As Figuras 1, 2, 3, 4 e 5 mostram a comparação entre as médias dos tempos gastos por cada algoritmo de ordenação em diferentes quantidades de dados. Observa-se

que a medida que os dados aumentam, se torna mais evidente a diferença de desempenho entre os algoritmos.

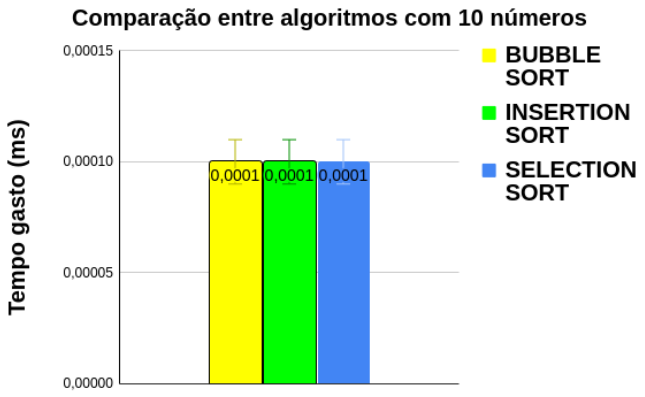

Fig. 1. Desempenho com 10 números.

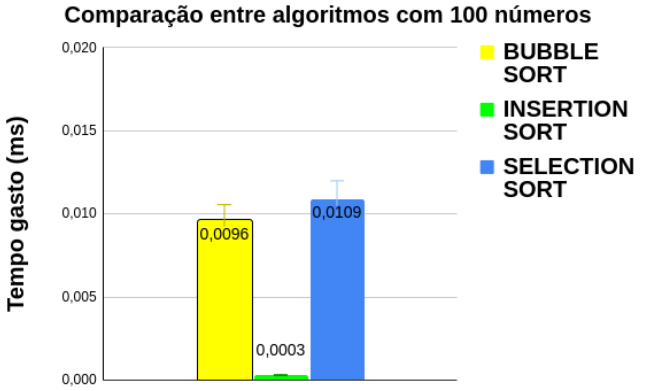

Fig. 2. Desempenho com 100 números.

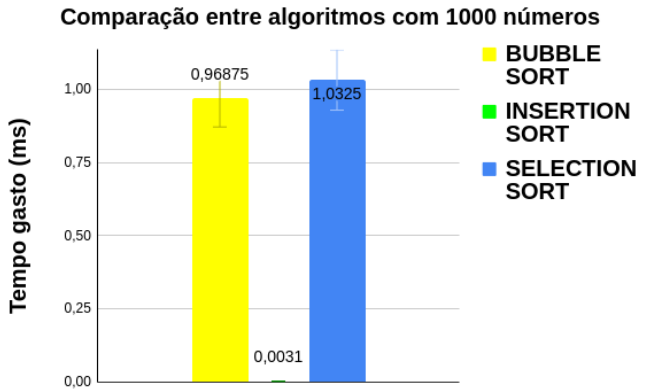

Fig. 3. Desempenho com 1.000 números.

Conforme foi apresentado nos resultados anteriores, podemos concluir que se nossa base de dados é muito pequena, podemos optar por escolher qualquer algoritmo. Entretanto, à medida que nossos dados vão aumentando devemos optar por escolher o Insertion Sort como método de ordenação

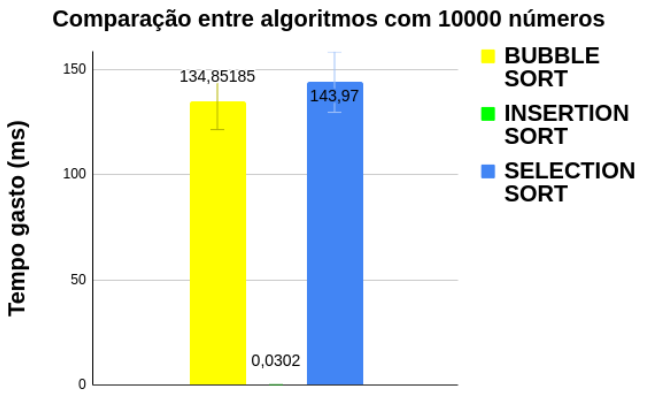

Fig. 4. Desempenho com 10.000 números.

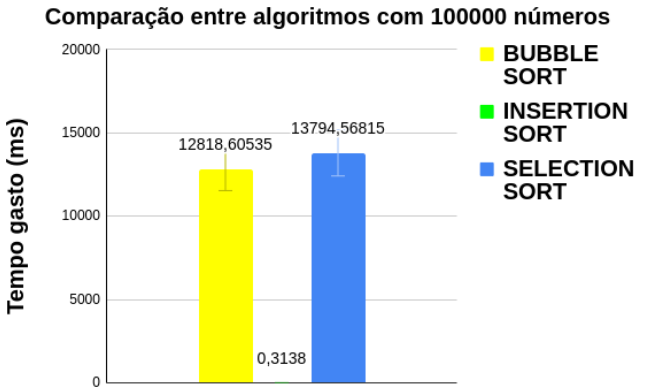

Fig. 5. Desempenho com 100.000 números

se comparado com os demais, pois teremos muito mais eficiência de tempo nos permitindo realizar muito mais ações no mesmo intervalo.

IV. ANEXO

Esta seção será dedicada para ilustrar os algoritmos de ordenação citados no decorrer deste artigo codificados em linguagem C.

```c
void bubbleSort(int array[], int size)
{
    int i, j;
    int temp;

    for (i = 0; i < size - 1; i++)
    {
        for (j = 0; j < size - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

Listing 1. Bubble Sort em linguagem C

```c
void insertionSort(int array[], int size)
```

```
{
    int i, key, j;


    for (i = 1; i < size; i++)
    {
        key = array[i];
        j = i - 1;


        while (j >= 0 && array[j] > key)
        {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
```

Listing 2. Insertion Sort em linguagem C

```
void selectionSort(int array[], int size)
{
    int i, j, min_idx, aux;

    for (i = 0; i < size - 1; i++)
    {
        min_idx = i;


        for (j = i + 1; j < size; j++)
        {
            if (array[j] < array[min_idx])
                min_idx = j;
        }
        aux = array[min_idx];
        array[min_idx] = array[i];
        array[i] = aux;
    }
}
```

Listing 3. Selection Sort em linguagem C

REFERENCES

[1] Alexandre da Silva Pedroso and Fausto Gonçalves Cintra. Estudo analítico do desempenho de algoritmos de ordenação.

[2] Jackson EG Souza, João Victor G Ricarte, and Náthalee Cavalcanti de Almeida Lima. Algoritmos de ordenação: Um estudo comparativo. *Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA (ISSN 2526-7574)*, (1), 2017.

[3] Maria Eugénia Graça Martins and João Pedro da Ponte. Organização e tratamento de dados, 2011.

[4] Herbert Schildt. *C completo e total*. Makron, 1997.

[5] Nivio Ziviani et al. *Projeto de algoritmos: com implementações em Pascal e C*, volume 2. Thomson Luton, 2004.