Desenvolvimento e análise do algoritmo BubbleSort Clássico e BubbleSort com Flag

Gustavo Martini[1], Gustavo Macedo[2], Vinicius Gilnek Drage[1]

[1]Universidade Estadual do Oeste do Paraná(UNIOESTE)
R. Universitária, 1619 - Universitário, Cascavel - PR, 85819-110

gutamen@live.com, gustavomacedo1366@hotmail.com, vinidrage@gmail.com

Abstract. *This work aims to evaluate the behavior of the BubbleSort sorting algorithm, in particular its temporal complexity, in different scenarios and conditions..*

Resumo. *Este trabalho tem como objetivo avaliar o comportamento do algoritmo de ordenação BubbleSort, em particular a sua complexidade temporal, em diferentes cenários e condições*

1. Introdução

O objetivo deste trabalho é implementar, testar e analisar o algoritmo BubbleSort, tanto sua implementação clássica quanto a versão com flag de controles.

A linguagem escolhida para implementação foi o cpp, usando para o código principal, onde efetivamente é medido o tempo do algoritmo, uma instrução em Assembly.

2. Materiais Utilizados

O principal material de utilização foi a máquina de testes classificada por sua seguinte configuração:

Processador: Ryzen 7700X(16 núcleos, 8 thread, 5ghz)

Memória Ram: 32Gb DRR5 (6000 mhz)

Sistema Operacional de Execução: Linux (Debian 12)

IDE: Fora apenas utilizado apenas nvim para implementação, execução no terminal

3. Resultado e Discussão

Para com a análise assintótica do código fora feita sobre a parte a qual efetivamente executa o BubbleSort, obtém-se então:

- Para o código Bubble Sort Clássico: $18n^2 + 9n + 2 \rightarrow O(n^2)$
- Para o código Bubble Sort com Flag: $22n^2 + 6n + 2 \rightarrow O(n^2)$

Em relação a ordem assintótica, ambas implementações estão em $O(n^2)$, por mais que o BubbleSort com Flag tenha um custo maior em seus valores, durante a execução, devido ao seu teor de finalização prévia, a velocidade de execução do BubbleSort com Flag, em casos Ordenados ou Parcialmente Ordenados serão mais rápidas.

Podemos verificar um modelo em Pseudo Código do BubbleSort Clássico e Bubble Sort com flag, respectivamente no Código 1 e Código 2, abaixo:

```
função bubble_sort_Classica(vetor[], tamanho)
{
        aux = 0;
        enquanto(k = 1; k < tamanho; k = k + 1)
{
                se(vetor[j] > vetor[j + 1]{
                        aux = vetor [j];
                        vetor[j] = vetor[j+1]
                        vetor[j+1] = aux;
                }
        }
}
```

Código 1. Pseudo Código BubbleSort Classico

```
função bubble_sort_Flag(vetor[], tamanho)
{
        aux = 0;
        enquanto(k = 1; k < tamanho; k = k + 1)
{
                mudou = falso;
                se(vetor[j] > vetor[j + 1]{
                        aux = vetor [j];
                        mudou = verdade;
                        vetor[j] = vetor[j+1]
                        vetor[j+1] = aux;
                }
                se(não mudou) retorna;
        }
```

```
}
```

Código 2. Pseudo Código BubbleSort com Flag

Abaixo nas tabelas, **Tabela 1** e **Tabela 2**, é possível observar os resultados das execuções efetivadas respectivamente com os códigos BubbleSort Clássico e BubbleSort com Flag, para os conjuntos de dados Aleatórios, Decrescente, Ordenados, Parcialmente Ordenados, todos com os respectivos valores 100, 200, 500, 1000, 2000, 5000, 7500, 10000, 15000, 30000, 50000, 75000, 100000, 200000, 500000, 750000, 10000000, 12500000, 15000000, 20000000.

Tabela 1. Conjunto de resultados dos testes com algoritmo BubbleSort Clássico.

	100	200	500	1000	2000	5000	7500	10000	15000	30000
Aleatório	0	0	0	0	0	0	0	0	0	0
Decrescentes	0	0	0	0	0	0	0	0	0	1
Ordenados	0	0	0	0	0	0	0	0	0	0
Parc. Ordenados	0	0	0	0	0	0	0	0	0	0

	50000	75000	100000	200000	500000	750000	10000000	12500000	15000000	20000000
Aleatório	2	5	11	65	430	1006	1831	2919	4181	7381
Decrescentes	3	7	14	57	351	795	1401	2177	3153	5607
Ordenados	1	2	4	22	145	328	599	917	1384	2369
Parc. Ordenados	1	2	5	22	134	328	577	917	1384	2362

Tabela 2. Conjunto de resultados dos testes com algoritmo BubbleSort com Flag.

	100	200	500	1000	2000	5000	7500	10000	15000	30000
Aleatório	0	0	0	0	0	0	0	0	0	0
Decrescentes	0	0	0	0	0	0	0	0	0	1
Ordenados	0	0	0	0	0	0	0	0	0	0
Parc. Ordenados	0	0	0	0	0	0	0	0	0	0

	50000	75000	100000	200000	500000	750000	10000000	12500000	15000000	20000000
Aleatório	2	5	10	62	441	1003	1882	2882	4097	7274
Decrescentes	3	7	12	52	325	733	1400	2154	3150	5567
Ordenados	0	0	0	0	0	0	0	0	0	0
Parc. Ordenados	0	0	1	2	5	8	12	16	16	18

Com os dados demonstrados nas tabelas, **Tabela 1** e **Tabela 2**, foi possível montar os gráficos, na figura **Figura 1** é demonstrado a comparação do algoritmo convencional e com flag, onde é observado o comportamento quadrático do bubble sort e além da pequena melhora do algoritmo modificado, por se tratar de um conjunto de dados aleatorizado o aumento no desempenho é dependente do quanto o vetor está organizado. Quando ocorre um conjunto de dados ordenado de forma decrescente é encontrado o pior caso para o algoritmo, porém nos testes executados o aleatório apresentou um desempenho inferior ao decrescente, o fato pode ser observado nas tabelas **Tabela 1** e **Tabela 2**, isso acontece pela fato de que repetição a instrução de troca o processador passa a fazer preempção da instrução a ser executada (SILBERSCHATZ et al, 2015). A melhora com a flag é praticamente inexistente, apresentado na figura **Figura 2** em que as duas linhas referentes ao tempo de execução estão sobrepostas.

Tempo de Execução do BubbleSort em segundos para dados desorganizados.

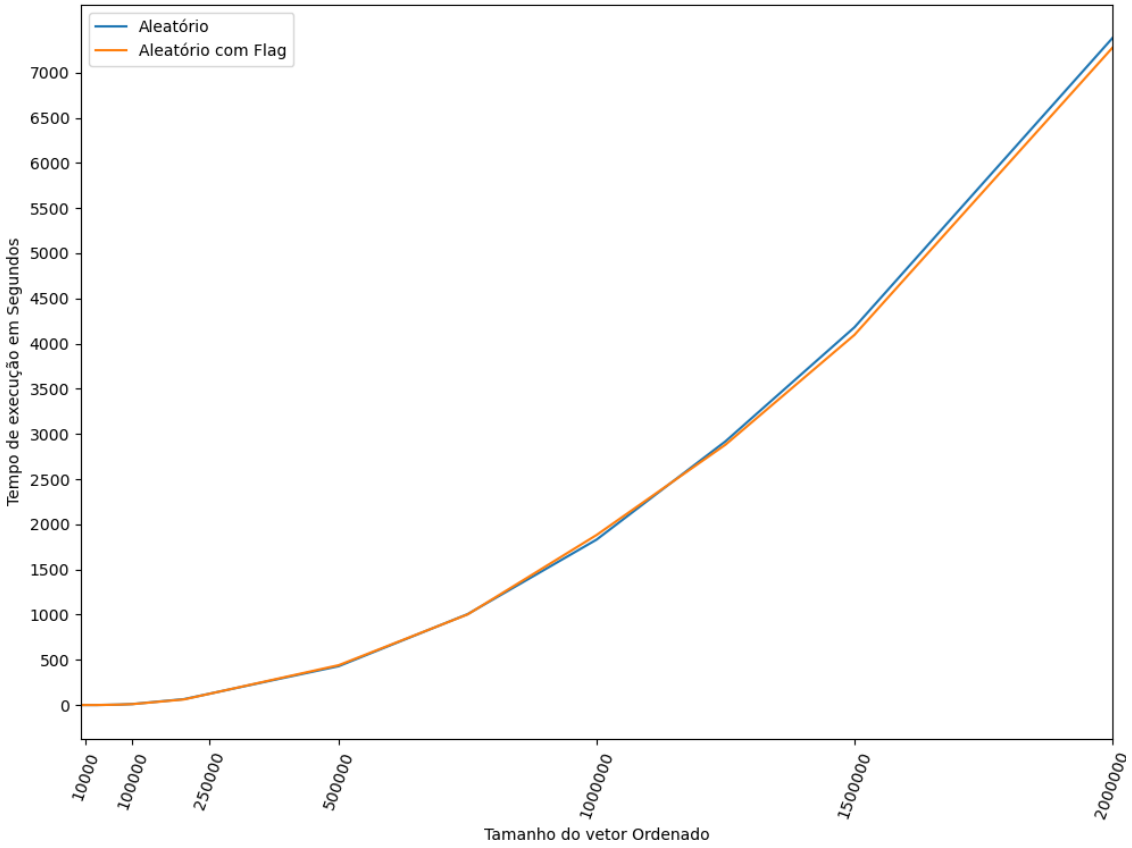

Figura 1. Gráfico da execução das duas variações do BubbleSort com dados aleatoriamente posicionados.

Tempo de Execução do BubbleSort em segundos para dados organizados em ordem decrescente.

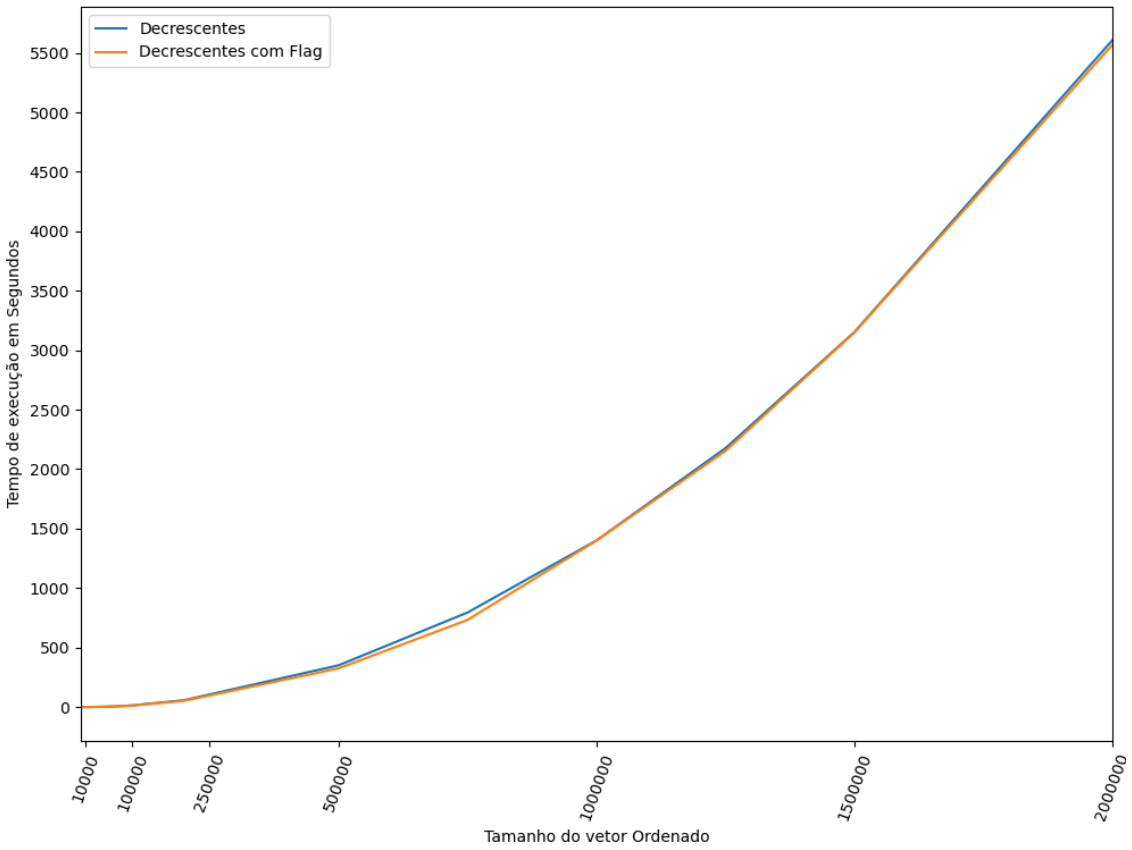

Figura 2. Gráfico da execução das duas variações do BubbleSort com dados organizados decrescente.

A partir do momento em que o conjunto de dados se encontra quase ordenado o salto de desempenho, que a variação com a flag proporciona, é notável, reduzindo tempos de 2300 segundos para 18 segundos, explicitado na figura **Figura 3**.

Tempo de Execução do BubbleSort em segundos para dados quase organizados.

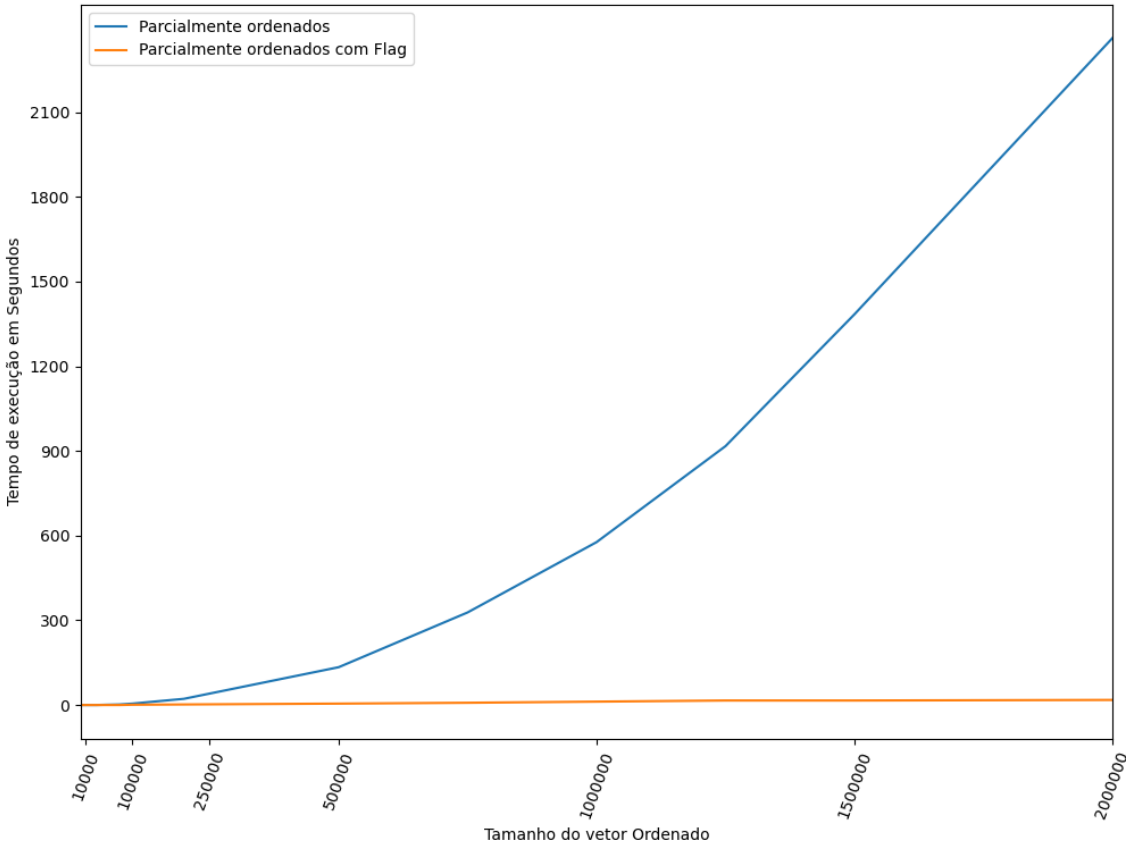

Figura 3. Gráfico da execução das duas variações do BubbleSort com dados quase organizados.

Continuando a discussão de ordenação a flag ainda proporciona a possibilidade de um tempo de execução de menos de 1 segundo para o caso dos dados estarem completamente ordenados, assim como na figura **Figura 4**.

Tempo de Execução do BubbleSort em segundos para dados organizados.

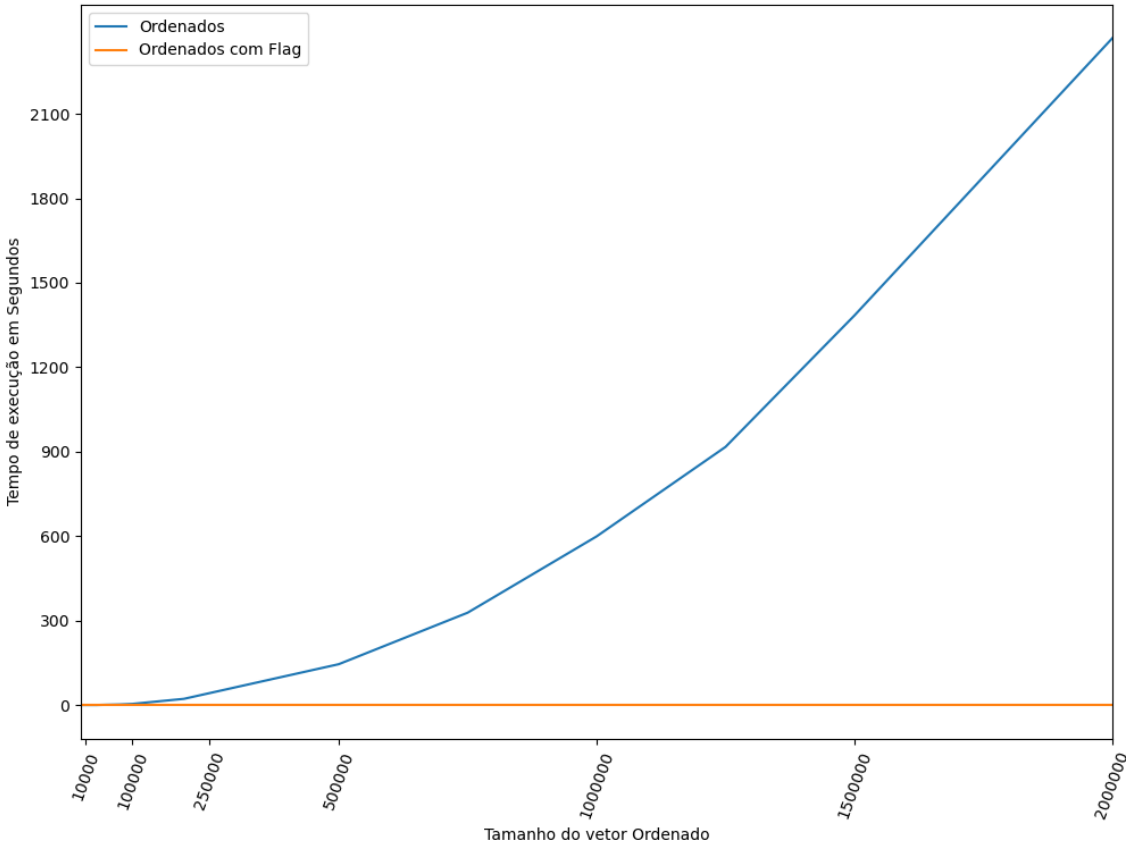

Figura 4. Gráfico da execução das duas variações do BubbleSort com dados organizados.

Na figura **Figura 5**, foram apresentados todos os resultados obtidos a partir dos distintos conjuntos de dados, assim como métodos diferentes, onde ficam visíveis as vantagens do algoritmo quando o conjunto está ordenado e utiliza a função com flag, além disso é apresentada a divergência que ocorre com os dados na decrescente com relação aos dados aleatórios. Também existe o tempo menor para o caso em que o algoritmo está parcialmente ordenado e coordenado mesmo sem a flag, isso ocorre pois o acesso a memória para organizar o vetor não acontece, o que reduz significativamente o tempo de execução, já que o acesso à memória é significativamente mais demora que uma comparação entre registadores.

Tempo de Execução do BubbleSort em segundos para todos os casos de teste.

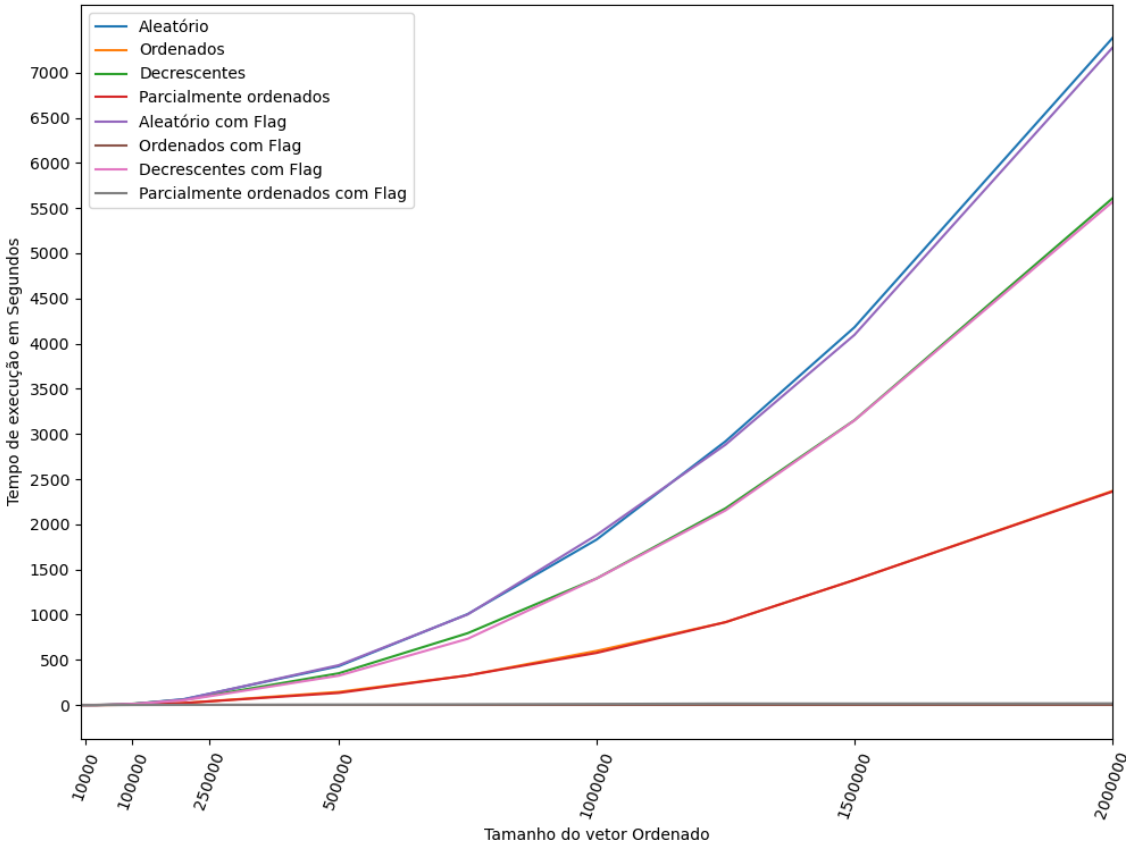

Figura 5. Gráfico de todas as execuções.

4. Conclusão

Por fim, pode-se concluir que os algoritmos testados BubbleSort Clássico e BubbleSort com Flag são relativamente não recomendáveis para conjuntos de dados grandes. Em aplicações simples com níveis baixos de quantidade de dados, o mesmo desempenha seu papel de forma aceitável.

Em relação às diferenças entre os dois modelos de implementação, o sistema com Flag representa uma grande vantagem em cenários específicos com conjunto de dados Ordenados ou Parcialmente Ordenados.

References

Abraham Silberschatz, Greg Gagne, Peter Baer Galvin. Fundamentos de Sistemas Operacionais. Edição 6. Editora, LTC. ISBN, 8521614144, 9788521614142. Num. págs. 600 páginas, 2015.