Análise Empírica de Algoritmos de Ordenação e Busca

Thais Zorawski, Vitor Bueno de Camargo

3 de Dezembro de 2018

1 Introduction

O objetivo deste trabalho é comparar resultados práticos com os valores obtidos em sala através de métodos matemáticos para encontrar o custo da execução de algoritmos. Foram realizadas as análises dos algoritmos Bubble Sort, Bubble Sort c/ otimizações, Insertion Sort, Selection Sort, Merge Sort, Heap Sort, Quick Sort, Busca Binária e Busca do Vetor Máximo. Serão testados com listas de 1000 elementos (valores de -999 a 1000), 10.000 elementos (valores de -9999 a 10000) e 100.000 elementos (valores de -99999 a 100000).

2 Análise dos Algoritmos

Para a análise dos algoritmos foram captados dois recursos: números de instruções executadas e tempo de execução. Para obter o número de linhas executadas colocou-se um contador no laço mais interno, para os métodos iterativos, ou logo após a entrada na função, para métodos recursivos. Para análise do tempo de execução foi cálculado os ciclos de clock para cada execução e então divididos pelo fator de clocks por segundo, da máquina.

2.1 BubbleSort

Para o BubbleSort e o BubbleSort Otimizado, analizamos seu caso médio, que tem como custo O(n^2). Ao se analizar o gráfico obtido (Figura 1), pode-se perceber que o número de linhas executadas nos dois casos, para valores suficientemente grandes, é inferior ao esperado pela notação assintótica. Isso se da pelo fato de o compilador fazer otimizações no código. Já para número menores, as duas funções obedecem o padrão da notação O(n^2).

Ao realizarmos uma comparação entre os dois algoritmos, vemos que o BubbleSort Otimizado tem uma melhor execução em relação ao BubbleSort, porém essa melhora não é tão significativa, como pode ser visto na Tabela 1.

Em relação ao tempo de execução, vemos que, para os dois algoritmos, ele cresce de forma exponencial (Tabela 2), porém, ao contrário do comportamento

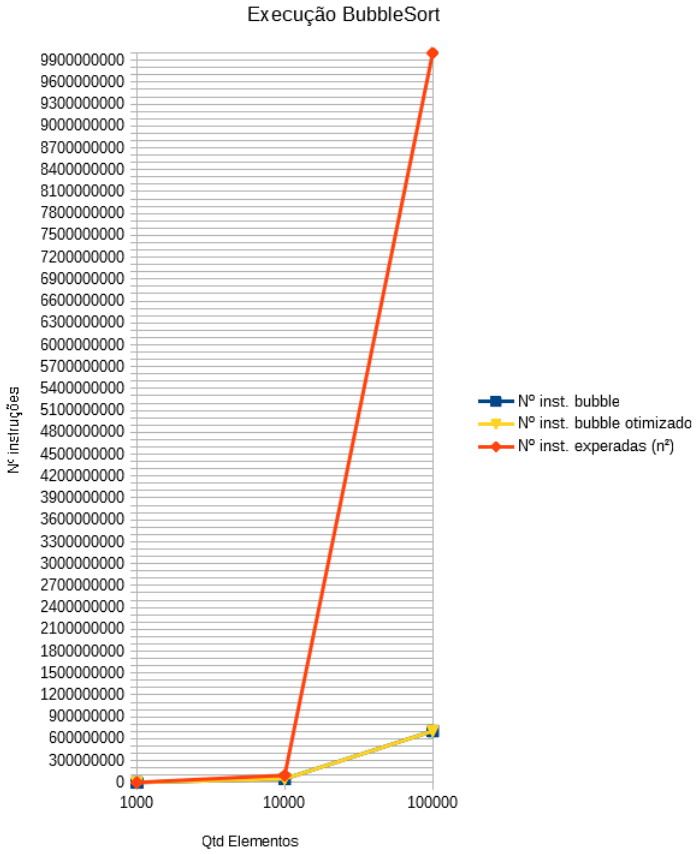

Figura 1: Gráfico do número de linhas executadas com os algoritmos BubbleSort e BubbleSort Otimizado. A terceira reta representa o comportamento esperado a partir da análise assintótica do método BubbleSort.

Tabela 1: Resultado da Execução dos Algoritmos BubbleSort e BubbleSort Otimizado

Qtd Elementos	Nº inst. experadas (n)	Nº inst. bubble	Nº inst. bubble otimizado
1000	1000000	500500	500200
10000	100000000	50005000	49991139
100000	10000000000	705082704	705009551

do número de linhas executadas, na maioria das vezes o tempo de execução do BubbleSort Otimizado foi maior que o BubbleSort puro. Isso pode ter acontecido

porque o método otimizado faz mais verificações durante seu processamento do que o método puro.

Tabela 2: Tempo de Execução dos Algoritmos BubbleSort e BubbleSort Otimizado

Qtd Elementos	Tempo BubbleSort (s)	Tempo BubbleSort Otimizado (s)
1000	0.002000	0.003000
10000	0.446000	0.277000
100000	30.868000	31.515000

2.2 InsertionSort

Para o método InsertionSort analizamos o seu caso médio, de notação O(n^2). No gráfico da Figura 2 é possível perceber que sua execução obeteve melhor desempenho do que a notação matemática prevê.

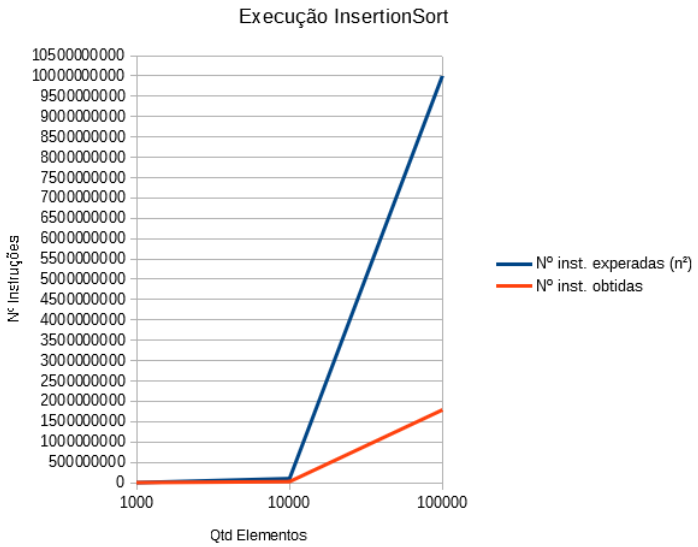

Figura 2: Gráfico do número de linhas executadas pelo algoritmo InsertionSort e o comportamento previsto pela análise assintótica.

O seu tempo de execução também cresce de acordo com os dados, porém em uma medida menor que o tempo do BubbleSort (Tabela 3).

Tabela 3: Tempo de Execução do Algoritmo InsertionSort

Qtd Elementos	Tempo
1000	0.001000
10000	0.062000
100000	5.968000

2.3 SelectionSort

O custo do método SelectionSort é O(n^2), tanto para o melhor caso quanto para o caso médio ou pior caso. Este método teve um comportamento muito parecido com o InsertionSort, como é possível observar na Figura 3. Seu tempo de execução é maior que o do InsertionSort, porém continua menor que o do BubbleSort. O crescimento do tempo também se assemelha a uma taxa exponencial (olhe a Tabela 4).

Tabela 4: Tempo de Execução do Algoritmo SelectionSort

Qtd Elementos	Tempo
1000	0.002000
10000	0.129000
100000	12.505000

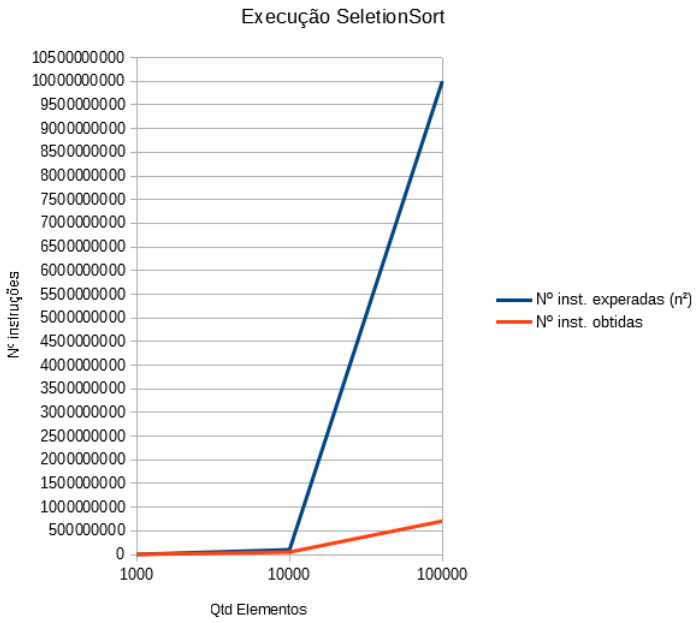

Figura 3: Gráfico que mostra o número de linhas executadas pelo algoritmo SelectionSort e o comportamento previsto pela análise assintótica.

2.4 MergeSort

O MergeSort tem custo O(n log n), assim como os demais algoritmos de ordenação recursivos analisados, porém foi o que apresentou os melhores resultados, tanto em tempo (Tabela 5) quanto em linhas de execução. Foi o único algoritmo recursivo que a quantidade de linhas executadas foi menor que a quantidade prevista pela notação (Figura 4).

Tabela 5: Tempo de Execução do Algoritmo MergeSort

Qtd Elementos	Tempo
1000	0.000000
10000	0.001000
100000	0.010400

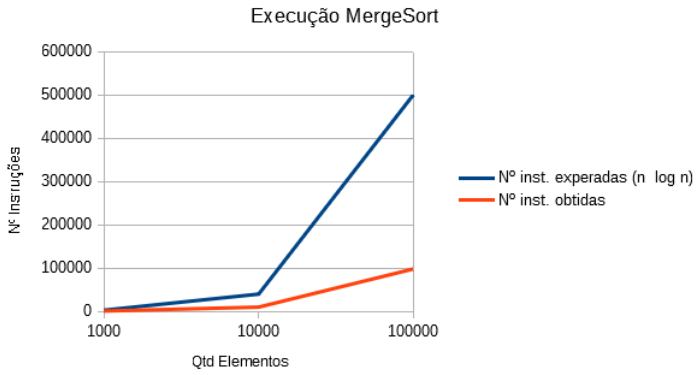

Figura 4: Gráfico que mostra o número de instruções executadas pelo algoritmo MergeSort e o comportamento previsto por sua análise assintótica.

2.5 HeapSort

Para todos os casos o HeapSort é O(n log n). É possível verificar que o número de instruções rodadas supera o previsto pela notação assintótica (Figura 5). Isso se deve ao número de chamadas recursivas feitas durante a execução. Apesar disso, seu tempo de processamento foi bem baixo (Tabela 7), sendo o terceiro algortimo mais rápido avalidado neste trabalho. Assim como os demais algoritmos recursivos avaliados, o tempo para execução com a lista com 1000 elementos foi bem pequeno, muito próximo de 0,000000 segundos.

Tabela 6: Tempo de Execução do Algoritmo HeapSort

Qtd Elementos	Tempo
1000	0.000000
10000	0.004000
100000	0.031000

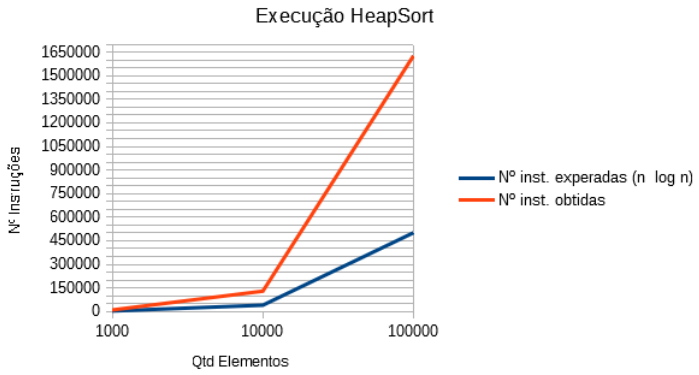

Figura 5: Gráfico que mostra o número de linhas executadas pelo algoritmo HeapSort e o comportamento previsto pela análise assintótica.

2.6 QuickSort

De todos os algoritmos recursivos avaliados, o QuickSort é o que mais apresenta discrepância com seu custo matemático (avaliamos aqui o caso médio, O(n log n)), como podemos verificar na Figura 6. Todavia, seu tempo de execução (Tabela **??**) foi o segundo melhor.

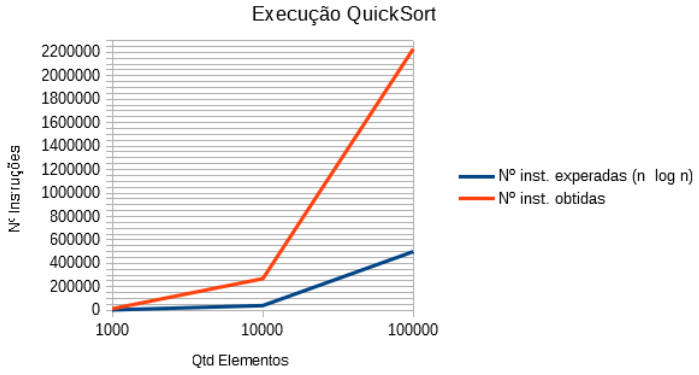

Figura 6: Gráfico que mostra o número de linhas executadas pelo algoritmo QuickSort e o comportamento previsto pela análise matemática.

Tabela 7: Tempo de Execução do Algoritmo QuickSort

Qtd Elementos	Tempo
1000	0.000000
10000	0.002000
100000	0.015000

2.7 Busca Binária

Com pior caso O(log n), o algoritmo Busca Binário foi o que teve melhores resultados nos testes. Para as três execuções, seu tempo ficou muito próximo de 0,000000 segundos, não sendo possível estabelecer um número diferente de zero. Ao se procurar um elemento inexistente na lista, nenhuma das execuções superaram 20 instruções (Figura 7). Apesar dos números de linhas serem superiores aos valores previstos por sua notação assintótica, ainda são muito bons.

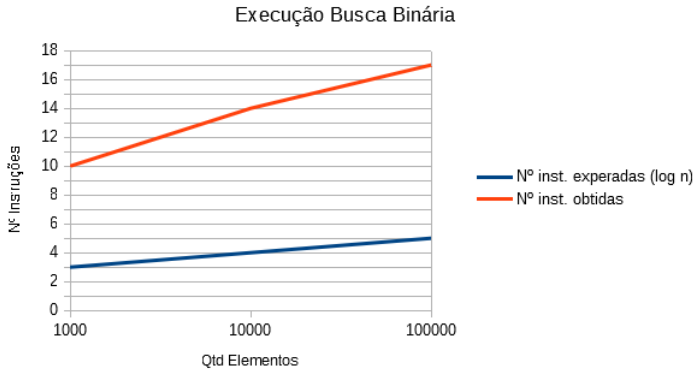

Figura 7: Gráfico que mostra o número de linhas executadas pelo algoritmo Busca Binária e o comportamento previsto por análise matemática.

2.8 Busca do Vetor Máximo

Aqui, optamos pela versão iterativa desse algoritmo, com custo O(n^2) para todos os casos. O número de instruções executadas para uma quantidade de elementos grande é bem menor que o estimado por sua notação (veja Figura 8), provavelmente por causa das otimizações do compilador. Seu tempo ficou bem próximo do tempo do SelectionSort, que também é O(n^2) (Tabela 8).

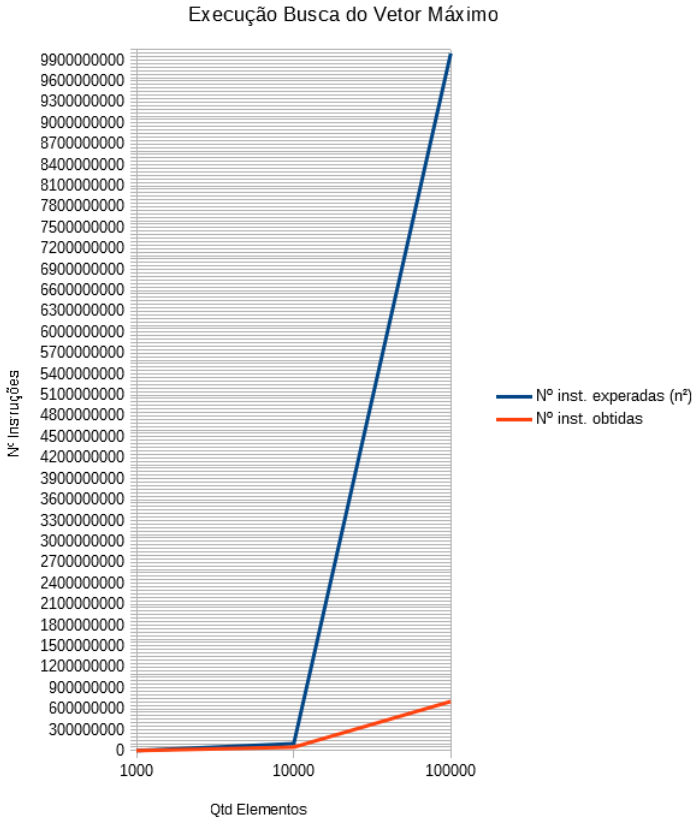

Figura 8: Gráfico que mostra o número de linhas executadas pelo algoritmo Busca Vetor Máximo e o comportamento previsto pela análise matemática.

Tabela 8: Tempo de Execução do Algoritmo Busca Vetor Máximo

Qtd Elementos	Tempo
1000	0.001000
10000	0.150000
100000	13.973000

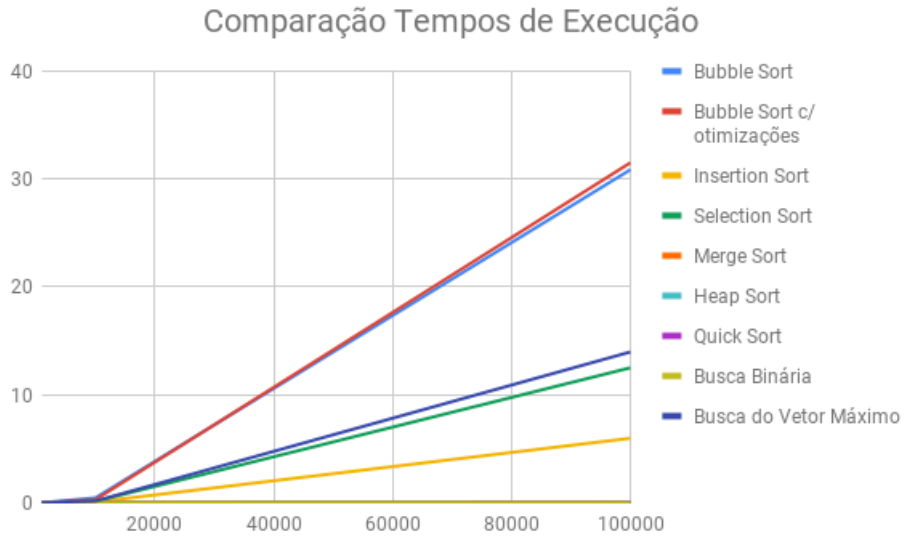

Figura 9: Gráfico que mostra a comparação dos tempos de execução dos algoritmos analisados neste trabalho. Repare que as linhas dos algoritmos recursivos nem chegam a aparecer, pois seus tempos são muito baixos, se comparados com os métodos iterativos.

3 Conclusão

Com a execução deste trabalho, pudemos perceber que, apesar de alguns métodos iterativos serem custosos, o compilador pode otimizar sua execução, melhorando assim seu desempenho. Outro ponto que se tornou visivel é que algoritmos recursivos podem ser bem mais ágeis que algoritmos iterativos (Figura 9). E, por fim, comprovamos a constatação de que nem toda linha tem o mesmo custo de

execução que a outra, pois os tempos de execução não foram totalmente diretamente proporcionais aos números de instruções.