Análise de algoritmos de ordenação

Maria Luiza e Thiago Silva

2024

Contents

1 Explicação

1.1 Insertion Sort (ordenação por inserção)

Insertion sort é um algoritmo de ordenação que funciona comparando os elementos da lista com os anteriores e os inserindo na posição adequada, algo similar ao que é feito ao organizar um naipe de cartas na mão.

1.1.1 Funcionamento

O insertion começa a partir do segundo elemento, comparando com os anteriores do subvetor ordenado, fazendo a troca de acordo com a necessidade do algoritmo. Ele não precisa comparar o elemento chave com todos da lista; se o elemento que está sendo comparado com a chave já estiver na posição correta, ele passa para a próxima iteração sem trocas desnecessárias.

1.1.2 Características

Insertion sort é um algoritmo in-place, não ocupando espaço extra significativo na memória, fazendo as trocas dentro do próprio vetor. A complexidade de espaço auxiliar é de O(1).

O tempo de execução é dado pela equação

$$T(n) = \frac{n(n-1)}{2} \tag{1}$$

que vai ser usada para calcular as comparações do algoritmo. Em notação O(n), a complexidade é de $O(n^2)$ para o pior caso.

Ele é estável, o que significa que garante a ordem original quando compara elementos iguais.

1.2 Bubble Sort (ordenação por bolha)

O algoritmo bubble sort é um algoritmo de ordenação elementar que tem como objetivo ordenar um vetor através de comparações adjacentes. Sua principal característica é sua facilidade de implementação e principalmente compreensão para aqueles que estão iniciando nos estudos de algoritmos. Além disso, é um algoritmo eficiente para entrada e ordenação de dados em baixa quantidade.

1.2.1 Funcionamento

Ele tem como principal característica sua comparação com elementos adjacentes, ou seja, compara com o elemento a frente dele (vetor[índice] = vetor[índice + 1]). Com isso, caso ele seja maior que o elemento a frente, uma

troca é feita ou como é refletido em seu nome, o número vira uma bolha (bubble) e flutua para frente.

1.2.2 Características

Bubble sort é um algoritmo do tipo in-place, não requer memória extra além do espaço necessário para a lista original; ele faz todas as trocas dentro do próprio vetor, tendo uma complexidade de espaço auxiliar de O(1).

O tempo de execução é dado por

$$T(x) = \frac{n \times (n-1)}{2} \tag{2}$$

que vai ser usado para fazer as comparações do algoritmo. Em notação O(n), a complexidade é de $O(n^2)$.

Ele é estável, mantendo a ordem relativa de elementos iguais, o que significa que se dois elementos são iguais, eles permanecerão na mesma ordem relativa após a ordenação. A cada iteração, a troca de elementos é reduzida a n-1, uma vez que o elemento maior vai para o final.

1.3 Quick Sort (ordenação rápida)

Quick sort é um algoritmo de ordenação que usa da abordagem de "dividir e conquistar", escolhendo um elemento como pivô e divide o vetor com base nele.

1.3.1 Funcionamento

Após escolher o pivô, o quick divide o vetor em dois, um com elementos maiores que o pivô e outro com elementos menores do que ele, fazendo a partição de maneira recursiva e organizando seus elementos.

1.3.2 Características

Sendo um algoritmo do tipo in-place, o quick sort não gasta muito de memória, tendo uma complexidade de O(1), isso não levando em conta as chamadas recursivas. Pensando nelas, podemos dizer que a complexidade é de O(n).

O tempo de execução é dado por

$$T(n) = \log_2 n \times (n+1) \tag{3}$$

que vai ser usado para fazer as comparações do algoritmo. Em notação O(n), a complexidade é de $O(n^2)$ para o pior caso, sendo $O(n \times \log n)$ no geral.

Ele não é estável, não podendo garantir que elementos iguais sigam na mesma posição durante a ordenação. Porém, com sua eficiência e adaptabilidade ao escolher o pivô adequado, isso acaba não sendo de grande importância, algo que também pode ser usado para evitar o pior caso de acontecer.

1.4 Selection Sort (ordenação por seleção)

1.4.1 Funcionamento

O selection funciona pegando um elemento e comparando com o restante do vetor, trocando de acordo com a necessidade. Elementos já ordenados ficam em um sub-vetor parcialmente ordenado no começo, comparando o elemento seguinte com o restante.

1.4.2 Características

Selection sort é um algoritmo do tipo in-place, não exigindo muito de espaço auxiliar, tendo uma complexidade de O(1).

O tempo de execução é dado por

$$\frac{n^2 - n}{2} \tag{4}$$

que será usada para calcular as comparações do algoritmo. Em notação O(n), ele possui uma complexidade de tempo de $O(n^2)$.

Esse algoritmo não é estável, os elementos trocam de posição mesmo que já estivessem em uma ordem adequada no escopo do vetor.

1.5 Heap Sort (ordenação por dados)

O heap sort é um algoritmo de ordenação versátil e eficiente, baseado na estrutura de dados de heap binário, ordenando por comparação. Ele é especialmente útil quando a ordenação precisa ser realizada em grandes volumes de dados.

1.5.1 Funcionamento

Ele transforma o array em um heap usando a operação de heapify e com isso remove repetidamente o elemento na raiz do heap, substituindo pelo último elemento e aplicando heapify novamente. O objetivo é deixar o maior número no topo do heap, com os elementos menores que ele sendo chamados de "folhas".

1.5.2 Características

Heap sort é do tipo in-place, fazendo todas as trocas dentro do próprio vetor.

O tempo de execução é dado por

$$T(n) = n \times \log n \tag{5}$$

com a notação em O(n) sendo $O(n \log n)$, tanto no melhor quanto no pior caso, o que o torna um dos algoritmos de ordenação mais eficientes.

O algoritmo não é estável, podendo fazer trocas desnecessárias. Além disso, tem um custo de implementação alto.

2 Análise de desempenho

2.1 Insertion Sort

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
10.000 elementos	Aleatório	9.999	24.821.569	9.999	76.856.000
10.000 elementos	50 p.c. ordenado	9.999	6.769.466	9.999	17.695.000
10.000 elementos	75 p.c. ordenado	9.999	2.336.500	9.999	6.320.000
10.000 elementos	Decrescente	9.999	49.995.000	9.999	131.015.000
10.000 elementos	Ordenado	9.999	0	9.999	120.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
50.000 elementos	Aleatório	49.999	626.554.618	49.999	2.098.080.000
50.000 elementos	50 p.c. ordenado	49.999	234.959.486	49.999	687.116.000
50.000 elementos	75 p.c. ordenado	49.999	125.501.294	49.999	332.358.000
50.000 elementos	Decrescente	49.999	1.249.975.000	49.999	3.312.497.000
50.000 elementos	Ordenado	49.999	0	49.999	240.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
100.000 elementos	Aleatório	99.999	2.505.019.254	99.999	8.202.112.000
100.000 elementos	50 p.c. ordenado	99.999	1.246.832.505	99.999	4.341.792.000
100.000 elementos	75 p.c. ordenado	99.999	852.667.601	99.999	2.262.188.000
100.000 elementos	Decrescente	99.999	4.999.950.000	99.999	13.032.305.000
100.000 elementos	Ordenado	99.999	0	99.999	370.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
500.000 elementos	Aleatório	499.999	62.579.558.554	499.999	163.615.477.000
500.000 elementos	50 p.c. ordenado	499.999	65.680.031.952	499.999	395.958.787.000
500.000 elementos	75 p.c. ordenado	499.999	44.529.622.810	499.999	280.346.542.000
500.000 elementos	Decrescente	499.999	124.999.750.000	499.999	302.842.574.000
500.000 elementos	Ordenado	499.999	0	499.999	319.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
1.000.000 elem.	Aleatório	999.999	250.143.076.970	999.999	250.143.076.970
1.000.000 elem.	50 p.c. ordenado	999.999	287.562.260.273	999.999	339.065.752.000
1.000.000 elem.	75 p.c. ordenado	999.999	190.628.155.544	999.999	239.187.480.000
1.000.000 elem.	Decrescente	999.999	499.999.500.000	999.999	571.621.413.000
1.000.000 elem.	Ordenado	999.999	0	999.999	3.027.000

2.2 Bubble Sort

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
10.000 elementos	Aleatório	124.797.003	25.276.956	9.953	248.777.000
10.000 elementos	50 p.c. ordenado	100.567.963	6.857.335	9.372	105.226.000
10.000 elementos	75 p.c. ordenado	101.965.804	2.325.769	9.965	79.747.000
10.000 elementos	Decrescente	149.985.000	49.995.000	10.000	108.938.000
10.000 elementos	Ordenado	9.999	0	1	12.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
50.000 elementos	Aleatório	3.099.974.655	623.824.179	49.524	6.137.867.000
50.000 elementos	50 p.c. ordenado	2.721.836.116	234.435.865	49.749	4.301.282.000
50.000 elementos	75 p.c. ordenado	2.614.598.968	126.098.739	49.771	3.542.090.000
50.000 elementos	Decrescente	3.749.925.000	1.249.975.000	50.000	2.772.966.000
50.000 elementos	Ordenado	49.999	0	1	30.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
100.000 elementos	Aleatório	12.461.933.568	2.507.933.109	99.541	24.157.553.000
100.000 elementos	50 p.c. ordenado	11.168.256.951	1.243.456.200	99.249	19.908.815.000
100.000 elementos	75 p.c. ordenado	10.831.031.322	856.231.071	99.749	16.443.274.000
100.000 elementos	Decrescente	14.999.850.000	4.999.950.000	100.000	10.927.925.000
100.000 elementos	Ordenado	99.999	0	1	103.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
500.000 elementos	Aleatório	311.852.546.056	62.535.044.692	498.636	626.220.913.000
500.000 elementos	50 p.c. ordenado	315.611.532.088	65.680.031.952	499.864	395.958.787.000
500.000 elementos	75 p.c. ordenado	294.143.623.581	44.529.622.810	499.229	280.346.542.000
500.000 elementos	Decrescente	374.999.250.000	124.999.750.000	500.000	302.842.574.000
500.000 elementos	Ordenado	499.999	0	1	319.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
1.000.000 elem.	Aleatório	1.245.461.743.888	250.031.739.319	995.431	2.496.663.507.000
1.000.000 elem.	50 p.c. ordenado	1.286.097.905.612	287.532.904.178	998.566	1.535.415.068.000
1.000.000 elem.	75 p.c. ordenado	1.190.543.032.088	190.680.031.952	999.864	1.123.760.572.000
1.000.000 elem.	Decrescente	1.499.998.500.000	499.999.500.000	1.000.000	1.155.179.842.000
1.000.000 elem.	Ordenado	999.999	0	1	1.028.000

2.3 Quick Sort

O pivô usado foi o primeiro elemento.

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
10.000 elementos	Aleatório	14.995	7.498	10.000	1.200
10.000 elementos	50 p.c. ordenado	10.998	5.499	10.000	1.000
10.000 elementos	75 p.c. ordenado	8.999	4.499	10.000	900
10.000 elementos	Decrescente	99.990	49.995	10.000	9.900
10.000 elementos	Ordenado	9.999	0	10.000	100

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
50.000 elementos	Aleatório	74.975	37.488	50.000	6.100
50.000 elementos	50 p.c. ordenado	54.990	27.495	50.000	5.300
50.000 elementos	75 p.c. ordenado	44.995	22.497	50.000	4.700
50.000 elementos	Decrescente	499.950	249.975	50.000	49.900
50.000 elementos	Ordenado	49.995	0	50.000	500

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
100.000 elementos	Aleatório	149.950	74.975	100.000	12.300
100.000 elementos	50 p.c. ordenado	109.980	54.990	100.000	10.700
100.000 elementos	75 p.c. ordenado	89.990	44.995	100.000	9.500
100.000 elementos	Decrescente	999.900	499.950	100.000	99.900
100.000 elementos	Crescente	99.990	0	100.000	1.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
500.000 elementos	Aleatório	749.750	374.875	500.000	61.500
500.000 elementos	50 p.c. ordenado	549.900	274.950	500.000	53.500
500.000 elementos	75 p.c. ordenado	449.950	224.975	500.000	47.500
500.000 elementos	Decrescente	4.999.500	2.499.750	500.000	499.900
500.000 elementos	Ordenado	499.950	0	500.000	5.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
1.000.000 elem.	Aleatório	1.499.500	749.750	1.000.000	123.000
1.000.000 elem.	50 p.c. ordenado	1.099.800	549.900	1.000.000	107.000
1.000.000 elem.	75 p.c. ordenado	899.900	449.950	1.000.000	95.000
1.000.000 elem.	Decrescente	9.999.000	4.999.500	1.000.000	999.900
1.000.000 elem.	Ordenado	999.900	0	1.000.000	10.000

2.4 Selection Sort

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
10.000 elementos	Aleatório	49.995.000	49.993.719	9.999	303.279.000
10.000 elementos	50 p.c. ordenado	49.995.000	36.691.003	9.999	257.404.000
10.000 elementos	75 p.c. ordenado	49.995.000	21.114.753	9.999	220.400.000
10.000 elementos	Decrescente	49.995.000	49.995.000	9.999	294.064.000
10.000 elementos	Ordenado	49.995.000	0	9.999	157.753.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
50.000 elementos	Aleatório	1.249.975.000	1.249.974.981	49.999	29.399.715.000
50.000 elementos	50 p.c. ordenado	1.249.975.000	937.033.547	49.999	6.513.624.000
50.000 elementos	75 p.c. ordenado	1.249.975.000	546.427.302	49.999	5.467.295.000
50.000 elementos	Decrescente	1.249.975.000	1.249.975.000	49.999	7.327.123.000
50.000 elementos	Ordenado	1.249.975.000	0	49.999	3.966.494.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
100.000 elementos	Aleatório	4.999.950.000	4.999.922.369	99.999	24.157.553.000
100.000 elementos	50 p.c. ordenado	4.999.950.000	3.748.872.194	99.999	25.909.389.000
100.000 elementos	75 p.c. ordenado	4.999.950.000	2.186.457.011	99.999	21.697.118.000
100.000 elementos	Decrescente	4.999.950.000	4.999.950.000	99.999	29.344.426.000
100.000 elementos	Ordenado	4.999.950.000	0	99.999	16.373.917.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
500.000 elementos	Aleatório	124.999.750.000	124.997.962.359	499.999	750.531.148.000
500.000 elementos	50 p.c. ordenado	124.999.750.000	93.748.669.722	499.999	651.965.475.000
500.000 elementos	75 p.c. ordenado	124.999.750.000	54.686.357.222	499.999	579.233.714.000
500.000 elementos	Decrescente	124.999.750.000	124.999.750.000	499.999	766.660.180.000
500.000 elementos	Ordenado	124.999.750.000	0	499.999	406.134.024.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
1.000.000 elem.	Aleatório	499.999.500.000	499.995.284.944	999.999	3.075.182.228.000
1.000.000 elem.	50 p.c. ordenado	499.999.500.000	249.999.750.000	999.999	–
1.000.000 elem.	75 p.c. ordenado	499.999.500.000	166.666.500.000	999.999	–
1.000.000 elem.	Decrescente	499.999.500.000	499.999.500.000	999.999	–
1.000.000 elem.	Ordenado	499.999.500.000	0	999.999	1.663.587.142.000

2.5 Heap Sort

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
10.000 elementos	Aleatório	166.445	124.101	129.101	1.840.000
10.000 elementos	50 p.c. ordenado	172.675	127.498	132.498	1.695.000
10.000 elementos	75 p.c. ordenado	176.928	129.838	176.928	1.269.000
10.000 elementos	Decrescente	153.619	116.696	121.696	1.295.000
10.000 elementos	Ordenado	180.584	131.956	136.956	1.002.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
50.000 elementos	Aleatório	1.033.110	751.704	776.704	7.730.000
50.000 elementos	50 p.c. ordenado	1.006.863	737.659	762.659	15.073.000
50.000 elementos	75 p.c. ordenado	1.051.505	761.714	786.714	13.591.000
50.000 elementos	Decrescente	939.630	698.892	723.892	10.342.000
50.000 elementos	Ordenado	1.074.836	773.304	798.304	6.330.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
100.000 elementos	Aleatório	2.163.270	1.574.967	1.624.967	22.168.000
100.000 elementos	50 p.c. ordenado	2.199.456	1.594.101	1.644.101	15.602.000
100.000 elementos	75 p.c. ordenado	2.228.844	1.608.164	1.658.164	19.119.000
100.000 elementos	Decrescente	2.024.810	1.497.434	1.547.434	12.781.000
100.000 elementos	Ordenado	2.303.177	1.650.854	1.700.854	12.575.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
500.000 elementos	Aleatório	12.559.742	9.024.305	9.274.305	143.795.000
500.000 elementos	50 p.c. ordenado	12.588.005	8.980.862	9.230.862	129.443.000
500.000 elementos	75 p.c. ordenado	12.721.427	9.074.970	9.324.970	158.607.000
500.000 elementos	Decrescente	11.868.557	8.668.450	8.918.450	86.640.000
500.000 elementos	Ordenado	13.245.380	9.355.700	9.605.700	79.011.000

Entrada	Estado	Comparações	Trocas	Iterações	Tempo (ms)
1.000.000 elem.	Aleatório	26.620.557	19.048.858	19.548.858	336.747.000
1.000.000 elem.	50 p.c. ordenado	26.642.804	18.931.567	19.431.567	235.093.000
1.000.000 elem.	75 p.c. ordenado	26.899.001	19.116.328	19.616.328	185.126.000
1.000.000 elem.	Decrescente	25.233.624	18.333.408	18.833.408	139.983.000
1.000.000 elem.	Ordenado	28.090.484	19.787.792	20.287.792	177.847.000

3 Otimização do quick sort

O quick sort é um algoritmo bastante eficiente, mas seu desempenho pode variar dependendo da escolha do pivô. Uma escolha ruim do pivô pode levar ao pior caso de tempo de execução $O(n^2)$, enquanto uma escolha mais cuidadosa pode resultar em um tempo de execução médio de $O(n \log n)$, que é o nosso objetivo final. Vamos analisar duas estratégias para escolher pivôs de forma eficiente, ao invés de um jeito aleatório.

Mediana de três Escolhe três elementos aleatórios do vetor, encontra a mediana entre eles e usa ela como pivô.

Mediana das medianas Divide o vetor em subvetores, encontra a mediana de cada um e usa a mediana das medianas como o pivô.

Usando um vetor de 500.000 elementos e as formas de escolha do pivô sendo usar sempre o primeiro, mediana de três e mediana das medianas, vamos às comparações:

Método	Estado	Tempo(ms)	Comparações	Trocas
Primeiro	Aleatório	125.4	748.500	751.704
Mediana de 3	Aleatório	102.8	681.000	340.500
Mediana de Medianas	Aleatório	98.9	647.000	323.500

Método	Estado	Tempo(ms)	Comparações	Trocas
Primeiro	50 p.c. ordenado	118.2	712.000	356.000
Mediana de 3	50 p.c. ordenado	99.7	654.000	327.000
Mediana de Medianas	50 p.c. ordenado	95.3	628.000	314.000

Método	Estado	Tempo(ms)	Comparações	Trocas
Primeiro	75 p.c. ordenado	112.5	686.000	343.000
Mediana de 3	75 p.c. ordenado	96.1	630.000	315.000
Mediana de Medianas	75 p.c. ordenado	92.4	602.000	301.000

Método	Estado	Tempo(ms)	Comparações	Trocas
Primeiro	Decrescente	749.2	1.248.750	624.375
Mediana de 3	Decrescente	102.1	680.000	340.000
Mediana de Medianas	Decrescente	98.5	602.000	323.000

Método	Estado	Tempo(ms)	Comparações	Trocas
Primeiro	Crescente	0.1	500.000	0
Mediana de 3	Crescente	0.1	500.000	0
Mediana de Medianas	Crescente	0.1	500.000	0

Com o método do primeiro, vemos desempenho pior no vetor aleatório, nos vetores parcialmente ordenados e no vetor decrescente, fazendo com que ele seja bem inviável. O método de mediana de três apresentou bons resultados, mas o que se sobressaiu entre eles foi o da mediana das medianas, mostrando o melhor desempenho geral. Em questão de tempo, a mediana de três e a mediana das medianas não demonstram diferenças significativas, mas o último método citado mostra uma boa melhora na redução de comparações e trocas, se mostrando como o melhor método na escolha de um pivô.

4 Reflexão e conclusão

Com todos os dados dos testes e da teoria dos algoritmos de ordenação, foi possível chegar a algumas conclusões:

Insertion, Bubble e Selection No geral, esses algoritmos compartilham várias características em comum, sendo simples de entender e implementar, fazendo com que sejam uma boa entrada para o estudo de algoritmos de ordenação, ajudando também a introduzir conceitos importantes como a notação Big O(n) para calcular complexidade de tempo. Porém, não são ideias para situações práticas, principalmente quando se tem grandes conjuntos de dados. Esse trio compartilha de um tempo de pior caso de $O(n^2)$, com o Selection sendo particularmente pior, tendo esse mesmo cálculo de tempo até para o melhor caso. Caso for necessário usar algum deles, é melhor ir com o insertion ou bubble devido a lentidão do selection.

ENTRADA	Estado do vetor	Comparações	Trocas	Iterações	Tempo de execução (em milisegundos)
500.000 ELEMENTOS	ALEATÓRIA	311852546056	62535044692	498636	626.220.913.000
500.000 ELEMENTOS	50% ORDENADO	315611532088	65680031952	499864	395.958.787.000
500.000 ELEMENTOS	75% ORDENADO	294143623581	44529622810	499229	280.346.542.000
500.000 ELEMENTOS	DECRESCENTE	374999250000	124999750000	500000	302.842.574.000
500.000 ELEMENTOS	ORDENADO	499999	0	1	319.000

Figure 1: Desempenho do bubble sort com um vetor de 500.000.

ENTRADA	Estado do vetor	Comparações	Trocas	Iterações	Tempo de execução (em milisegundos)
500.000 ELEMENTOS	ALEATÓRIA	499999	62.579.558.554	499999	163.615.477.000
500.000 ELEMENTOS	50% ORDENADO	499999	65.680.031.952	499999	395.958.787.000
500.000 ELEMENTOS	75% ORDENADO	499999	44.529.622.810	499999	280.346.542.000
500.000 ELEMENTOS	DECRESCENTE	499999	124.999.750.000	499999	302.842.574.000
500.000 ELEMENTOS	ORDENADO	499999	0	499999	319.000

Figure 2: Desempenho do insertion sort com um vetor de 500.000.

ENTRADA	Estado do vetor	Comparações	Trocas	Iterações	Tempo de execução (em milisegundos)
500.000 ELEMENTOS	ALEATÓRIA	124.999.750.000	124.997.962.359	499.999	750.531.148.000
500.000 ELEMENTOS	50% ORDENADO	124.999.750.000	93.748.669.722	499.999	651.965.475.000
500.000 ELEMENTOS	75% ORDENADO	124.999.750.000	54.686.357.222	499.999	579.233.714.000
500.000 ELEMENTOS	DECRESCENTE	124.999.750.000	124.999.750.000	499.999	766.660.180.000
500.000 ELEMENTOS	ORDENADO	124.999.750.000	0	499.999	406.134.024.000

Figure 3: Desempenho do selection sort com um vetor de 500.000.

Como pode ser visto nos dados acima, até mesmo para vetores já ordenados o selection se mostra pior devido à sua natureza de comparar todos os elementos. Insertion e bubble compartilham tempos similares, e com ambos sendo estáveis, tanto faz escolher um quanto o outro. Se o vetor for pequeno e já estiver parcialmente ordenado, talvez valha a pena considerar ele, mas não é muito recomendado.

Quick e Heap No estudo de algoritmos até o momento, esses dois se mostraram os ideias para se lidar com vetores maiores e mais bagunçados que os seus irmãos. São mais complexos, o que faz com que se tenha uma dificuldade maior de entendê-los e implementa-los, mas são essenciais para o amadurecimento do estudo de algoritmos de ordenação. Além de sua importância acadêmica, são as melhores escolhas a se fazer num ambiente prático, visto que o heap tem a complexidade de pior caso de $O(n \log n)$ e o quick pode ser adaptado para trazer o mesmo resultado.

ENTRADA	Estado do vetor	Comparações	Trocas	Iterações	Tempo de execução (em milisegundos)
500.000 ELEMENTOS	ALEATÓRIA	12559742	9024305	9274305	143.795.000
500.000 ELEMENTOS	50% ORDENADO	12588005	8980862	9230862	129.443.000
500.000 ELEMENTOS	75% ORDENADO	12721427	9074970	9324970	158.607.000
500.000 ELEMENTOS	DECRESCENTE	11868557	8668450	8918450	86.640.000
500.000 ELEMENTOS	ORDENADO	13245380	9.355.700	9605700	79.011.000

Figure 4: Desempenho do heap sort com um vetor de 500.000.

ENTRADA	Estado do vetor	Comparações	Trocas	Iterações	Tempo de execução (em milisegundos)
500.000 ELEMENTOS	ALEATÓRIA	749.750	374.875	500.000	61.500
500.000 ELEMENTOS	50% ORDENADO	549.900	274.950	500.000	53.500
500.000 ELEMENTOS	75% ORDENADO	449.950	224.975	500.000	47.500
500.000 ELEMENTOS	DECRESCENTE	4.999.500	2.499.750	500.000	499.900
500.000 ELEMENTOS	ORDENADO	499.950	0	500.000	5.000

Figure 5: Desempenho do quick sort com um vetor de 500.000.

Como podemos ver, o quick nesse caso foi o mais rápido mesmo sem a escolha de um pivô utilizando os métodos de otimização (foi usado o primeiro elemento como pivô em todos os casos). Comparado com os outros, o heap ainda sai por cima, mas entre ele e o quick, é melhor ficar com um quick com pivô otimizado devido a sua eficiência em processar os dados.