Filipe Matos Ferracini

RGA: 2020.1907.052-6

ALGORITMOS E PROGRAMAÇÃO II – T04:

Comparação de algoritmos de ordenação

Primeiramente, cabe apontar que o objetivo do código anexo é calcular a velocidade de execução de cinco diferentes métodos de ordenação (Bubble Sort – Ordenação Bolha, Insertion Sort – Ordenação por inserção, Selection Sort – Ordenação por inserção, Merge Sort – Ordenação por intercalação e QuickSort), para ordenar na ordem crescente um dado vetor.

1. MÉTODOS

Os métodos de ordenação podem ser divididos em dois grupos:

a) Métodos simples, que são mais adequados para pequenos vetores e, portanto, possuem códigos mais simples, e;
b) Métodos eficientes, que são adequados para quantidades maiores de dados, que, por sua vez, possuem códigos mais complexos, mas que requerem menor número de comparações.

Primeiro, no método Bolha, o intuito do código é fazer com que o maior valor fique à direita do elemento analisado. Assim, o vetor deve ser percorrido n vezes até a enésima posição. Portanto, é evidente que sua complexidade ser **$O(n^2)$**.

De forma similar, o método de inserção se dá por comparação e inserção. Conforme o vetor é percorrido, o algoritmo vai organizando um a um os valores de modo que o menor elemento fique sempre à esquerda do próximo. Assim, apesar de ser um algoritmo mais eficiente, sua complexidade é de **$O(n^2)$**. No melhor caso, ou seja, no vetor crescente, a complexidade cai para **$O(n)$**.

Finalmente, o último dos métodos simples analisados, o método de seleção parte da premissa que a cada iteração, o menor elemento ficará na primeira posição. Uma vez o menor valor alocado, percorre-se o vetor inteiro para encontrar o segundo menor valor e assim sucessivamente. Assim, para todos os seus casos, sua complexidade é de **$O(n^2)$**.

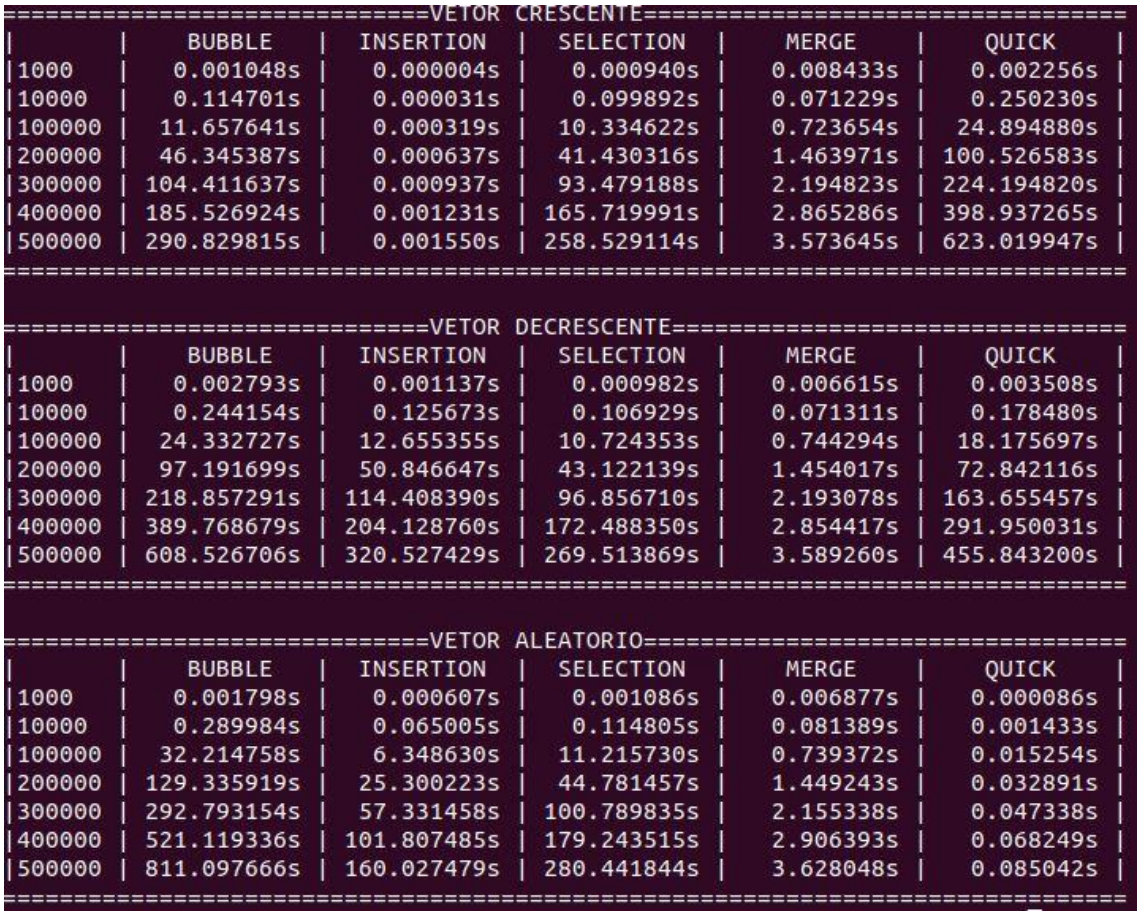
No grupo de métodos eficientes, o método de intercalação (merge sort) parte do princípio do “dividir para conquistar”. Ou seja, dividindo o vetor em partes cada vez menores até que possam ser analisadas uma a uma, onde a resolução se torna simples. Após esta analise, os valores são intercalados de maneira ordenada. Assim, para todos os casos sua complexidade é de **$O(n \log n)$**.

E, finalmente, o quicksort utiliza a mês a estratégia do método de intercalação de “dividir para conquistar”: Através da determinação de um pivô central, o problema é divido a cada iteração em dois problemas menores, sempre a partir de um valor intermediário. Este processo é repetido até que se obtenha uma lista unitária e, consequentemente, ordenada. O quicksort, entretanto possui como pior caso complexidade de **$O(n^2)$** e, no melhor caso, **$O(n \log n)$**.

2. RESULTADOS

Todos os métodos de ordenação foram testados para vetores de ordem crescente, vetores de ordem decrescente e, finalmente, para vetores de ordem aleatória gerados segundo o algoritmo de Fisher-Yates. Quanto aos tamanhos, foram gerados cinco tamanho para cada ordem inicial: 1000, 10000, 100000, 200000, 300000, 400000, 500000.

Os resultados obtidos foram os que seguem:



VETOR CRESCENTE

	BUBBLE	INSERTION	SELECTION	MERGE	QUICK
1000	0.001048s	0.000004s	0.000940s	0.008433s	0.002256s
10000	0.114701s	0.000031s	0.099892s	0.071229s	0.250230s
100000	11.657641s	0.000319s	10.334622s	0.723654s	24.894880s
200000	46.345387s	0.000637s	41.430316s	1.463971s	100.526583s
300000	104.411637s	0.000937s	93.479188s	2.194823s	224.194820s
400000	185.526924s	0.001231s	165.719991s	2.865286s	398.937265s
500000	290.829815s	0.001550s	258.529114s	3.573645s	623.019947s

VETOR DECRESCENTE

	BUBBLE	INSERTION	SELECTION	MERGE	QUICK
1000	0.002793s	0.001137s	0.000982s	0.006615s	0.003508s
10000	0.244154s	0.125673s	0.106929s	0.071311s	0.178480s
100000	24.332727s	12.655355s	10.724353s	0.744294s	18.175697s
200000	97.191699s	50.846647s	43.122139s	1.454017s	72.842116s
300000	218.857291s	114.408390s	96.856710s	2.193078s	163.655457s
400000	389.768679s	204.128760s	172.488350s	2.854417s	291.950031s
500000	608.526706s	320.527429s	269.513869s	3.589260s	455.843200s

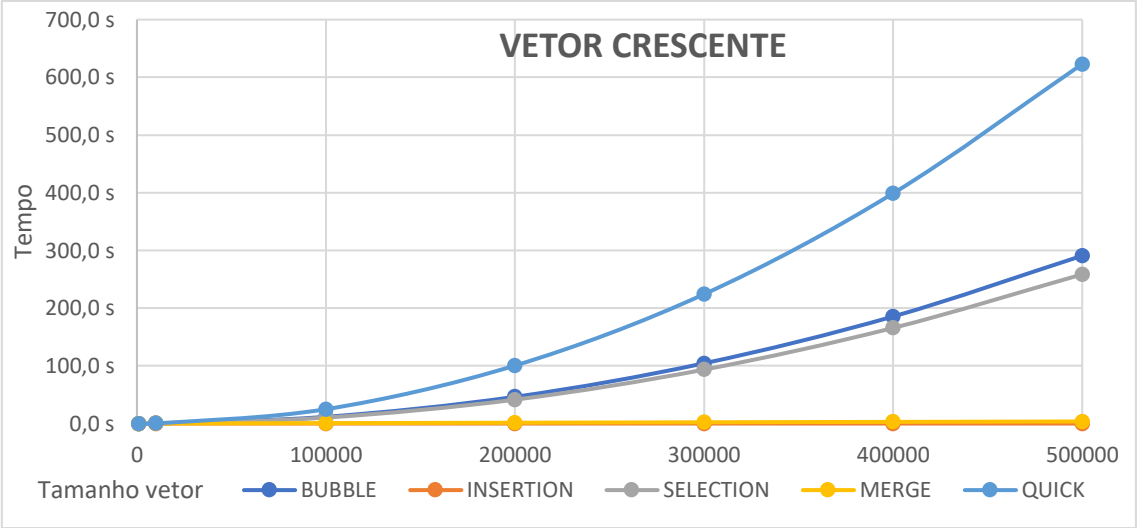
VETOR ALEATORIO

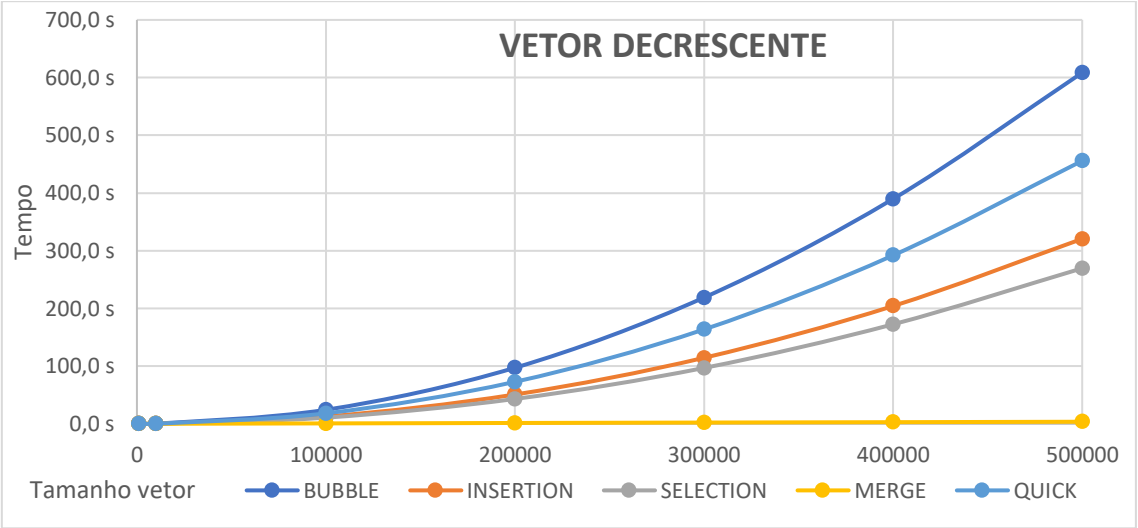
	BUBBLE	INSERTION	SELECTION	MERGE	QUICK
1000	0.001798s	0.000607s	0.001086s	0.006877s	0.000086s
10000	0.289984s	0.065005s	0.114805s	0.081389s	0.001433s
100000	32.214758s	6.348630s	11.215730s	0.739372s	0.015254s
200000	129.335919s	25.300223s	44.781457s	1.449243s	0.032891s
300000	292.793154s	57.331458s	100.789835s	2.155338s	0.047338s
400000	521.119336s	101.807485s	179.243515s	2.906393s	0.068249s
500000	811.097666s	160.027479s	280.441844s	3.628048s	0.085042s

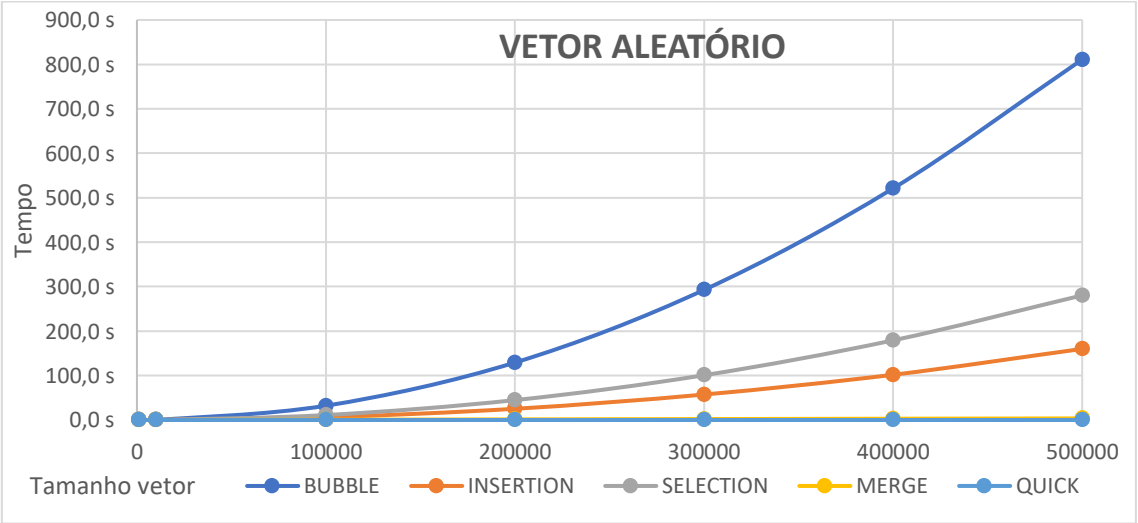
Cabe aqui, inclusive, ressaltar que para que a execução do programa tenha sucesso, é necessário habilitar um limite maior de alocação de memória, de 8mb para 32mb. Caso o mesmo não seja feito, o programa sofre segmentation fault, justamente por sofrer stack overlflow. Esta alteração do limite da shell resolve o problema e permite o sucesso da aplicação.

3. DISCUSSÃO

Para que a discussão seja mais facilmente visualizada, faz-se uso de gráficos de tempo x tamanho do vetor:


VETOR CRESCENTE
700,0 s
600,0 s
500,0 s
400,0 s
300,0 s
200,0 s
100,0 s
0,0 s
Tempo
0
100000
200000
300000
400000
500000
Tamanho vetor
BUBBLE
INSERTION
SELECTION
MERGE
QUICK


VETOR DECRESCENTE
700,0 s
600,0 s
500,0 s
400,0 s
300,0 s
200,0 s
100,0 s
0,0 s
Tempo
0
100000
200000
300000
400000
500000
Tamanho vetor
BUBBLE
INSERTION
SELECTION
MERGE
QUICK


VETOR ALEATÓRIO
900,0 s
800,0 s
700,0 s
600,0 s
500,0 s
400,0 s
300,0 s
200,0 s
100,0 s
0,0 s
Tempo
0
100000
200000
300000
400000
500000
Tamanho vetor
BUBBLE
INSERTION
SELECTION
MERGE
QUICK

Fica evidente que de modo geral a maior duração é sempre do método bolha, crescendo exponencialmente, quanto maior o tamanho do vetor. Semelhantemente, porém mais eficiente, o método de seleção está sempre próximo e com um comportamento similar, porém se prova um método mais eficiente que o método bolha, mesmo para vetores grandes.

Grande atenção deve ser dada ao método de inserção que, em seu pior caso, tem comportamento muito semelhante aos métodos de seleção e bolha, evidenciando sua complexidade de **O(n²)**. Entretanto, no caso seu melhor caso, que seria o de vetor crescente, sua eficiência de **O(n log n)** é evidenciada com valores menores que 0,002s para vetor de 500.000 posições.

Outro ponto de grande atenção deve ser dado ao quicksort que, apesar de ter resultado espantoso para um vetor aleatório de 500.000 posições, atingindo valores menor que 0,09s e provando sua eficiência de **O(n log n)**, tem péssimo comportamento em seus piores casos. Em particular no vetor de ordem crescente de 500.000, o método atinge o pior tempo e, no caso vetor decrescente de mesmo tamanho, também está entre os piores.

Finalmente, tratamos do método de intercalação: o merge sort. O método se prova altamente estável com valores abaixo dos 4s para todos vetores de 500.000 posições. Apesar de não ser o mais veloz em alguns casos, sua complexidade de **O(n log n)** em todos os casos é atestada.

4. CONCLUSÃO

Através da analise dos resultados e a plotagem dos gráficos é possível verificar que:

- Dentre os métodos simples, o método bolha se mostra o menos eficiente. Enquanto isso os métodos de seleção e de inserção se provam mais velozes (com a ressalva de que, no melhor caso, o método de inserção se provou o mais rápido;
- Dentre os métodos complexos, apesar de o quicksort se provar o método mais rápido para um vetor aleatório, seu pior caso obtém o pior tempo dentre os cinco métodos. Portanto, seu uso deve ocorrer mediante muita cautela;
- Finalmente, o método de inserção esteve sempre dentre os métodos mais velozes em todos os casos testes. O fato de o método se comportar de forma homogênea tanto no melhor quanto no maior teste, atesta a eficiência do algoritmo.