Organização e Recuperação da Informação

Métodos de Ordenação

Trabalho 1

Prof. Jander Moreira

Alunos:
Thales Eduardo Adair Menato – 407976
João Vitor Brandão Moreira – 407496

2º Semestre / 2012

Introdução

Como alunos do curso de Ciência da Computação vemos durante o curso a disciplina de Organização e Recuperação da Informação onde nos deparamos com a análise e métodos de resolução de problemas do dia a dia como, neste caso, a ordenação de dados. São diversos métodos apresentados onde entendemos a lógica por trás de cada um, seu melhor e pior caso, e como implementá-los.

Objetivos

Este trabalho possui o intuito de realizar a análise da performance de 3 métodos de ordenação, sendo que escolhemos para analisar o *BubbleSort*, *QuickSort* e *MergeSort*. A análise será realizada utilizando contadores para saber quantos movimentos ocorreram entre os registros e quantas comparações foram realizadas. Dessa forma poderemos visualizar graficamente o desempenho de cada um dos métodos e compará-los.

A análise será realizada da seguinte maneira:

- Definir quantidade de chaves do registro partindo de um registro com poucas chaves e ir aumentando gradualmente.
- Em cada quantidade de chaves definida, vamos criar um registro com chaves aleatórias e visualizar seu desempenho.
- Também será feito a análise do desempenho com chaves que já estão ordenadas.

Apesar de não ter sido pedido no trabalho também implementamos, utilizando a biblioteca *time.h* um cronômetro que inicia logo antes do método e finaliza assim que o mesmo termina nos dando um tempo de execução aproximado, apenas por motivo de curiosidade.

Métodos

Os métodos foram escolhidos de acordo com o que achamos mais interessante dentre os que estavam apresentados.

BubbleSort – o método do *BubbleSort* também conhecido como método "Bolha" é um dos algoritmos de ordenação mais simples. Sendo sua complexidade no pior caso de O(n^2) e de O(n) no melhor. Ele consiste em criar um laço externo que inicia com o numero de chaves + 1, é feito a comparação se o contador é maior que 0, se sim, ele entra no laço interno onde, desde a primeira posição do vetor, é comparado com sua posição seguinte, se a posição seguinte for maior que a atual, elas trocam suas posições até que o contador interno não seja mais menor que o contador externo, quando isso ocorre ele sai do laço interno, o contador externo decrementa. Em suma o contador externo vai decrementando desde o numero de elementos + 1 até 0 e o interno incrementa da primeira posição até a posição atual do externo, trocando as posições das chaves vizinhas.

Pela descrição podemos perceber que é um métodos que acabará realizando muitas comparações excessivas quando tivermos um vetor com um tamanho significativo.

Utilizamos o código disponível em:
http://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Bubble_sort

MergeSort – é um algoritmo de ordenação do tipo divisão e conquista, sua complexidade no pior e melhor caso é Θ(*n* log *n*). A lógica consiste em dividir o vetor em pedaços menores, classificar os pedaços menores chamando o método novamente, recursão, e então juntá-los novamente em um vetor ordenado.

Utilizamos o código disponível em:
http://www.ic.unicamp.br/~islene/mc102/aula22/mergesort.c

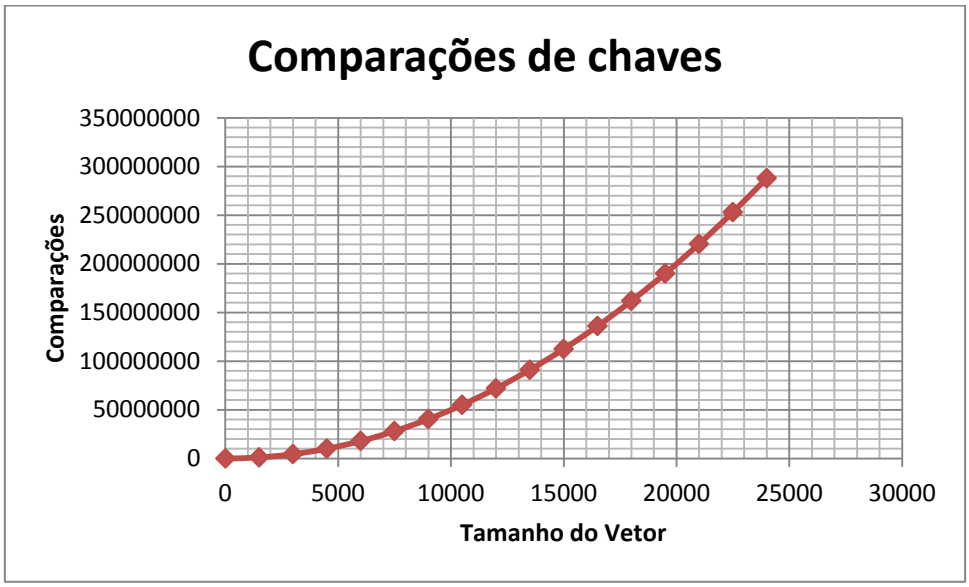
QuickSort – um método de ordenação muito rápido e eficiente, também utiliza a divisão e conquista, no pior caso sua complexidade é O(n^2) e no melhor caso O(n log n). Um dos elementos do vetor é escolhido, denominado pivô. Todos os elementos anteriores ao pivô serão rearranjados de modo que sejam sempre menores que ele e todos os elementos posteriores, maiores, no final das ordenações o pivô estará na sua posição final dentro do vetor, as duas sublistas geradas pelos elementos não ordenados antes e depois do pivô serão ordenadas, recursivamente, por este mesmo método.

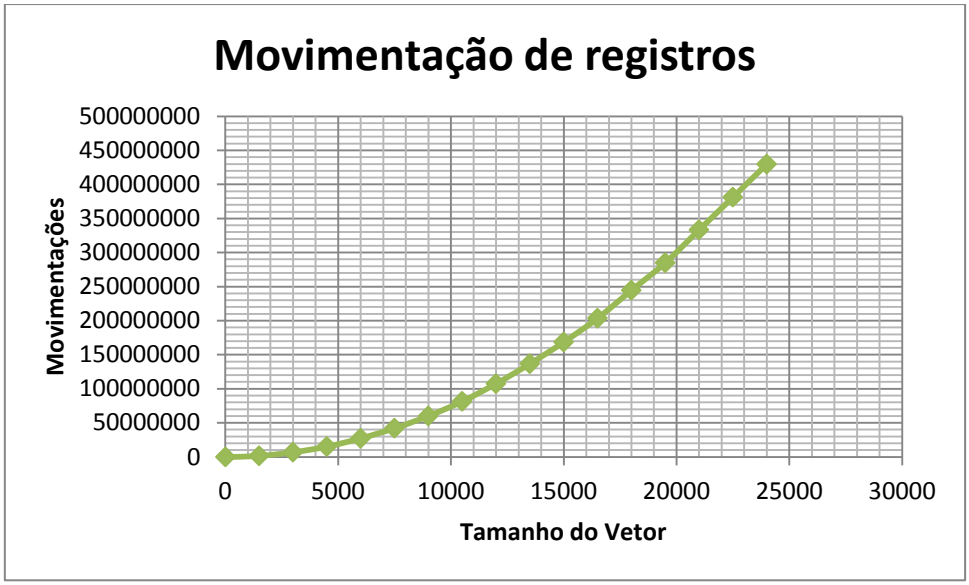
Utilizamos o código disponível em:
http://www.comp.dit.ie/rlawlor/Alg_DS/sorting/quickSort.c

Desempenho

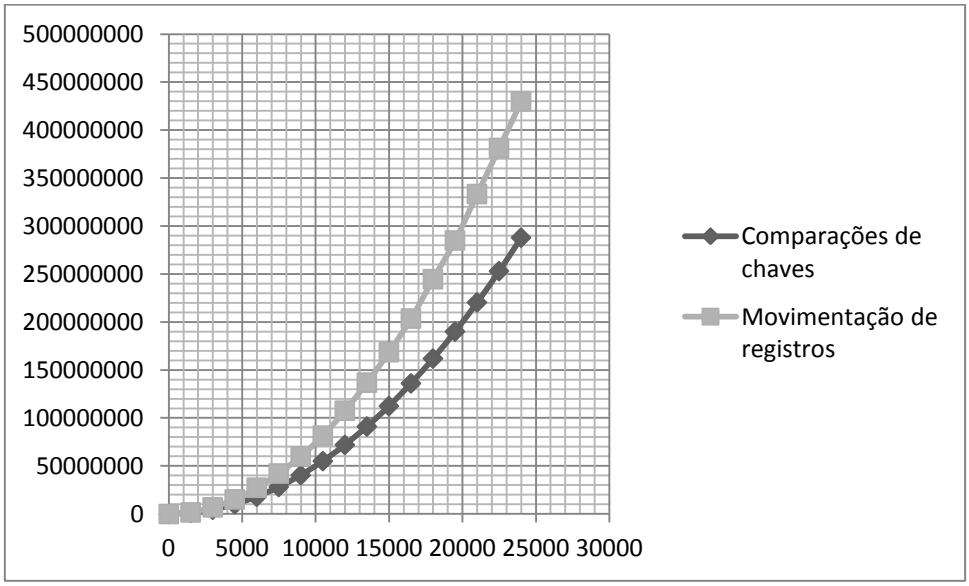
Método de ordenação BubbleSort

Chaves Aleatórias			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	42	45	0
1500	1123925	1645902	48
3000	4497174	6971121	197
4500	10108215	15214809	468
6000	17995347	27400806	864
7500	28112472	42195357	1349
9000	40490144	60171375	1947
10500	55110972	81651858	2673
12000	71987097	107651571	3531
13500	91115169	136843791	4474
15000	112461624	168654954	5553
16500	136108875	203683086	6742
18000	161985849	244707963	8114
19500	190091160	285054813	9461
21000	220480047	333434022	11060
22500	253100709	381445044	12708
24000	287980374	429925569	14561

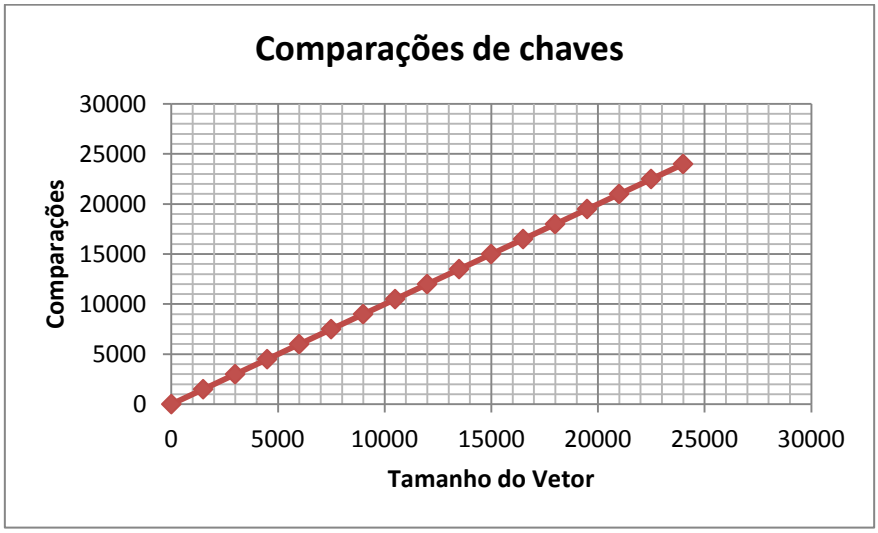


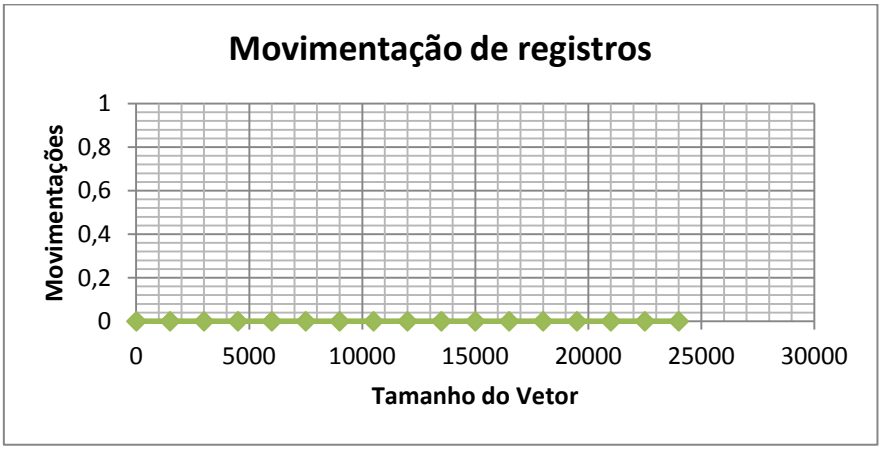

Movimentação de registros
Movimentações
500000000
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
0
5000
10000
15000
20000
25000
30000
Tamanho do Vetor


Tempo de Execução
tempo (milissegundos)
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
30000
Tamanho do Vetor

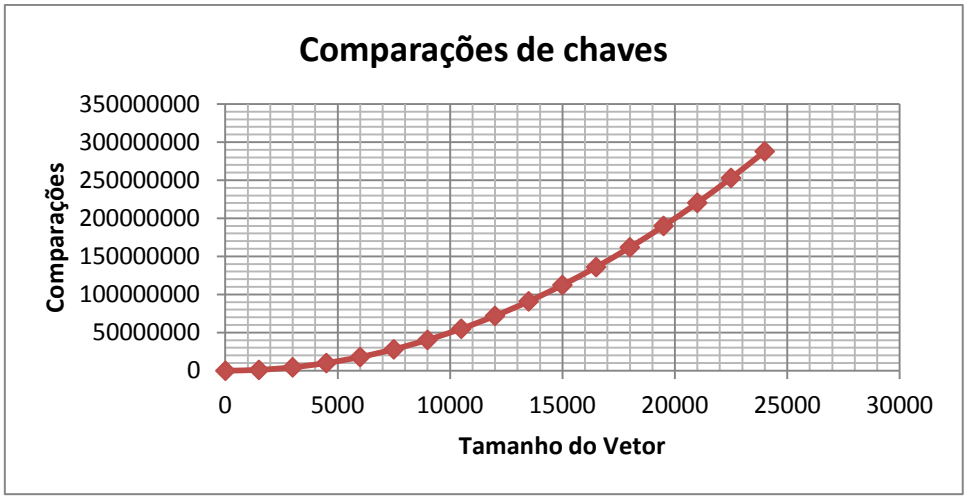

500000000
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
0
5000
10000
15000
20000
25000
30000
Comparações de chaves
Movimentação de registros

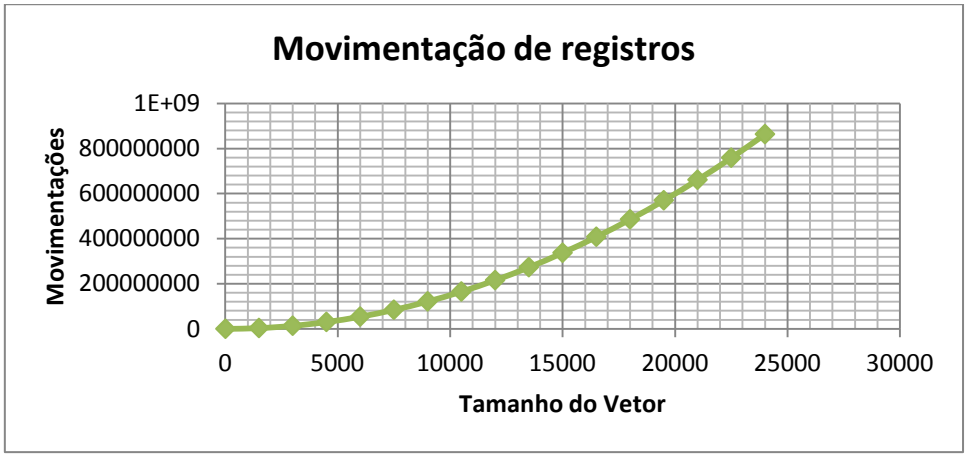
Chaves Ordenadas (crescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	9	0	0
1500	1499	0	0
3000	2999	0	0
4500	4499	0	0
6000	5999	0	0
7500	7499	0	0
9000	8999	0	0
10500	10499	0	0
12000	11999	0	0
13500	13499	0	0
15000	14999	0	0
16500	16499	0	0
18000	17999	0	0
19500	19499	0	0
21000	20999	0	0
22500	22499	0	0
24000	23999	0	0





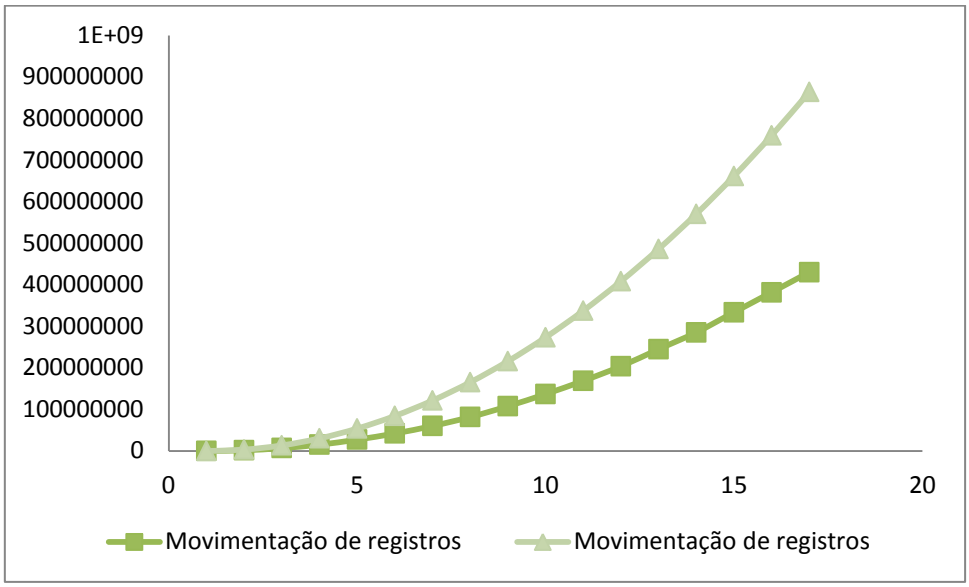
Chaves Ordenadas (decrescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	44	132	0
1500	1124249	3372747	81
3000	4498499	13495497	326
4500	10122749	30368247	761
6000	17996999	53990997	1370
7500	28121249	84363747	2150
9000	40495499	121486497	3109
10500	55119749	165359247	4238
12000	71993999	215981997	5566
13500	91118249	273354747	7046
15000	112492499	337477497	8696
16500	136116749	408350247	10519
18000	161990999	485972997	12564
19500	190115249	570345747	14813
21000	220489499	661468497	17141
22500	253113749	759341247	19712
24000	287987999	863963997	22602





Observações do BubbleSort

Como pudemos observar nos gráficos, quanto maior o tamanho do vetor a curva do grafo sobe exponencialmente, o que é um problema para aplicações que trabalham com vetores muito grandes. Podemos observar também que quando as chaves já estão ordenadas (crescente) não há nenhuma movimentação e a comparação é sempre o tamanho do vetor – 1. Quando fazemos o contrário, ordenamos de modo decrescente as comparações voltam a ser exponencial e o crescimento desta função se comparada às chaves aleatórias é maior como podemos ver no gráfico abaixo:



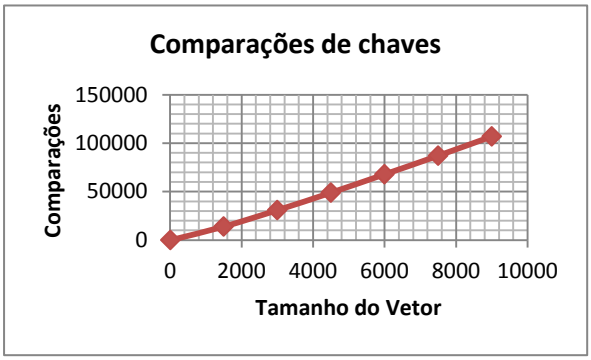
Verde claro representa a curva das chaves ordenadas decrescentemente.

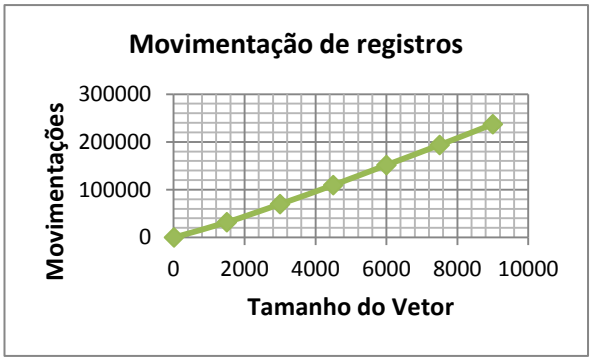
O tempo de execução passou a ser grande, evidente, logo com vetores de tamanho 4500, onde demora, aproximadamente, 0,5 segundos para ser executado. E continuou crescendo exponencialmente até 14 segundos na execução com 24000 chaves.

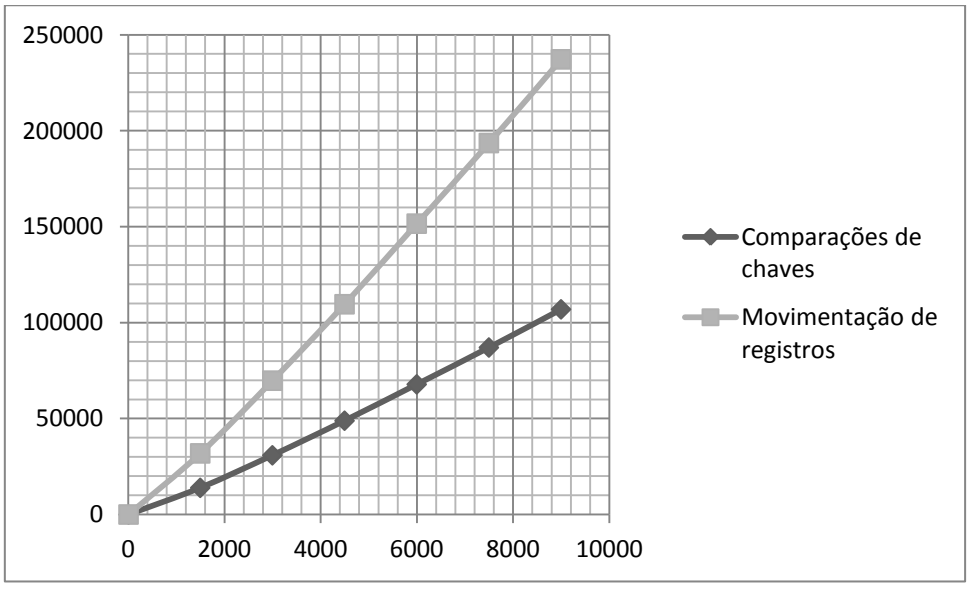
Desempenho

Método de ordenação MergeSort

Chaves Aleatórias			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	22	68	0
1500	13960	31904	2
3000	30911	69808	5
4500	48944	109616	12
6000	67884	151616	21
7500	87085	193616	34
9000	106969	237232	48







Chaves Ordenadas (crescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	19	68	0
1500	8288	31904	2
3000	18076	69808	5
4500	28148	109616	12
6000	39152	151616	21
7500	49432	193616	33
9000	60796	237232	46

Chaves Ordenadas (decrescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	15	68	0
1500	7664	31904	1
3000	16828	69808	4
4500	26660	109616	11
6000	36656	151616	21
7500	47376	193616	32
9000	57820	237232	46

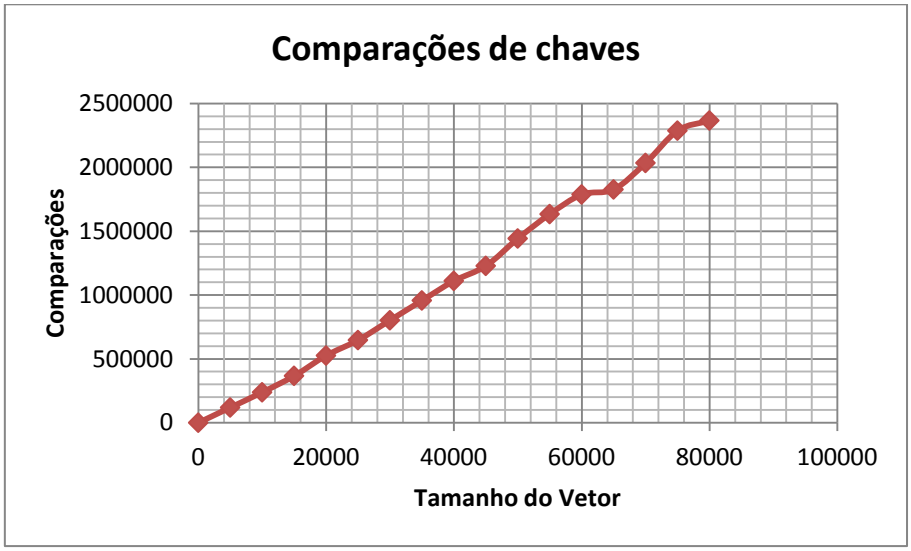
Observações do MergeSort

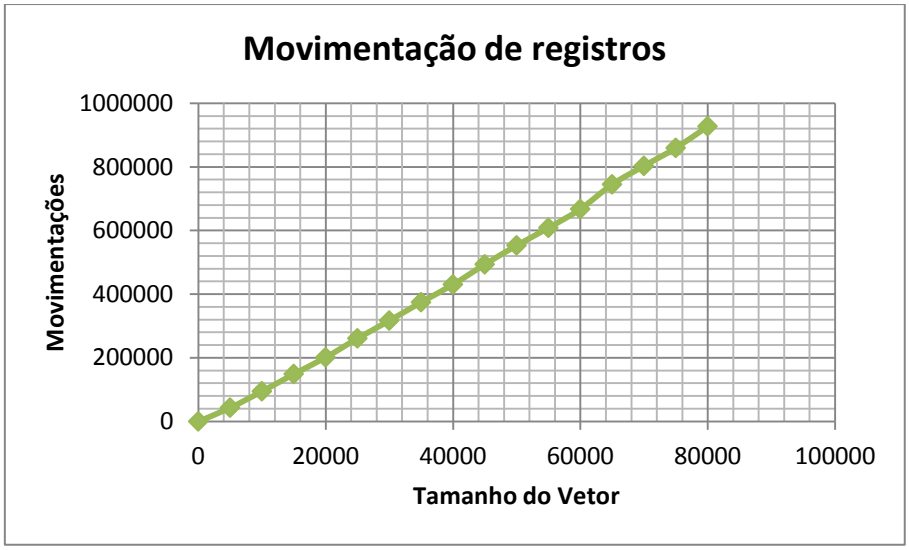
Podemos observar que o crescimento, nos três casos, é próximo de uma função linear. O crescimento do tempo de execução é uma curva exponencial, porém não cresce tão rapidamente e durante todas as execuções não ouve um aumento significativo que pudesse ser percebido pelo usuário. Podemos dizer que onde ocorreu maior número de comparações foi quando as chaves foram geradas aleatoriamente.

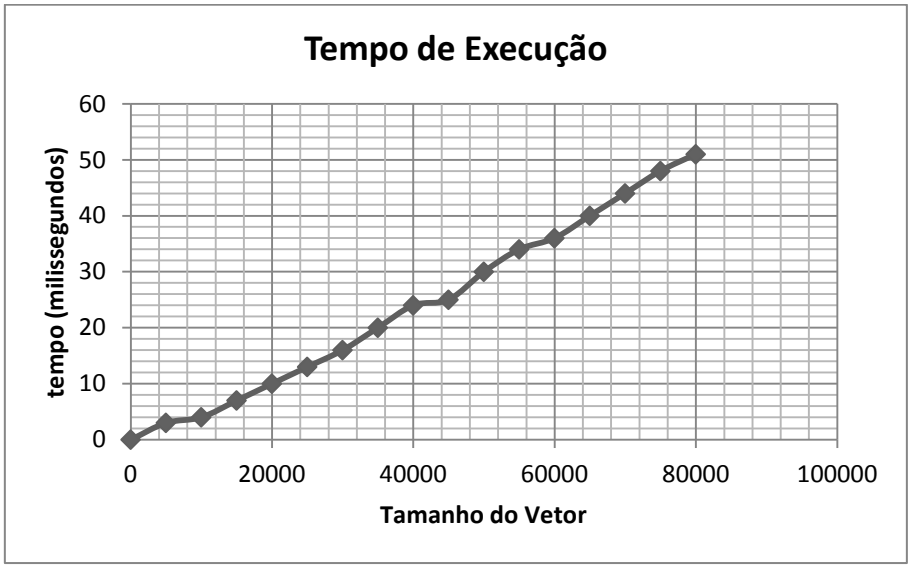
Desempenho

Método de ordenação QuickSort

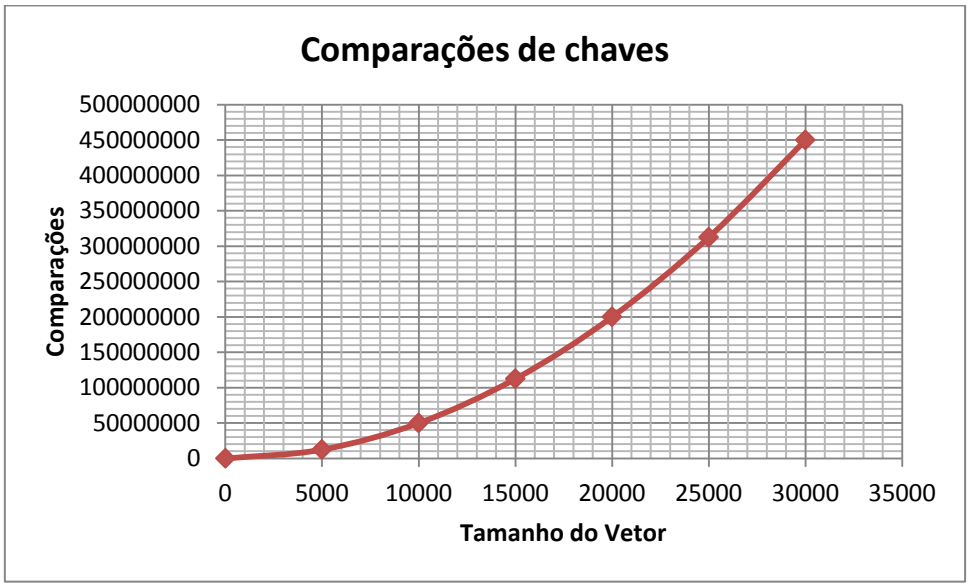
Chaves Aleatórias			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	61	30	0
5000	119150	43296	3
10000	238345	94713	4
15000	367134	149097	7
20000	526022	201555	10
25000	647124	261141	13
30000	802517	316989	16
35000	957415	374889	20
40000	1110862	430536	24
45000	1228425	493827	25
50000	1442815	553443	30
55000	1634079	608721	34
60000	1787587	667830	36
65000	1827219	745272	40
70000	2034644	803328	44
75000	2287625	859884	48
80000	2366785	927747	51

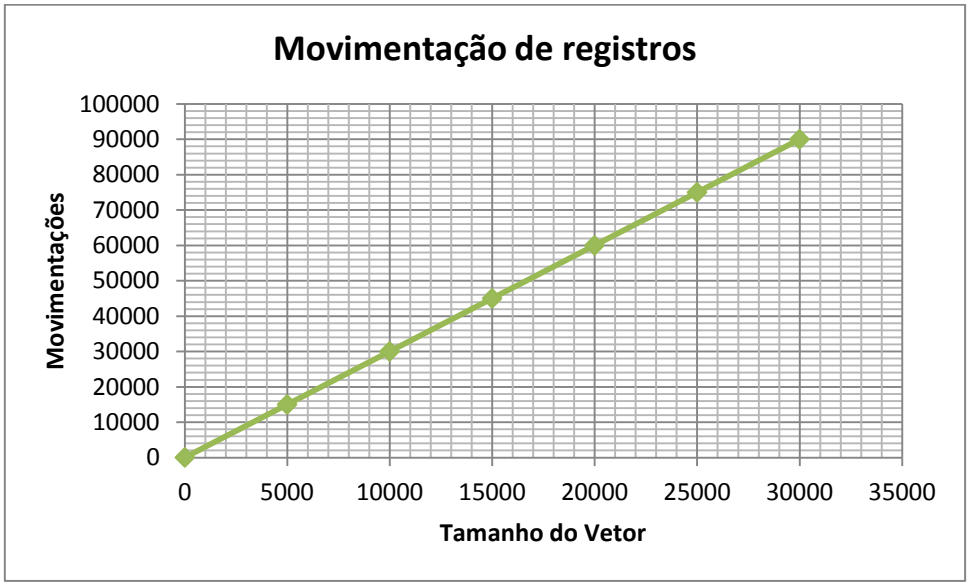



Movimentação de registros
Movimentações
1000000
800000
600000
400000
200000
0
0
20000
40000
60000
80000
100000
Tamanho do Vetor

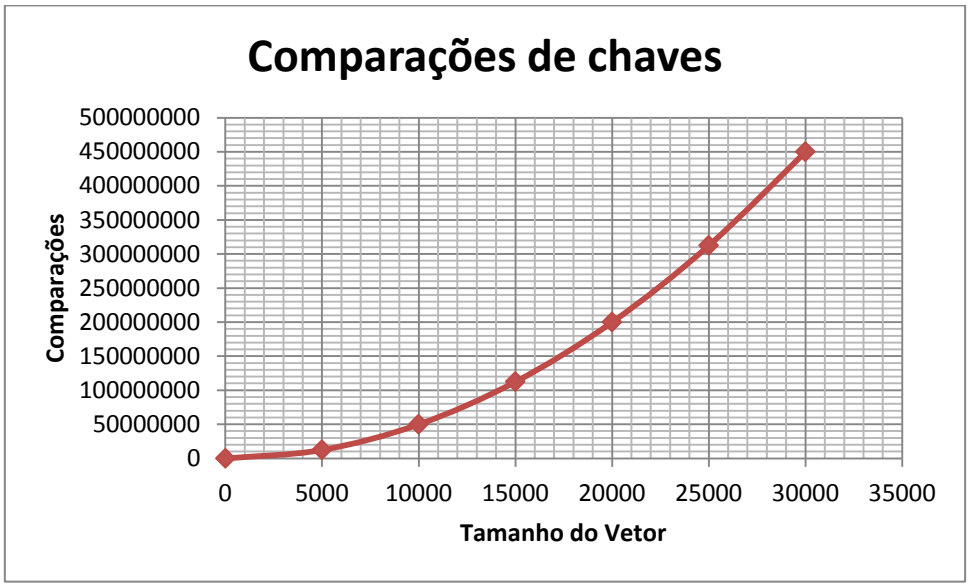

Tempo de Execução
tempo (milissegundos)
60
50
40
30
20
10
0
0
20000
40000
60000
80000
100000
Tamanho do Vetor

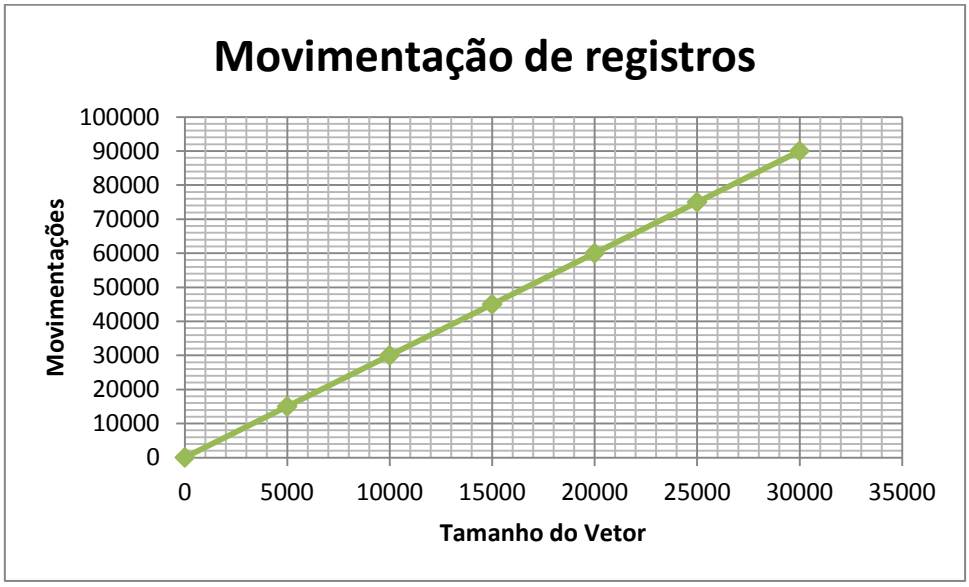
Chaves Ordenadas (crescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	90	27	0
5000	12522495	14997	82
10000	50044995	29997	325
15000	112567495	44997	752
20000	200089995	59997	1362
25000	312612495	74997	2153
30000	450134995	89997	3119





Chaves Ordenadas (decrescente)			
Tamanho do Vetor	Comparações de chaves	Movimentação de registros	Tempo (ms)
10	90	27	0
5000	12522495	14997	82
10000	50044995	29997	325
15000	112567495	44997	725
20000	200089995	59997	1378
25000	312612495	74997	2183
30000	450134995	89997	3190





Observações do QuickSort

No QuickSort algo interessante pode ser observado, quando temos as chaves geradas aleatoriamente ele se comporta como uma função linear, quase logarítmica, mas se pegarmos os piores casos, ordenados, ele passa a ser uma função exponencial quanto ao número de comparações. Já o número de movimentações é sempre uma função linear. O tempo de execução se comporta da mesma maneira que o número de comparações, isso nos leva a crer que o tempo de execução está ligado diretamente ao número de comparações.

Conclusão

Podemos concluir que para vetores muito grandes o método BubbleSort é o menos eficiente, dentre os aqui avaliados, ele só apresenta uma característica interessante comparado aos outros métodos quando tentamos ordenar um vetor já ordenado, ele executa extremamente rápido e não perde tempo realizando uma nova ordenação, o que não ocorre com os outros. O MergeSort é mais rápido que o BubbleSort mas ainda não é rápido o suficiente, apesar de seu crescimento ser praticamente linear, se estivermos pensando em uma situação onde as chaves estão de uma maneira aleatória no vetor, o QuickSort ainda terá maiores chances de ser o mais eficiente dentre os três caso o vetor seja extremamente grande. Depois de olhar para o código fonte dos três, o que se apresenta mais simples e fácil implementação é o BubbleSort, já o MergeSort e QuickSort, por utilizarem recursão e seguirem a lógica da Divisão e Conquista, se apresentam mais complexos e, não sendo tão simples sua implementação, não é algo tão direto.

Referências

http://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Bubble_sort
http://www.ic.unicamp.br/~islene/mc102/aula22/mergesort.c
http://www.comp.dit.ie/rlawlor/Alg_DS/sorting/quickSort.c
http://moodle2.dc.ufscar.br/pluginfile.php/1281/mod_resource/content/1/pdfs/apostila.pdf