Relatório

Heloisa Benedet Mendes
Bacharelado em Ciência da Computação
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
heloisamendes@ufpr.br

***Resumo*—Realização e detalhamento de testes sobre algoritmos de busca e ordenação de vetores, obtendo e relacionando dados de tempo de execução e número de comparações entre elementos de vetor.**

***Index Terms*—algoritmo, tempo, comparação, busca, ordenação, Sort**

I. Introdução

Este relatório busca demonstrar e relacionar os resultados obtidos em uma gama de testes aplicados nos algoritmos de busca *Busca Sequencial* e *Busca Binária*, e nos algoritmos de ordenação *Insertion Sort, Selection Sort, Merge Sort, Quick Sort* e *Heap Sort*, utilizando-se de sua estruturação recursiva. Tais algoritmos apresentam um custo computacional, este trabalho estuda, em específico, os gastos de tempo e de comparações entre elementos de vetor.

O trabalho é escrito em linguagem C e conta com os arquivos de cabeçalho *ordenação.h* - contém os códigos dos programas principais - e *auxiliares.h* - armazena os demais programas utilizados pela *ordenação.h* -, criados com o fim de segmentar e organizar o programa.

O código principal, ademais, realiza os testes e expõe, de forma geral, o número de comparações feitas e o tempo utilizado pelo algoritmo.

II. Especificações

Os experimentos realizados consistem na execução de todos os 5 algoritmos de ordenação em 3 modalidades diferentes:

- Vetor aleatório: elementos gerandos randomicamente com valores variando entre 0 e 1000.
- Vetor ordenado de forma decrescente: tem como primeiro elemento o tamanho do seu vetor, e subtrai 1 deste valor a cada posição.
- Vetor ordenado de forma crescente: o primeiro elemento é 0, e, até o fim do vetor, acresce 1 ao elemento a cada posição.

Para fins de comparação, estes testes foram feitos com diferentes tamanhos de vetor: 10, 100, 1.000, 2.000, 10.000, 20.000 e 100.000. Os mesmos valores foram aplicados a ambos os algoritmoa de busca, no entanto, nesses casos é utilizado somente o vetor ordenado de forma crescente. Dessa forma, é possível observar os diferentes comportamentos dos programas com vetores pequenos, bem como com vetores significantes.

Consegue-se, assim, obter uma quantidade suficiente de dados para que se faça possível relacionar os resultados e chegar à conclusões sólidas.

III. Testes

A seguir, tem-se os resultados dos testes anteriormente detalhados, com o tempo medido em segundos. Cada tabela tem como parâmento a recorrência do respectivo algoritmo, levando em conta os melhores e piores casos - se existirem.

Cada um dos casos apresentados nas figuras a seguir são calculados a partir das recorrências listadas, as quais definem $C^-(n)$, e $C^+(n)$, as comparações restantes foram obtidas a partir da execução do programa:

- *Busca Sequencial*, Figura 1: $1 \leq C(n) \leq n$ [1]

Busca Sequencial			Vetor aleatório	
Tamanho	C-(n)	C+(n)	Comparações	Tempo
10	1	10	7	0,000002
100	1	100	97	0,000005
1000	1	1000	997	0,00006
2000	1	2000	1997	0,000135
10000	1	10000	9997	0,000642
20000	1	20000	19997	0,001214
100000	1	100000	99997	0,004235

Figura 1.

- *Busca Binária*, Figura 2: $C(n) \approx log_2 n$ [2]

Busca Binária		Vetor crescente	
Tamanho	C(n)	Comparações	Tempo
10	3,32	4	0,000001
100	6,64	7	0,000001
1000	9,96	10	0,000001
2000	10,96	11	0,000001
10000	13,28	13	0,000001
20000	14,28	14	0,000001
100000	16,6	17	0,000001

Figura 2.

- *Insertion Sort*, Figura 3: $n - 1 \leq C(n) \leq \frac{n^2+n-2}{2}$ [3]

Insertion Sort			Vetor Aleatório		Vetor decrescente		Vetor crescente	
Tamanho	C-(n)	C+(n)	Comparações	Tempo	Comparações	Tempo	Comparações	Tempo
10	9	54	26	0,000003	21	0,000002	28	0,000002
100	99	5049	540	0,000067	485	0,000082	579	0,000016
1000	999	500499	8583	0,003527	7995	0,002641	8986	0,000048
2000	1999	2000999	19181	0,00694	17973	0,00699	19963	0,0001
10000	9999	50004999	118998	0,084329	113643	0,162749	123630	0,000586
20000	19999	200009999	257973	0,327703	247261	0,652099	267247	0,001273
100000	99999	705082703	1522715	8,191676	1468961	16,353844	1568945	0,009215

Figura 3.

- *Selection Sort*, Figura 4: $C(n) \approx \frac{n^2-n}{2}$ [4]

Selection Sort		Vetor Aleatório		Vetor decrescente		Vetor crescente	
Tamanho	C(n)	Comparações	Tempo	Comparações	Tempo	Comparações	Tempo
10	45	45	0,000003	45	0,000002	45	0,000002
100	4950	4950	0,00006	4950	0,000051	4950	0,00005
1000	499500	499500	0,001102	499500	0,00099	499500	0,001043
2000	1999000	1999000	0,004304	1999000	0,00386	1999000	0,0041
10000	49995000	49995000	0,101431	49995000	0,095285	49995000	0,101574
20000	199990000	199990000	0,403389	199990000	0,382839	199990000	0,407615
100000	704982704	704982704	10,026455	704982704	9,504642	704982704	10,11985

Figura 4.

- *Merge Sort*, Figura 5: $C(n) = n \cdot log_2 n$ [5]

Merge Sort		Vetor Aleatório		Vetor decrescente		Vetor crescente	
Tamanho	C(n)	Comparações	Tempo	Comparações	Tempo	Comparações	Tempo
10	33	24	0,000005	24	0,000003	24	0,000002
100	664	572	0,000037	572	0,000024	572	0,000024
1000	9965	8976	0,000119	8976	0,000066	8976	0,000071
2000	21931	19952	0,000254	19952	0,000142	19952	0,000155
10000	132877	123616	0,001286	123616	0,000759	123616	0,000809
20000	285754	267232	0,002644	267232	0,001587	267232	0,001569
100000	1660964	1568928	0,014313	1568928	0,009017	1568928	0,009451

Figura 5.

Quick Sort			Vetor Aleatório		Vetor decrescente		Vetor crescente	
Tamanho	C-(n)	C+(n)	Comparações	Tempo	Comparações	Tempo	Comparações	Tempo
10	33	54	31	0,000002	45	0,000002	45	0,000002
100	664	5049	780	0,000023	4950	0,000073	4950	0,000094
1000	9965	500499	10316	0,000084	497506	0,001598	499500	0,002043
2000	21931	2000999	25303	0,000177	1999000	0,006211	1999000	0,008814
10000	132877	50004999	182612	0,001047	49995000	0,148952	49995000	0,208125
20000	285754	200009999	435485	0,002596	199950000	0,597171	199990000	0,839982
100000	1660964	705082703	6380642	0,030353	704982704	14,917822	704982704	20,847204

Figura 6.

- *Quick Sort*, Figura 6: $n \cdot log_2 n \leq C(n) \leq \frac{n^2+n-2}{2}$ [6]
- *Heap Sort*, Figura 7: $0 \leq C(n) \leq n + 2 \cdot n \cdot log_2 n$ [7]

Heap Sort			Vetor Aleatório		Vetor decrescente		Vetor crescente	
Tamanho	C-(n)	C+(n)	Comparações	Tempo	Comparações	Tempo	Comparações	Tempo
10	0	76	68	0,000003	52	0,000002	70	0,000002
100	0	1428	1252	0,000037	1132	0,000031	1380	0,000037
1000	0	20931	19178	0,00012	17632	0,000101	20416	0,000121
2000	0	45862	42296	0,000313	39416	0,000232	44600	0,000271
10000	0	275754	258190	0,001803	243688	0,001356	273912	0,001453
20000	0	591508	556606	0,003589	528668	0,003457	585756	0,003515
100000	0	3421928	3248138	0,022805	3094868	0,018735	3401708	0,019909

Figura 7.

IV. Análise

Nota-se, com base nos dados obtidos, que o *Merge Sort* é o melhor algoritmo dentre os testados, uma vez que combina o menor custo computacional e os menores gastos de tempo nos 3 tipos de vetores.

A Figura 8 apresenta um gráfico gerado a partir dos resultados obtidos em relação ao vetor aleatório. Em sua análise, é possível perceber a disparidade entre as comparações realizadas pelo *Selection Sort* e pelos demais programas.

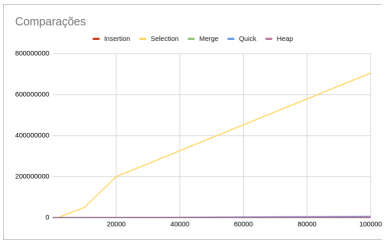

Figura 8.

Ao aproximar o gráfico dos menores custos, Figura 9, observa-se que, como demonstrado em suas recorrências, tanto o *Insertion Sort* quanto o *Merge Sort* seguem a mesma equação e, nesse caso, o *Insertion Sort* obedece sua recorrência de pior caso, $C^{-}(n)$.

No entanto, em um detalhamento dos piores casos para cada um dos algoritmos, obtém-se as Tabelas I e II e, a partir destas, observa-se que os gastos computacionais não são interdependentes, ou seja, mesmo que um algoritmo ordene um vetor em um tempo suficientemente bom, ele pode realizar um número indesejavelmente alto de comparações entre elementos do vetor.

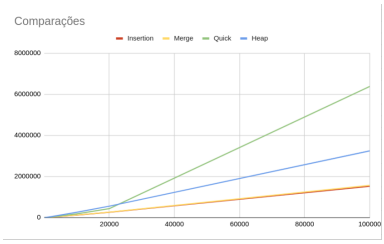

Figura 9.

Tabela I

Comparações

Algoritmo	Caso
Insertion	Cresente
Selection	-
Merge	-
Quick	Crescente, Decrescente
Heap	Crescente

Tabela II

Tempo

Algoritmo	Caso
Insertion	Decrescente
Selection	Crescente
Merge	Aleatório
Quick	Crescente
Heap	Aleatório

Exemplificando, o *Quick Sort* tem um desempenho análogo com os vetores ordenados, e, no entanto, leva 14,91 segundos para executar com um vetor decrescente de 100.000 posições, e 20,84 segundos com um vetor crescente de mesmo tamanho. O mesmo programa em condições semelhantes apresenta uma diferença de 1,39 segundos.

V. Conclusão

Em conclusão, este trabalho tem como objetivo a análise do comportamento de algoritmos de busca e ordenação de vetores em diferentes situações, que variam entre o tamanho do vetor fornecido e sua composição.

É observado, de forma resumida, que a eficiência de um algoritmo, portanto, não deve ser calculada levando-se em conta a análise de um único tipo de custo, mas sim do máximo de dados possíveis de serem obtidos.

Referências

[1] "Busca Sequencial - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=948806redirect=1

[2] "Busca Binária - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=949632redirect=1

[3] "Insertion Sort - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=956984redirect=1

[4] "Selection Sort - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=957854redirect=1

[5] "Merge Sort - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=958662redirect=1

[6] "Quick Sort - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=965765redirect=1

[7] "Heap Sort - cálculo da recorrência." UFPR. [Online]. Available: https://ufprvirtual.ufpr.br/mod/resource/view.php?id=969183redirect=1