Trabalho 1 - Ordenação

Thalita Maria do Nascimento

Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
thalitanascimento@ufpr.br

25 de março de 2022

Resumo

O presente trabalho visa comparar o custo dos algoritmos *Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Busca Sequencial* e *Busca Binária* pela comparação de elementos e pelo tempo de execução em segundos.

1 Introdução

No presente trabalho, foram desenvolvidos os algoritmos *Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Busca Sequencial* e *Busca Binária* recursivamente na linguagem C, no sistema operacional Linux. O objetivo era a realização de testes de custo em segundos e em comparações entre os elementos dos vetores, para que se fosse analisado os resultados obtidos em diferentes algoritmos.
Os testes foram realizados para cada um dos algoritmos com as seguintes quantias de elementos: 1000, 2000, 5000, 10000, 20000 e 50000. O elemento utilizado para o teste de busca nos vetores foi o número 42.

2 Comparações entre os Elementos

A forma escolhida de visualizar os dados foi transpô-los numa tabela. A tentativa de plotar gráficos não foi efetiva, pois os resultados dos algoritmos de *Insertion Sort* e *Selection Sort* estavam em ordem numérica muito maior que o *Merge Sort* e *Quick Sort*, assim, a visualização linear ofuscou totalmente os seus dados. Em razão disso, a forma encontrada para analisar foi reestruturar os valores obtidos em notação científica. Dessa maneira, é possível de visualizar as comparações entre elementos analisando a ordem numérica.

O algoritmo que melhor performou em todos os casos de testes entre os de ordenação foi o *Merge Sort*. Em segundo lugar, o *Quick Sort* efetou comparações em quantidades muito próximas do *Merge*. Em terceiro lugar está o *Insertion Sort* e, no extremo oposto está o Selection Sort, que realizou o maior número de comparações em relação aos outros três algoritmos em todos os casos.

Tabela 1: Comparações por Quantidade de Elementos

	Quantidade de Elementos					
Algoritmo	1000	2000	5000	10000	20000	50000
Insertion	2,5E5	9,99E5	6,2E6	2,4E7	9,9E7	6,3E8
Selection	4,99E5	1,999E6	1,2E7	4,999E7	1,9999E8	1,25E9
Merge	9,976E3	2,19E4	6,18E4	1,34E5	2,87E5	7,84E5
Quick	1,084E4	2,76E4	7,29E4	1,78E5	3,47E5	1,005E6
Busca S	1000	2000	5000	10000	20000	50000
Busca B	18	20	24	26	28	30

Em todos os casos, o *Selection Sort* foi aproximadamente 2 vezes mais custoso que o *Insertion Sort*. A seguir está a tabela da razão entre o custo do *Selection Sort* e o *Merge Sort* em cada teste.

Tabela 2: Razão Selection Sort/Merge Sort

	Quantidade de Elementos					
	1000	2000	5000	10000	20000	50000
Razão	50,02	91,28	194,17	373,0597	696,83	1594,39

É possível de perceber que a razão é de aproximadamente o dobro de cada teste anterior. O caso de 50000 elementos é o qual tem a diferença que mais chama a atenção, pois o *Merge Sort* é quase 1600 vezes mais eficiente que o *Selection Sort*.
No caso das *Buscas Sequencial* e *Binária*, os valores obtidos foram os de pior caso, pois em nenhum dos casos o elemento foi encontrado. Enquanto a *Busca Sequencial* efetua n comparações no pior caso, a *Busca Binária* aumenta apenas duas comparações ao se dobrar o número de elementos de um vetor.

Tabela 3: Razão Busca Sequencial/Busca Binária

	Quantidade de Elementos					
	1000	2000	5000	10000	20000	50000
Razão	55,55	100	208,33	384,61	714,28	1666,67

Ao analisar a tabela acima, também é possível de perceber que a razão tende a dobrar em relação a anterior.

3 Tempo em Segundos

Ao compararmos os algoritmos de ordenação, o *Quick Sort* é o qual executa em menos tempo. Em seguida, o *Merge Sort* e, após, o *Insertion Sort* e o *Selection Sort*. Apenas no primeiro caso o *Insertion Sort* demorou mais que o *Selection Sort* para executar.

Tabela 4: Tempo em segundos por Quantidade de Elementos

	Quantidade de Elementos					
Algoritmo	1000	2000	5000	10000	20000	50000
Insertion	8,33E-3	3,06E-2	2,196E-1	9,5E-1	4,496	34,868
Selection	7,78E-3	3,36E-2	2,64E-1	1,251	7,520	61,765
Merge	2,7E-4	6,3E-4	1,71E-3	3,66E-3	7,7E-3	2,098E-2
Quick	2E-4	4,42E-4	1,22E-3	2,991E-3	6,066E-3	1,68E-2
Busca S	0	5,6E-5	1,3E-4	3,24E-4	7,81E-4	2,26E-3
Busca B	0	1E-6	1E-6	1E-6	2E-6	2E-6

A seguir está a razão entre o tempo em segundos do *Selection Sort* pelo *Quick Sort*.

Tabela 5: Razão Selection Sort/Quick Sort

	Quantidade de Elementos					
	1000	2000	5000	10000	20000	50000
Razão	38,9	76,02	216,39	418,25	1239,7	3676,5

A razão de tempo não segue um padrão, mas mesmo com poucos elementos em um vetor, a eficiência do *Quick Sort* se mostra muito melhor.

No caso dos algoritmos de busca, o primeiro teste ocorreu em uma faixa de tempo menor que o tipo de variável *clock_t* poderia armazenar, então o valor registrado foi 0 segundos. É interessante de se notar que o tempo de execução da *Busca Binária* praticamente se mantém ao se dobrar o número de elementos em um vetor. Já no caso da *Busca Sequencial*, ao se dobrar a quantidade de elementos de um vetor, o tempo de execução mais do que dobra.

4 Conclusão

Analisando os resultados obtidos entre os algoritmos, foi possível de perceber que o *Merge Sort* é o algoritmo mais eficiente para se ordenar vetores quando o custo é analisado por comparações entre elementos. Quando o custo se trata de tempo em segundos, o *Quick Sort* executa mais rápido que todos os outros. No caso das buscas, a *Busca Binária* é de longe o algoritmo mais eficiente, porque com menos de centenas de comparações verifica se o elemento está presente ou não no vetor, mesmo que as quantidades de elementos se aproximem da ordem de milhão.