Relatório

Lucas Emanuel de Oliveira Santos
GRR20224379
Universidade Federal do Paraná - UFPR
Curitiba, Brasil

I. Introdução

O presente trabalho é um estudo dos algoritmos de busca sequencial e binária, assim como dos seguintes algortimos de ordenação: InsertionSort, SelectionSort, MergeSort, QuickSort e HeapSort. Os algoritmos foram implementados através da linguagem de programação c. Os testes foram feitos utilizando vetores gerados aleatoriamente. Os tamanhos usados variam dependendo do algoritmo por estarem intrinsecamente relacionados ao uso de memória e ao hardware da máquina usada para os testes. O tempo foi calculado utilizando a biblioteca *time.h.*

II. Busca Sequencial

A função de custo de comparações da Busca Sequencial é $1 \leqslant C(n) \leqslant n$, como pode ser observado na tabela I, o número de comparações nunca é maior que o tamanho do vetor. Porem quanto maior o vetor, mais os números divergem com os observados na tabela II(busca binária) tanto no tempo, quanto no número de comparações, com o melhor caso da busca sequencial sendo cada vez mais improvável.

Tabela I

Tamanho	Número de Comparações	Tempo
10	7	0.000001
100	36	0.000001
1000	175	0.000009
10000	4098	0.000074
100000	25722	0.000467

III. Busca Binária

A função de custo de comparações da Busca Binária é C(n) = $\log_2(n)$. A tabela II mostra a eficiência do algoritmo, que por ser uma função logaritmica cresce de forma lenta, tanto no número de comparações, como no tempo gasto para a execução do algoritmo.

Tabela II

Tamanho	Número de Comparações	Tempo
10	4	0.000000
100	7	0.000001
1000	10	0.000001
10000	13	0.000001
100000	17	0.000001
1000000	20	0.000002
10000000	23	0.000004
100000000	27	0.000009
1000000000	30	0.000040

IV. Insertion Sort

A função de custo de comparações do InsertionSort está relacionada a busca em vetores e consequentemente varia de acordo com a funçaõ de busca utilizada. Usando busca sequencial a função de custo fica $n-1 \leqslant C(n) \leqslant \frac{n^2+n-2}{2}$, o desempenho do InsertionSort utilizando busca sequencial é mostrado na tabela III. Usando busca binária a função de custo fica aproximadamente C(n) = $n \cdot \log_2(n)$, o desempenho do InsertionSort utilizando busca binária é mostrado na tabela IV.

Tabela III

Tamanho	Número de Comparações	Tempo
10	25	0.000005
100	2419	0.000105
1000	241228	0.006029
10000	24092054	0.242648
100000	Overflow	24.338908

Tabela IV

Tamanho	Número de Comparações	Tempo
10	22	0.000006
100	528	0.000066
1000	8577	0.003397
10000	119038	0.076292
100000	1522619	7.309198

V. Selection Sort

A função de custo de comparações do Selection Sort é C(n) = $\frac{n^2-n}{2}$ como pode ser observado na tabela V, o número de comparações é constante.

Tabela V

Tamanho	Número de Comparações	Tempo
10	45	0.000003
100	4950	0.000062
1000	499500	0.001157
10000	49995000	0.077813
100000	4999950000	0.022026

VI. Merge Sort

A função de custo de comparações do Merge Sort é C(n) = $n \cdot \log_2(n)$. Quanto comparado com os outros algoritmos estudados nesse trabalho vemos que o Merge Sort faz menos comparações e é mais rápido que a maioria, o problema está no elevado custo de memória do mesmo, que é aproximadamente o dobro do utilizado pelos demais.

Tabela VI

Tamanho	Número de Comparações	Tempo
10	22	0.000004
100	544	0.000038
1000	8718	0.000082
10000	120512	0.001049
100000	1536140	0.022026
1000000	10066432	1.537742

VII. Quick Sort

A função de custo de comparações do Quick Sort é $n \cdot \log_2(n) \leqslant C(n) \leqslant \frac{n^2+n-2}{2}$.
Como é mostrado na tabela VII, os casos ruins do Quick Sort são muito improváveis, e se comparado a os outros algortimos analisados nesse trabalho, é percepitível a vantagem de desempenho do QuickSort no que diz respeito ao tempo, e na maioria das vezes, ao número de comparações. Pórem com o aumento do tamanho, pior fica o desempenho do quicksort nos piores casos e portanto, para vetores extremamente grandes deve ser analisado se o risco do pior caso deve ser ou não contemplado.

Tabela VII

Tamanho	Número de Comparações	Tempo
10	21	0.000003
100	651	0.000026
1000	13829	0.000074
10000	151754	0.000903
100000	2041850	0.011539
1000000	25029716	0.149896
10000000	296477215	1.899437

VIII. Heap Sort

A função de custo de comparações do Heap Sort é C(n) = $n+2 \cdot n \cdot \log_2(n)$, comparando a tabela VIII com as tabelas VII e VI, vemos que o número de comparações do Heap Sort é aproximamente duas vezes maior que o do Quick Sort e do Merge Sort, e o algortimo é mais lento que ambos. A vantagem está no fato do Heap Sort usar menos memória que o Merge Sort e não possuir um pior caso como o Quick Sort.

Tabela VIII

Tamanho	Número de Comparações	Tempo
10	82	0.000005
100	1458	0.000044
1000	21132	0.000134
10000	278642	0.001652
100000	3450818	0.025006
1000000	41097430	0.364537
10000000	477669544	5.387264

IX. Conclusão

O uso dos diversos algoritmos de ordenação é situacional e depende exclusivamente da necessecidade do usuário, todos possuem vantagens e disvantagens que devem ser consideradas na impletação e uso do algoritmo.

X. Referências

Curso de Algoritmos e Estrutura de Dados 2. Almeida, Paulo, 2022. Disponível em: https://ufprvirtual.ufpr.br/course/view.php?id=26889, Acesso em 18/11/2022 à 29/02/2023.