Trabalho 1 - Ordenação - CI1056

Wesley Peres de Freitas
GRR20221235
Universidade Federal do Paraná
Curitiba, Brasil

1. INTRODUÇÃO

Este trabalho busca analisar de forma breve os algoritmos de ordenação Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Heap Sort, Busca Binária e Busca Sequencial.

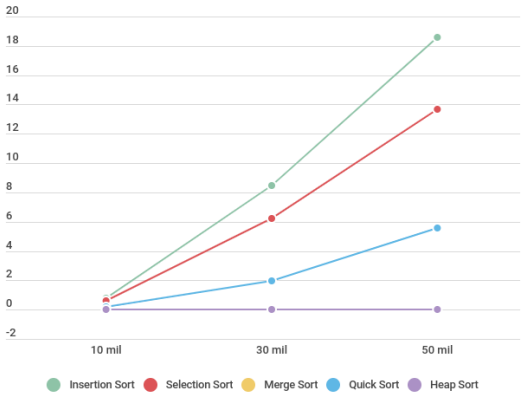
2. ALGORITMOS DE ORDENAÇÃO

Alguns algoritmos possuem melhor e pior caso. Para Insertion Sort e Quick Sort, o pior caso ocorre quando o vetor está ordenado inversamente (ex: 3, 2, 1). Por isso, tais vetores foram utilizados nos testes. No entanto, para não restringir a análise somente ao pior caso, testes foram realizados com vetores aleatórios (ex: 2, 1, 3).

2.1 VETOR ORDENADO INVERSAMENTE

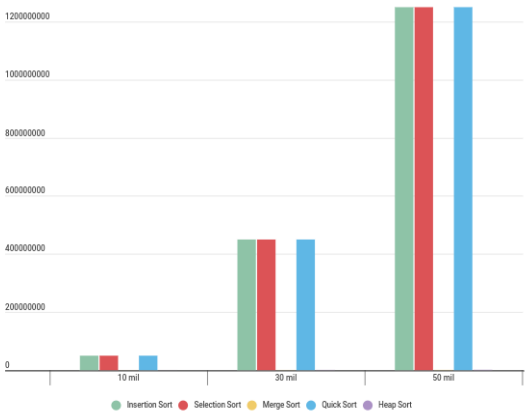
Foram utilizados três vetores nos testes, com dez mil, trinta mil e cinquenta mil elementos (FIGURA 1). Para 10 mil elementos, o tempo é quase o mesmo para todos os algoritmos. No entanto, Insertion Sort, Selection Sort e Quick Sort crescem de forma acelerada para tamanhos maiores.

FIGURA 1 - Running Time em Vetor Ordenado



Quanto ao número de comparações (FIGURA 2), verifica-se que os algoritmos com maior tempo de execução são também aqueles com mais comparações. Insertion Sort, Selection Sort e Quick Sort possuem quantidades parecidas de comparações, como sugerem suas fórmulas. Selection Sort tem custo fixo de $\frac{n(n-1)}{2}$, que é igual ao custo do pior caso do Insertion Sort. Quick Sort, em seu pior caso, tem um custo semelhante, de $\frac{n^2+n-2}{2}$. Por outro lado, Merge Sort e Heap Sort possuem ambos custo de $n \log_2 n$.

FIGURA 2 - Número de Comparações de Vetor Ordenado

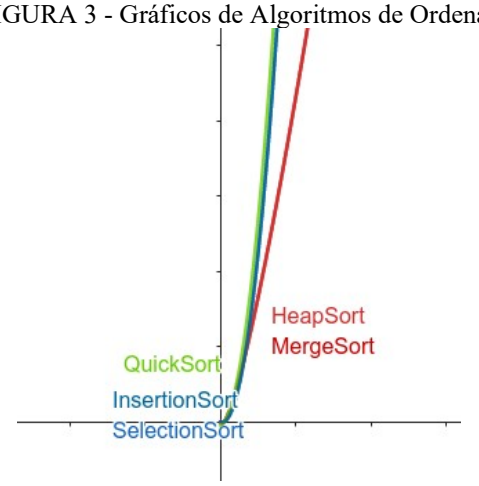


No entanto, o tempo de execução não depende somente do número de comparações. Por exemplo, o Insertion Sort, apesar de ter o mesmo número de comparações que o Selection Sort, tem um tempo de execução maior por realizar mais trocas.

De dez mil para cinquenta mil elementos, o tempo de execução aumenta cerca de 24 vezes para Insertion Sort, 13 vezes para Selection Sort e 25 vezes para Quick Sort. Quanto ao número de comparações, Insertion Sort aumenta cerca de 25 vezes, Selection Sort 25 vezes e Quick Sort 24 vezes. Observa-se que Merge Sort e Heap Sort pouco variam tanto em número de comparações como no tempo de execução.

Abaixo, os gráficos das funções que calculam o número de comparações no pior caso (ou no caso único, quando não há pior caso) de cada algoritmo.

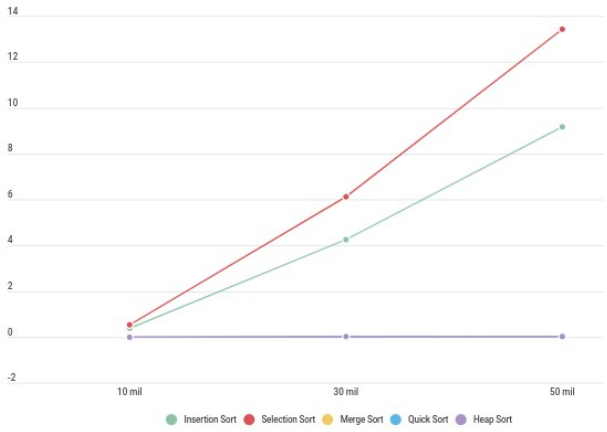
FIGURA 3 - Gráficos de Algoritmos de Ordenação



2.1 VETOR ALEATÓRIO

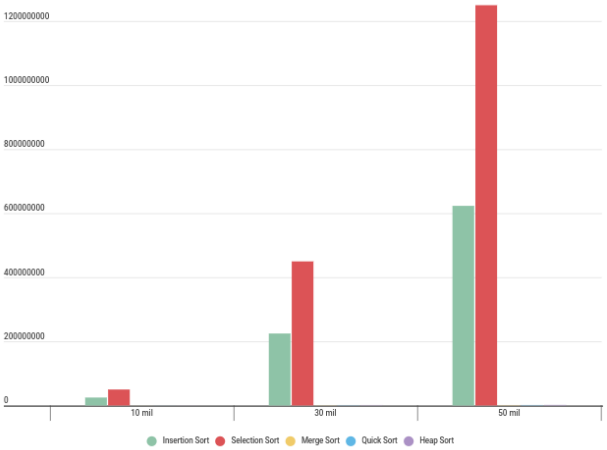
Os testes realizados com vetores aleatórios revelam uma dinâmica diferente (FIGURA 4). Insertion Sort realiza menos trocas e comparações em um vetor aleatório (FIGURA 5), contribuido para a redução do tempo. Selection Sort permanece com o mesmo tempo, sendo o mais lento. Além disso, Quick Sort deixa de gerar vetores de tamanho n - 1 e, consequentemente, aproxima-se do tempo de Merge Sort e Heap Sort, que permanecem baixos.

FIGURA 4 - Running Time em Vetor Aleatório



Para Quick Sort, o número de comparações também decai. Selection Sort permanece inalterado. Apesar de um número de comparações pouco maior que o de Merge Sort, Quick Sort é ligeiramente mais veloz para vetores aleatórios.

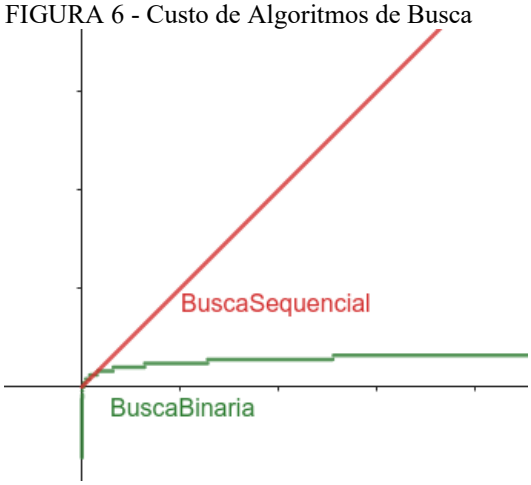
FIGURA 5 - Número de Comparações em Vetor Aleatório



3. ALGORITMOS DE BUSCA

O algoritmo da busca binária segue o melhor caso de 1 comparação e o pior caso de $2\lfloor \log_2 n \rfloor + 2$ comparações. Busca Sequencial tem o pior caso com custo de n comparações e melhor caso de 1 comparação. Abaixo, a representação dos gráficos das funções de custo.

FIGURA 6 - Custo de Algoritmos de Busca



4. CONCLUSÃO

O tempo de execução de algoritmos depende de outras variáveis além do número de comparações. Além disso, algoritmos com melhor caso tendem a ser mais eficientes para vetores aleatórios. Esse é o caso do Quick Sort, que apresenta maior eficiência em relação aos outros algoritmos analisados.