

Relatório de ALG2

Vinícius Fontoura e Carla Capurro
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
vfa20@inf.ufpr.br e ccc20@inf.ufpr.br

Index Terms—Busca Sequencial, Busca Binária, Insertion Sort, Selection Sort, Merge Sort, Quick Sort

I. INTRODUÇÃO

Nesse relatório, será abordado a implementação dos algoritmos de busca e ordenação vistas em aula. Assim como, seus custos em relação às comparações entre elementos de um vetor e o tempo de execução.

II. SOBRE AS ANÁLISES FEITAS

Nesse relatório, os algoritmos serão analisados com relação a: quantidade de comparações entre elementos de um dado vetor e o tempo de execução, dado um vetor de tamanho N .

Cada algoritmo terá sua complexidade exposta, assim como sua quantidade de comparações feitas.

III. SOBRE OS TESTES

O arquivo de testes - main.c - vai testar diferentes tamanhos e casos para cada algoritmo. De forma a mostrar que estes funcionam mesmo em casos extremos e/ou específicos. Mostrando também, para vetores de tamanho menor que 10, a visualização do vetor original e resultante de cada ordenação.

Os valores de cada índice dos vetores são gerados aleatoriamente pela função `srand()` [1], com geração fixa por questão de testes, mas facilmente alteráveis dentro do código.

Cada algoritmo faz um número de comparações, que é mostrado ao fim da execução de cada um deles. Assim como, o valor do tempo gasto pela cpu para processá-los.

IV. COMPLEXIDADE

A. Busca Sequencial

O algoritmo da busca sequencial é faz, em média, N comparações entre elementos de um vetor de tamanho N

B. Busca Binária

O algoritmo de Busca Binária é o mais eficiente entre os 2 algoritmos de busca abordados neste trabalho. Fazendo, em média $\log_2 N$ comparações entre elementos de um dado vetor de tamanho N .

C. Insertion Sort

O algoritmo de ordenação Insertion Sort faz, em média, $N^2/2$ comparações entre elementos de um dado vetor de tamanho N .

D. Selection Sort

O algoritmo de ordenação Selection Sort faz, em média, n^2 comparações entre elementos de um dado vetor de tamanho N

E. Merge Sort

O algoritmo de ordenação Merge Sort faz, em média, $N * \log_2 N$ comparações entre elementos de um dado vetor de tamanho N

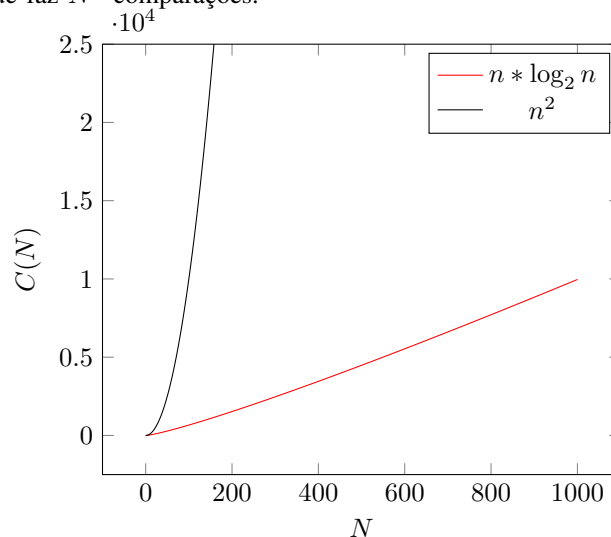
F. Quick Sort

O algoritmo de ordenação Quick Sort faz, em média, $N * \log N$ comparações entre elementos de um dado vetor de tamanho N . Seu pior caso acontece quando o vetor está quase ordenado, devido à comum escolha de selecionar o último elemento do vetor como "pivô".

G. Quantidade de comparações esperadas

Aqui vai uma visualização da diferença da quantidade de comparações entre elementos de um vetor para cada algoritmo.

É cristalino como os algoritmos que fazem: $n * \log_2 n$ comparações tem um custo muito menor que o pior deles, que faz N^2 comparações.

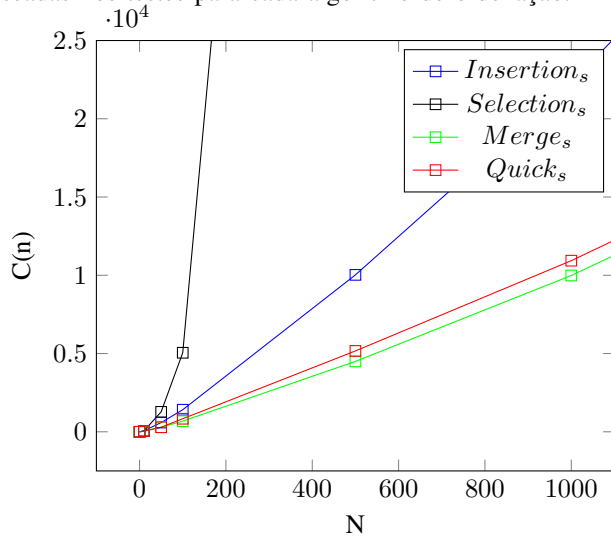


V. QUANTIDADE DE COMPARAÇÕES REAIS

Cada algoritmo gera um número de comparações entre elementos de um dado vetor. Para os testes serão usados 7 vetores, usando o seguinte padrão de tamanho = 10, 50, 100,

500, 1000, 5000, 10000 com elementos aleatórios dentro de cada vetor.

Os gráficos a seguir mostram a quantidade de comparações baseadas nos testes para cada algoritmo de ordenação:



Mostrando que: os testes refletem a quantidade de comparações esperadas entre os algoritmos, exceto o Insertion Sort (porém, trata-se de uma diferença nos intervalos usados para teste e intervalos usados no plot esperado).

VI. CONCLUSÃO

Fica claro que, em relação às comparações, para os algoritmos de busca: a Busca Binária é muito menos custosa. Já para os algoritmos de ordenação: o Merge Sort é menos custoso, devido ao seu pior e melhor caso serem $N * \log_2 N$.

Assim também, em relação ao tempo, para os algoritmos de busca: a Busca Binária é muito mais rápida - na ordem de 10^3 . Já para os algoritmos de ordenação: o Quick Sort é mais rápido.

REFERÊNCIAS

- [1] “função srand(),” Depto. de informática da UFSC, 01 2022. [Online]. Available: <http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5201-02202A/GerandoNumerosAleatoriosemC.pdf>