Universidade Federal
de São João del-Rei

Felipe Francisco Rios de Melo
Thales Mrad Leijoto

ANÁLISE DO ALGORITMO QUICKSORT

São João Del Rei, 2018

1. Introdução

O Algoritmo Quicksort, criado por *C. A. R. Hoare* em 1960, é o método de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

Provavelmente é o mais utilizado/customizado em projetos de linguagens de programação.

É um algoritmo de comparação que emprega a estratégia de *"divisão e conquista"*. A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores. Os problemas menores são ordenados independentemente e os resultados são combinados para produzir a solução final.

Este trabalho consiste em analisar o desempenho do algoritmo Quicksort (recursivo) em diferentes cenários, considerando três métricas de desempenho: número de comparações de chaves, o número de cópias de registros realizadas, e o tempo total gasto para ordenação (tempo de processamento e não o tempo de relógio).

2. Implementação

Para a implementação do algoritmo, foram dados três cenários para se realizar a ordenação:

• Os elementos a serem ordenados são inteiros armazenados em um vetor de tamanho N.

• Os elementos a serem ordenados são inteiros armazenados em uma lista duplamente encadeada com N elementos.

• Os elementos a serem ordenados são registros armazenados em um vetor de tamanho N. Cada registro contém:
  – Um inteiro, a chave para ordenação.
  – Dez cadeias de caracteres (strings), cada uma como 200 caracteres.
  – 1 booleano
  – 4 números reais.

<u>Funções e Procedimentos</u>

As funções criadas para serem utilizadas no programa principal foram:

int* leituraParametros(FILE *input, int *qtdN); Recebe como parâmetro o ponteiro para um arquivo de entrada contendo a quantidade e os valores que N assume, e o ponteiro para um

inteiro. Essa função percorre o arquivo, referenciando a quantidade de N ao ponteiro para o inteiro qtdN, e armazenando os valores de N em um vetor de inteiros. Por fim ela retorna o vetor N.

void randomVetor(int *v, int N);
void randomStruct(elemento *v, int N);
void randomLista(Lista *li, int *v, int N);
Essas funções recebem o ponteiro para a estrutura em questão (vetor de inteiros, struct de inteiros ou lista) e o inteiro N e gera números inteiros pseudo-aleatórios, no caso do randomStruct é gerado também cadeias de caracteres, um boleano e números reais.

void contaTempo(double *utime, double *stime); Recebe dois ponteiros para variáveis do tipo double, que iram marcar o tempo gasto em modo de usuário e em modo de sistema, essa função utiliza o comando getrusage(), que é parte da biblioteca padrão de C da maioria dos sistemas Unix. Ele retorna os recursos correntemente utilizados pelo processo, em particular os tempos de processamento (tempo de CPU) em modo de usuário e em modo sistema, fornecendo valores com granularidades de segundos e microsegundos.

void quickStruct(elemento *vetor, int inicio, int fim, int *compara, int *troca);
void quickInt(int *vetor, int inicio, int fim, int *compara, int *troca);
void quickLista(Elem* inicio, Elem* fim, int *compara, int *troca);
Essas funções são responsáveis pela ordenação dos elementos, nelas está contida a particiona(). Juntas, elas seguem a rotina abaixo:

- Escolhe o primeiro elemento da estrutura chamado pivô.
- Reorganiza a lista de forma que os elementos menores que o pivô fiquem de um lado, e os maiores fiquem de outro. Esta operação é chamada de “particionamento”.
- Recursivamente ordena a sub-lista abaixo e acima do pivô.

void computaEstatisticas(double *tmpmed, double *trcmed, double *cmpmed, int tipo, double tempo, int troca, int compara);
Essa função realiza o somatório do tempo total gasto, das trocas de registros e das comparações de chaves, para posteriormente calcular a média de cada métrica, para cada uma das três estruturas, e também para cada um dos valores de N.

void imprimeMediaArq(FILE *output, double *cmpmed, double *trcmed, double *tmpmed, int repeat);
Recebe como parametro o ponteiro para o arquivo de saída onde serão imprimidos os dados, e o ponteiro para o somatório do número de trocas, comparações e tempo gasto, além da quantidade de repetição (para ser usado como divisor da média).
A função imprime no arquivo a média de cada uma estrutura para as três métricas. Ela uma vez para cada valor de N.

Programa Principal

Para tal, primeiramente os parâmetros N's são lidos de um arquivo de texto. Dado a quantidade de N's e todos os valores que N irá assumir, em um laço de repetição que vai de 0 até a (quantidade de N)-1 aloca-se dinamicamente os três tipos de estrutura de dados, variando, em cada repetição, o seu tamanho em função de N.

Também em cada repetição deste laço, temos um outro laço que se repete 5 vezes, nele é chamado as funções que geram uma sequência números pseudo-aleatórios, "*pseudo*" pois a aleatoriedade é previamente definida pela função *srand()* que recebe como parâmetro um número inteiro que chamamos de semente (seed). A semente é passada pelo usuário por linha de comando no momento de execução do programa, e incrementada a cada repetição do laço, para que seja gerado valores diferentes para serem ordenados.

Geradas as sequências de dados randômicos para cada estrutura, é realizado o *quicksort* para o vetor de inteiros, a lista duplamente encadeada e o vetor de struct, para cada um desses dados, é coletado e armazenado as estatísticas de ordenação. Ao final do laço que se repete 5 vezes, teremos 5 *quicksort* realizados para cada uma das estruturas de dados, assim temos uma gama maior de informações para tirar conclusões.

Enfim, é chamada a função que imprime a média das 5 ordenações de cada tipo de estrutura, para cada um dos valores do parâmetro N.

3. Resultados e discussões

O algoritmo foi testado para diversas entradas diferentes e seu funcionamento foi satisfatório. Os testes foram realizados em um Intel Core i5 7ª geração i5-7200U, com 8gb de memória RAM. Abaixo está um dos testes realizados:

execução do programa: ./quicksort 10 entrada.txt saida.txt

Arquivo entrada.txt (parâmetro N):
7
1000
5000
10000
50000
100000
500000
1000000

Arquivo saida.txt (média das métricas para as três estruturas):

>>> MÉDIA DAS MÉTRICAS <<<

Para N = 1000

>>> VETOR DE INTEIROS
Comparações de chaves: 11110.40
Trocas de registros: 1735.40
Tempo gasto médio na ordenação: 0.000122s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 11193.40
Trocas de registros: 1751.00
Tempo gasto médio na ordenação: 0.000150s

>>> VETOR DE STRUCT
Comparações de chaves: 11316.60
Trocas de registros: 1733.60
Tempo gasto médio na ordenação: 0.000733s

Para N = 5000

>>> VETOR DE INTEIROS
Comparações de chaves: 74832.80
Trocas de registros: 11268.40
Tempo gasto médio na ordenação: 0.000722s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 74622.80
Trocas de registros: 11273.40
Tempo gasto médio na ordenação: 0.000886s

>>> VETOR DE STRUCT
Comparações de chaves: 73342.40
Trocas de registros: 11226.00
Tempo gasto médio na ordenação: 0.005423s

Para N = 10000

>>> VETOR DE INTEIROS
Comparações de chaves: 161244.80
Trocas de registros: 24830.80
Tempo gasto médio na ordenação: 0.001525s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 157024.40
Trocas de registros: 24942.80
Tempo gasto médio na ordenação: 0.001886s

>>> VETOR DE STRUCT
Comparações de chaves: 156774.20
Trocas de registros: 24995.80
Tempo gasto médio na ordenação: 0.012256s

Para N = 50000

>>> VETOR DE INTEIROS
Comparações de chaves: 979678.20
Trocas de registros: 150352.40
Tempo gasto médio na ordenação: 0.008828s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 955011.20
Trocas de registros: 151103.80
Tempo gasto médio na ordenação: 0.010910s

>>> VETOR DE STRUCT
Comparações de chaves: 1000581.60
Trocas de registros: 149806.80
Tempo gasto médio na ordenação: 0.078470s

Para N = 100000

>>> VETOR DE INTEIROS
Comparações de chaves: 2096944.20
Trocas de registros: 324355.80
Tempo gasto médio na ordenação: 0.018636s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 2018073.60
Trocas de registros: 327180.20
Tempo gasto médio na ordenação: 0.023164s

>>> VETOR DE STRUCT
Comparações de chaves: 2070947.00
Trocas de registros: 325238.20
Tempo gasto médio na ordenação: 0.171863s

Para N = 500000

>>> VETOR DE INTEIROS
Comparações de chaves: 11791584.60
Trocas de registros: 1900516.20
Tempo gasto médio na ordenação: 0.104764s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 11828565.40
Trocas de registros: 1898475.20
Tempo gasto médio na ordenação: 0.132370s

>>> VETOR DE STRUCT
Comparações de chaves: 12062699.80
Trocas de registros: 1888266.80
Tempo gasto médio na ordenação: 1.018445s

Para N = 1000000

>>> VETOR DE INTEIROS
Comparações de chaves: 24964785.20
Trocas de registros: 4039733.40
Tempo gasto médio na ordenação: 0.220663s

>>> LISTA DUPLAMENTE ENCADEADA DE INTEIROS
Comparações de chaves: 25698498.20
Trocas de registros: 4013377.40
Tempo gasto médio na ordenação: 0.281466s

>>> VETOR DE STRUCT
Comparações de chaves: 25068052.60
Trocas de registros: 4021382.60
Tempo gasto médio na ordenação: 2.187605s

TABELA 1. Comparações de chaves.

N	VETOR DE INTEIROS	LISTA DUPLAMENTE ENCADEADA DE INTEIROS	VETOR DE STRUCT
1000	11110.4	11193.4	11316.0
5000	74832.8	74622.8	73342.4
10000	161244.8	157024.4	156774.2
50000	979678.2	955011.2	1000581.6
100000	2096944.2	2018073.6	2070947.0
500000	11791584.6	11828565.4	12062699.8
1000000	24964785.2	25698498.2	25068052.6

GRÁFICO 1. Comparações de chaves.

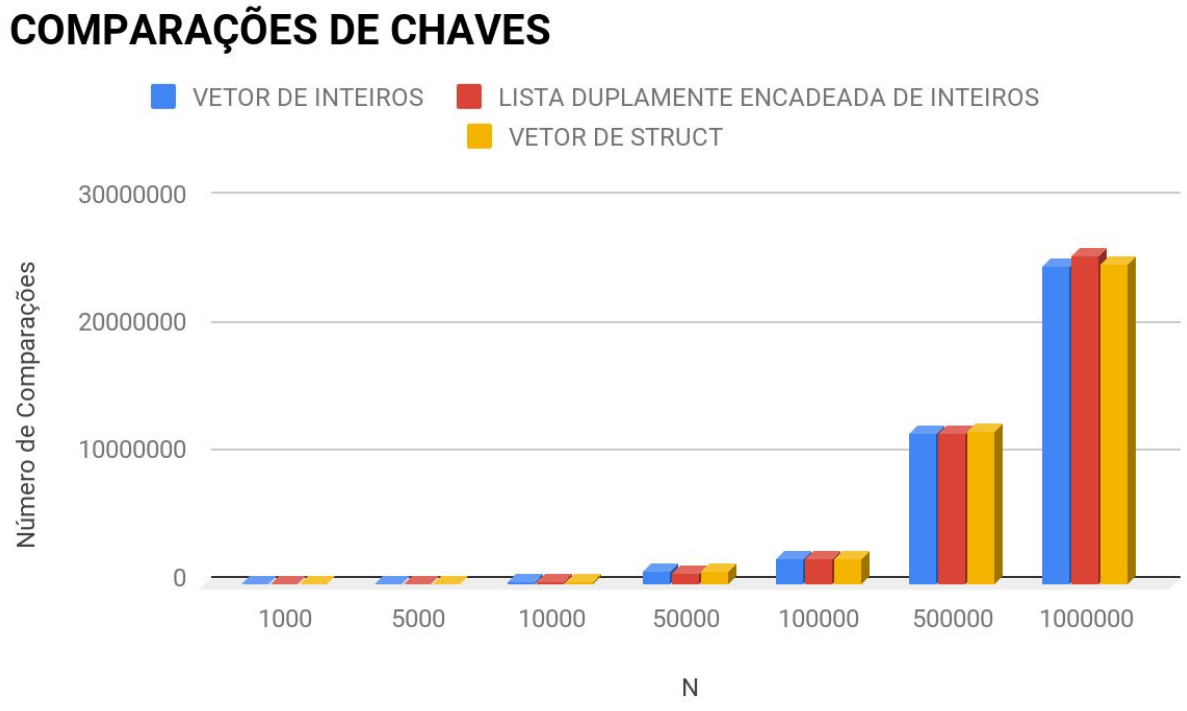


TABELA 2. Trocas de registros.

N	VETOR DE INTEIROS	LISTA DUPLAMENTE ENCADEADA DE INTEIROS	VETOR DE STRUCT
1000	1735.4	1751.0	1733.6
5000	11268.4	11273.4	11226.0
10000	24830.8	24942.8	24995.8
50000	150352.4	151103.8	149806.8
100000	324335.8	327180.2	325238.2
500000	1900516.2	1898475.2	1888266.8
1000000	4039733.4	4013377.4	4021382.6

GRÁFICO 2. Trocas de registros.

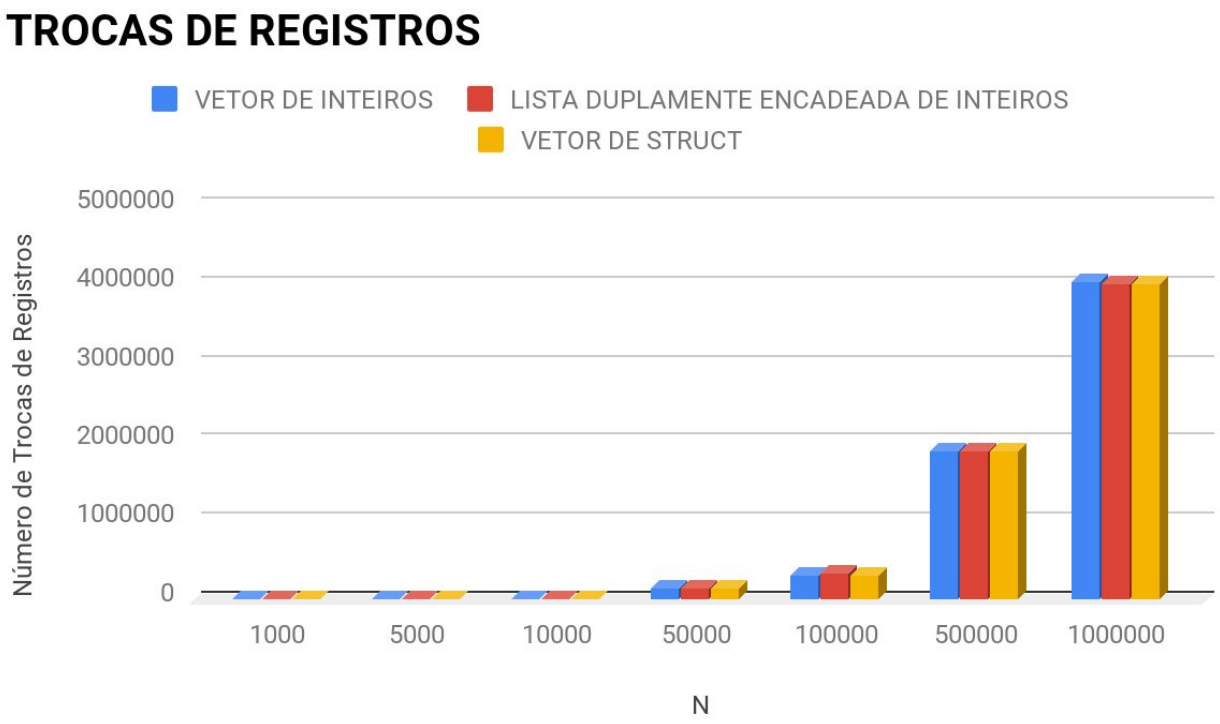
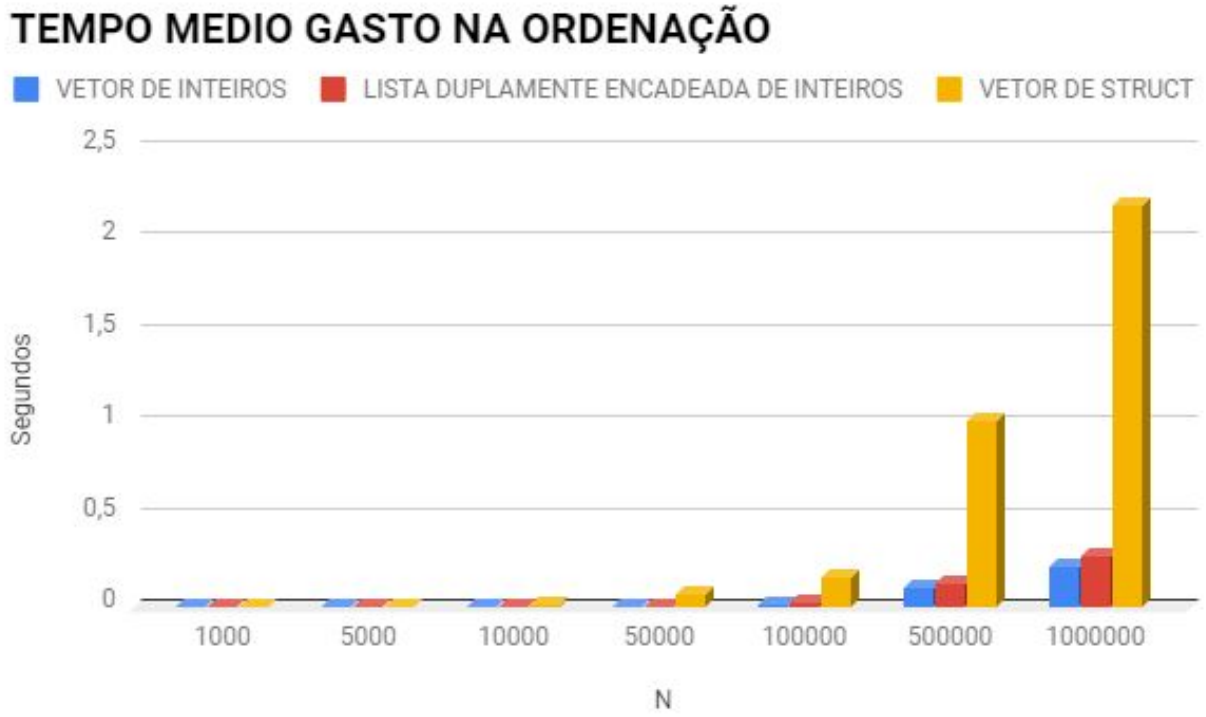


TABELA 3. Tempo médio gasto na ordenação em segundos.

N	VETOR DE INTEIROS	LISTA DUPLAMENTE ENCADEADA DE INTEIROS	VETOR DE STRUCT
1000	0,000122	0,000150	0,000733
5000	0,000722	0,000886	0,005423
10000	0,001525	0,001886	0,012256
50000	0,008828	0,010910	0,078470
100000	0,018636	0,023164	0,171863
500000	0,104764	0,132370	1,018445
1000000	0,220663	0,281466	2,187605

GRÁFICO 3. Tempo médio gasto na ordenação.



Ao analisar as tabelas e gráficos do teste realizado, podemos tirar algumas conclusões, tais como:

- Nos 3 casos, a quantidade de comparações e de trocas de registro são bem próximas, tendo uma pequena variação de um caso para outro. Isso acontece, pois independente da estrutura, as comparações de chaves e trocas de registro se dão de formas iguais em questão de quantidade (possui algoritmos similares).

- Quanto ao tempo gasto na ordenação, o caso de ordenação de vetor de struct gastou muito mais tempo, tendo um crescimento exponencial em relação aos demais casos. Chegando a gastar quase 10 vezes mais que os outros com 1000000 de elementos no vetor. O vetor de struct é mais lento em relação ao tempo de ordenação devido às trocas de posições, que ao contrário do vetor de inteiros e a lista de inteiros que tem apenas a troca de chaves, no vetor de struct ocorre a troca de todos os tipos de dados incluso na estrutura (*10 strings, 4 floats, 1 bool e 1 int*), ou seja, há um maior custo computacional nessas trocas.

4. Conclusão

Perante ao conteúdo exposto neste trabalho, podemos concluir que mesmo o quicksort sendo um algoritmo de ordenação não estável, ele é um dos mais rápidos e eficientes na maioria dos casos. Utilizando um princípio simples que é o “dividir e conquistar” de forma primorosa, é um dos algoritmos de ordenação mais utilizadas no mundo.

Por usar a recursividade e possuir alguns detalhes primordiais, não é simples de implementar no começo, principalmente na utilização de lista duplamente encadeada. Porém, ao entender a ideia elementar por trás do quicksort e tendo um pouco de paciência, a implementação se torna muito mais tranquila.

5. Referências Bibliográficas

Estrutura de dados descomplicada em linguagem C. André Backes. Editora Elsevier.

Conheça os principais algoritmos de ordenação, Treina Web. Disponível em: <https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/>

O ALGORITMO DE ORDENAÇÃO QUICKSORT, Gustavo Pantuza. Disponível em: <

https://blog.pantuza.com/artigos/o-algoritmo-de-ordenacao-quicksort>