## **Algoritmos e Estruturas de Dados II**

## Documentação - Switch Gravitacional

## Áulus Pinho

## 22 de Junho de 2017

# Intrudução

O código proposto simula um Switch Gravitacional representado pela ideia de uma caixa contendo n colunas de cubos arranjados em uma linha. Inicialmente a quantidade de cubos em cada coluna está distribuída de maneira desordenada e a gravidade puxa os cubos para baixo. Quando a gravidade é alterada, os cubos são puxados para o lado direito da caixa, de modo que, após o switch gravitacional, a quantidade de cubos em cada coluna fica distribuída de maneira ordenada e crescente.

Para representar a disposição (numérica) inicial dos cubos nas colunas da caixa, o programa aqui apresentado recebe um arquivo ‘input’ com os respectivos valores (ordenados ou não). Os valores então sao ordenados e gravados em um arquivo ‘output’, representando assim a disposição (numérica) final dos cubos na caixa.

## Como compilar?

Para compilar o arquivo no Linux basta abrir no terminal a pasta onde ele está e digitar:

. compile.sh

Para o sistema operacional Windows siga os passos abaixo:

gcc -c tp.c

gcc tp2.c tp.o -o tp2

./tp2 input.txt output.txt

# Desenvolvimento

Os valores obtidos do arquivo ‘input’ sao armazenados em um vetor de inteiros alocado dinamicamente e, o mesmo é submetido ao metodo de ordenação QuickSort. Tal método foi escolhido devido à sua complexidade O(n㏒n) nos melhores casos e nos casos médios, bem como sua melhor eficiência e rapidez para quaisquer tamanhos de entrada[[1]](#footnote-0). No caso do programa aqui proposto, dado a possibilidade de tamanhos de entrada diversos e ordenados aleatoriamente, o QuickSort é uma das melhores opções para ordenação interna.

O algoritmo QuickSort implementado foi estruturado em duas partes:

1. Partição: É feita a escolha de um pivô pelo cálculo da mediana de três. Logo após, o vetor é rearranjado e particionado em dois:
   * 1. Parte Esquerda: com valores menores do que o pivô,
     2. Parte Direita: com valores maiores do que o pivô.

void particao(int Esq,int Dir,int \*i, int \*j, int vetor[]){

int x, y;

\*i = Esq;

\*j = Dir;

x = vetor[(((\*i + \*j)/2) + \*i + \*j)/3]; /\* obtenho o pivo x \*/

do

{

while (x > vetor[\*i]) (\*i)++;

while (x < vetor[\*j]) (\*j)--;

if ( (\*i) <= (\*j) ){

y = vetor[\*i];

vetor[\*i] = vetor[\*j];

vetor[\*j] = y;

(\*i)++;

(\*j)--;

}

}

while ( (\*i) <= (\*j) );

}

1. Ordenação: Ordena as partes Esquerda e Direita do vetor através de novo particionamento:

void ordena(int Esq, int Dir, int vetor[]){

int i,j;

particao(Esq, Dir, &i, &j, vetor);

if(Esq < j)

ordena(Esq,j,vetor);

if (i < Dir)

ordena(i, Dir, vetor);

}

# Análise

O programa executa basicamente três tarefas:

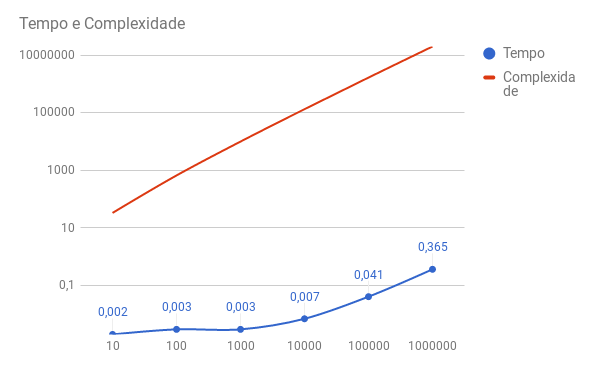
* Leitura do arquivo ‘input’ e gravação de dados em um vetor: O(n)
* QuickSort: O(n㏒n)
* Escrita dos valores do vetor no arquivo ‘output’: O(n)

Logo, a complexidade do programa é O(n㏒n) nos casos médios e nos melhores casos. Nos piores casos, a complexidade é O().

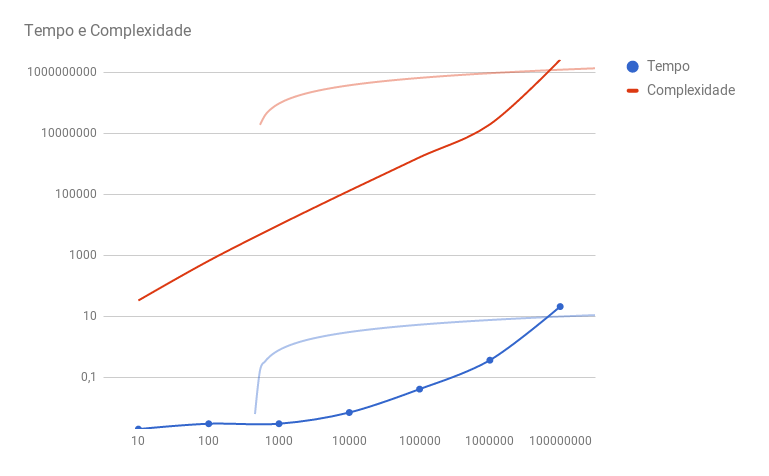
Para testar a relação do tempo de execução com o tamanho da entrada, foram feitos alguns testes com entradas aleatórias de tamanhos variados. Os tempos de cada execução foram obtidos por linha de comando (“time ./program\_name parameters…”) e foram computados somando-se os tempos “User” e “Sys” informados.

Os resultados mostraram que a relação entre o tempo de execução e o tamanho das entradas é representada por uma função n㏒n, que também caracteriza a complexidade do programa. O gráfico abaixo mostra a plotagem dos resultados.

Uma melhor representação da complexidade temporal do programa pode ser obtida com a realização de um teste de mais dados.



O mesmo gráfico é apresentado abaixo, porém com a representação da linha de tendência de crescimento das respectivas funções. Assim, torna-se possível observar o comportamento das funções que representam a relação entre os tamanhos das entradas, o tempo e a complexidade do programa, que sao essencialmente as mesmas.



# Conclusão

Baseado nas análises dos resultados dos testes, conclui-se que o desempenho do programa condisse com a complexidade dos algoritmos nele implementados.

# Referencias

“How to get execution time of a script effectively?”. 2013. Disponível em: <https://unix.stackexchange.com/questions/52313/how-to-get-execution-time-of-a-script-effectively> Acesso em: 22/06/2017.

“What do 'real', 'user' and 'sys' mean in the output of time(1)?”. 2009. Disponível em: <https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1/556411#556411> Acesso em: 22/06/2017.

1. Para entradas ordenadas ascendentemente, o QuickSort perde apenas para o algoritmo de Inserção. [↑](#footnote-ref-0)