Algoritmos de Ordenação Externa

Fernanda Maria de Souza, Guilherme A. M. Costa.

Projeto de Arquivos (PRA)

Bacharelado em Ciências da Computação

Universidade do Estado de Santa Catarina (UDESC)

***Abstract.*** *The proposed report defines and presents results of the creation and use of an external sorting algorithm to sort a set of records contained in secondary memory using one or more record fields. In addition, the proposed algorithm meets some previously defined prerequisites, which will be exposed later.*

**Resumo.** O relatório proposto define e apresenta resultados da criação e utilização de um algoritmo de ordenação externa, para efetuar a ordenação de um conjunto de registros contidos em memória secundária, utilizando um ou mais campos de registro. Além disso, o algoritmo proposto atende alguns pré-requisitos definidos anteriormente, os quais serão expostos por posterior.

**1. Contextualização**

**1.1 Ordenação Interna**

O método da ordenação interna se baseia principalmente no conceito de economia de memória interna do computador, tanto que o mesmo é utilizado quando o arquivo a ser ordenado cabe totalmente na memória principal (RAM). Seus passos para execução são divididos em três etapas: a leitura de todos os registros do arquivo, a ordenação dos registros em RAM e por fim a escrita dos registros ordenados no arquivo.

A principal diferença entre o método de ordenação interna para a externa é caracterizada pela questão de qualquer registro poder ser imediatamente acessado, enquanto na externa os registros só podem ser acessados em grandes blocos ou sequencialmente.

**1.2 Ordenação Externa**

A ordenação externa envolve arquivos compostos por um número de registros, o qual é maior do que a memória interna do computador pode armazenar. Diferentemente dos métodos de ordenação interna, a ordenação externa define que as estruturas de dados têm de levar em conta o fato de que os

dados estão armazenados em unidades de memória externa, relativamente muito mais lentas do que a memória principal.

Fatores que determinam as diferenças das técnicas de ordenação externa:

1. Custo para acessar um item é algumas ordens de grandeza maior.

2. O custo principal na ordenação externa é relacionado a transferência de dados entre a memória interna e externa.

3. Existem restrições severas de acesso aos dados.

4. O desenvolvimento de métodos de ordenação externa é muito dependente do estado atual da tecnologia.

5. A variedade de tipos de unidades de memória externa torna os métodos dependentes de vários parâmetros.

6. Assim, apenas métodos gerais serão apresentados.

**1.3.      Funcionalidade**

O algoritmo proposto opera para alocação de espaçohipoteticamente maior que a memória interna do computador com a criação aleatória de dados binários em um espaço de memória de 475 MB os quais serão subdivididos em 5 fitas. Inicialmente o algoritmo trabalha com dados desordenados e ao dividi-los em fitas, os mesmos serão ordenados pelo Quick-Sort.

**2. Construção do Algoritmo**

O Algoritmo foi construído na Linguagem C com a utilização dos seguintes arquivos: gerar\_arquivo\_main.c, gerar\_arquivo.c, gerar\_arquivo.h, ler\_arquivo\_gerado.c, ordenar\_arquivo\_main.c, ordenar\_arquivo.c, ordenar\_arquivo.h

**2.1.      Descrição**

gerar\_arquivo\_main.c - Responsável pela limitação do arquivo a ser criado por inicial (475 MB) e coletar informações de tempo de execução.

gerar\_arquivo.c - Responsável pela criação da estrutura (Dado) a qual é utilizada para criar o arquivo inicial com variáveis do tipo *float* e *unsigned long. Além disso,é criada a função escrever\_arquivo para escrever os dados num arquivo binário limitado.*

gerar\_arquivo.h - Define GB e MB como variáveis globais e mencionar a função escrever\_arquivo.

ler\_arquivo\_gerado.c - Responsável pela abertura do arquivo e posterior leitura de dados contidos com a função ler\_arquivo\_binário.

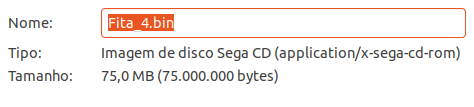
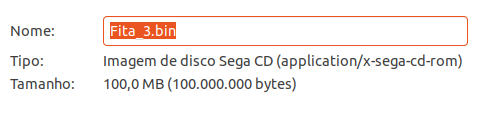
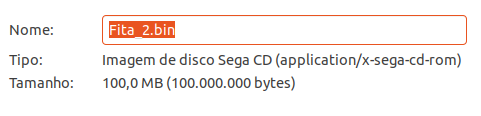
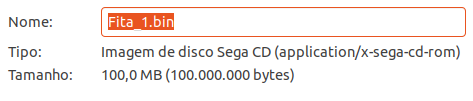
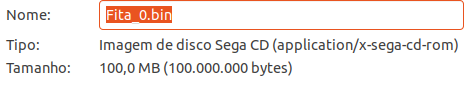
ordenar\_arquivo\_main.c - Responsável pela limitação das 5 fitas e iniciar a ordenação dos dados com o Algoritmo de Ordenação Interna Quick-Sort.

ordenar\_arquivo.c - Responsável pela ordenação das fitas com funções do Quick-Sort de empilhamento e desempilhamento

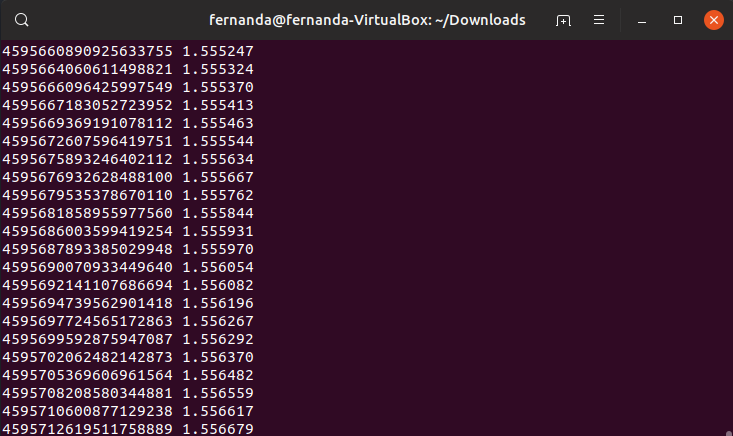
ordenar\_arquivo.h - Responsável pela definição de todas variáveis globais e criação de todo estrutura de ordenação de dados e também mencionar as funções próprias de arquivos.h.

**2.2.      Fluxo de Execução**

Primeiramente foram criadas as fitas iniciais, cada uma com tamanho máximo de 100 MB e por fim a última com os 75MB que sobraram do total.



Após a criação das fitas, é realizada a ordenação, para posterior intercalação, conforme a imagem retrata.



**2.3.      Algoritmos Utilizados**

**Quick-Sort**

**Definição e Uso**

O Quick-sort é um algoritmo de [ordenação por comparação](https://pt.wikipedia.org/wiki/Ordenação_por_comparação) [não-estável](https://pt.wikipedia.org/wiki/Ordenação_estável). O mesmo foi utilizado para ordenar as 5 fitas resultantes da ordenação.

Utilizamos o mesmo pronto com a função do C qsort(). O mesmo é executado com os seguintes passos:

- Escolha de um elemento denominado pivô.

- Particiona: rearranja a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores que ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sub listas não ordenadas.

- Recursivamente ordena a sub lista dos elementos menores e a sub lista dos elementos maiores;

**Complexidade**

-       Complexidade de Espaço:

o   Melhor Caso e Caso Médio: *O(log2n)*

o   Pior Caso: *O(log2n)*

-       Complexidade de Tempo:

o   Melhor Caso e Caso Médio: *O(n2)*

o   Pior Caso: *O(n2)*

**Intercalação Balanceada de Vários Caminhos**

A utilização do algoritmo da intercalação balanceada de vários caminhos é caracterizado pela execução de alguns passos para funcionamento geral, são eles:

- Leitura do primeiro registro de cada fita.

- Retirada do registro contendo a menor chave, armazenando o mesmo no registro de saída.

- Leitura de um novo registro da fita de onde o registro retirado é proveniente:

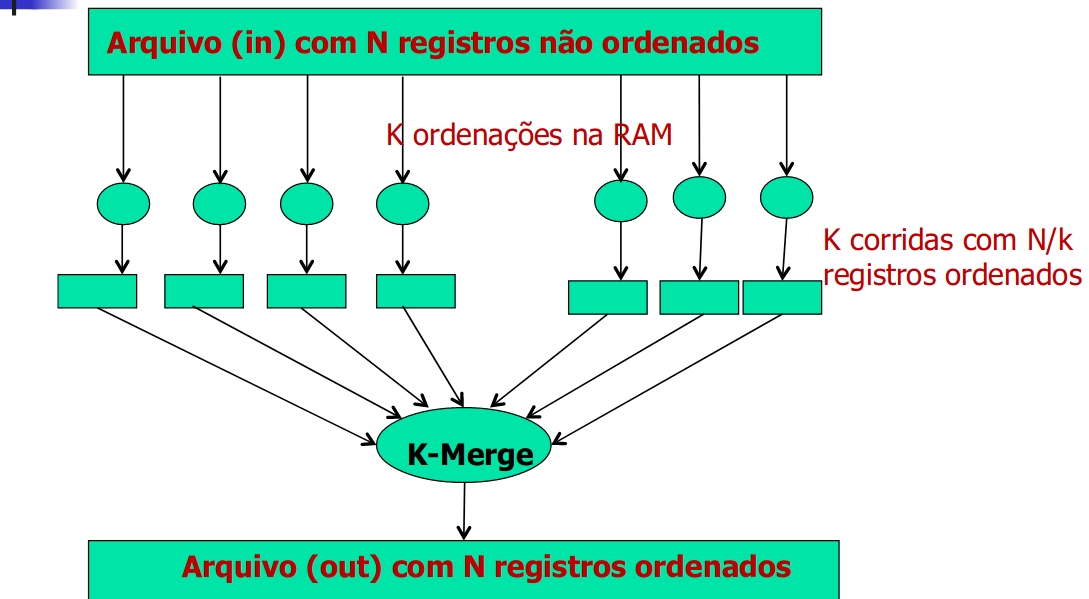
- Repetição do processo para os blocos restantes.

**K-way MergeSort**

Ao invés de considerar os registros individualmente, podemos considerar blocos de registros ordenados (corridas, ou runs) para minimizar os seeks.

Para ordenar um arquivo de tamanho arbitrário devemos utilizar a fórmula P(n) = logf(n/m), onde n é o número de registros do arquivo e m o número de registros possíveis na memória, sendo P(n) o número de passadas na fase de intercalação.

O método envolve 2 fases: geração das corridas (runs, blocos ordenados de dados), e intercalação.



**Desvantagens**

-       Utiliza funções recursivas;

-       Gasto extra de memória. O algoritmo cria uma cópia na memória para cada nível da chamada recursiva, gerando um uso extra na memória.

Fita 4



Fita 1

Fita 2

Fita 3



Fita 5

Fita 6



Memória Interna



Fita 7

Figura: Exemplo do funcionamento da intercalação balanceada de vários caminhos.

**2.4.      Execução**

Para compilar:

gcc -c gerar\_arquivo.c gerar\_arquivo\_main.c

gcc .gerar\_arquivo.o .gerar\_arquivo\_main.o -o gerador

gcc -c ordenar\_arquivo.c ordenar\_arquivo\_main.c

gcc .ordenar\_arquivo.o ordenar\_arquivo\_main.o -o ordenar

gcc -c ler\_arquivo\_gerado.c

gcc ler\_arquivo\_gerado.o -o ler

Para executar:

./gerador (para criar o arquivo de 475mb)

./ordenar out.bin (criar o arquivo de saída)

./ler out.bin (para ler e conferir êxito)

**2.5 Funções implementadas**

**2.5.1 Geração do arquivo**

No arquivo gerar\_arquivo.c é feita a criação do arquivo com 475 MB juntamente com seus campos unsigned long chave1 e f.loat chave2. Os dois registros são gerados aleatoriamente.

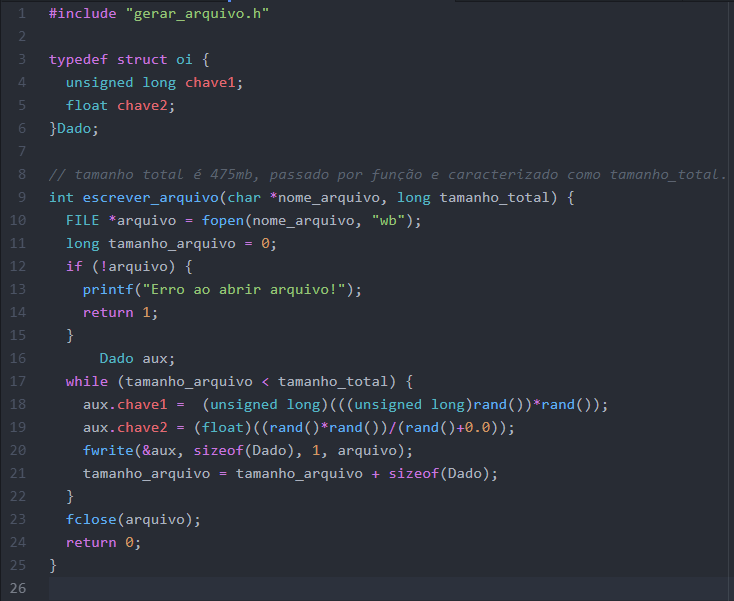


Imagem: Arquivo gerar\_arquivo.c

**2.5.2 Leitura de arquivo binário para confirmação da ordenação**

Elaboramos uma função para confirmar que a ordenação está sendo feita corretamente e que os dados estão localizados na posição correta. Dessa forma, o arquivo ler\_arquivo realiza a leitura de qualquer fita .bin.

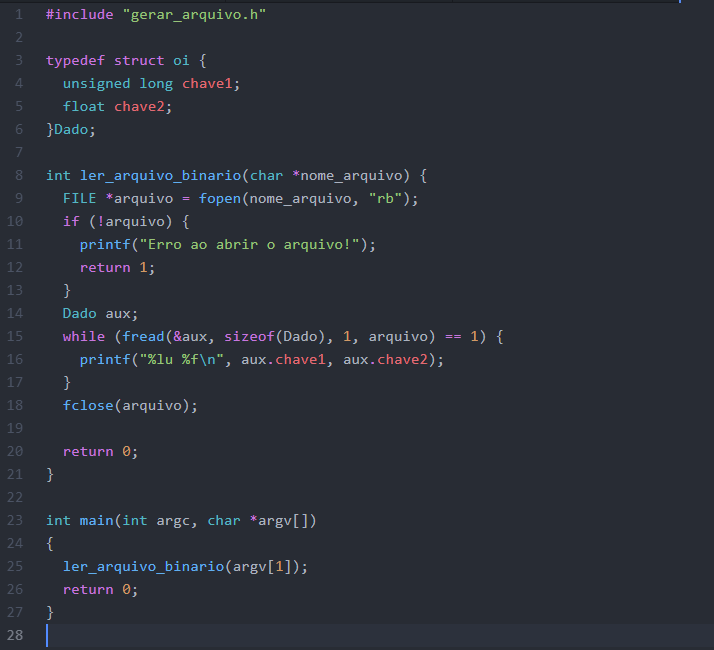


Imagem: Arquivo ler\_arquivo.c

**2.5.3 Geração das fitas auxiliares**

Conforme requisitado, de acordo com o número M de 5 fitas, fizemos uma função para criação dessas fitas para posterior utilização visando a intercalação balanceada. Os parâmetros i, file\_name, modo de abertura, tamanho atual e da execução são passados para a struct que armazena os dados das fitas.

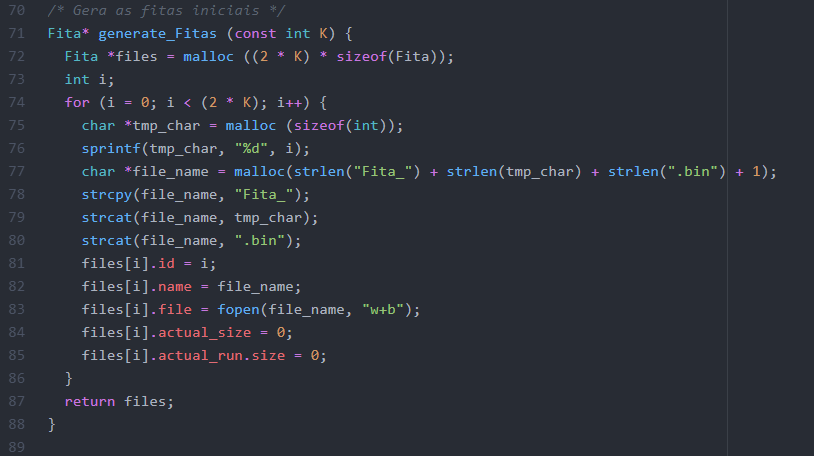
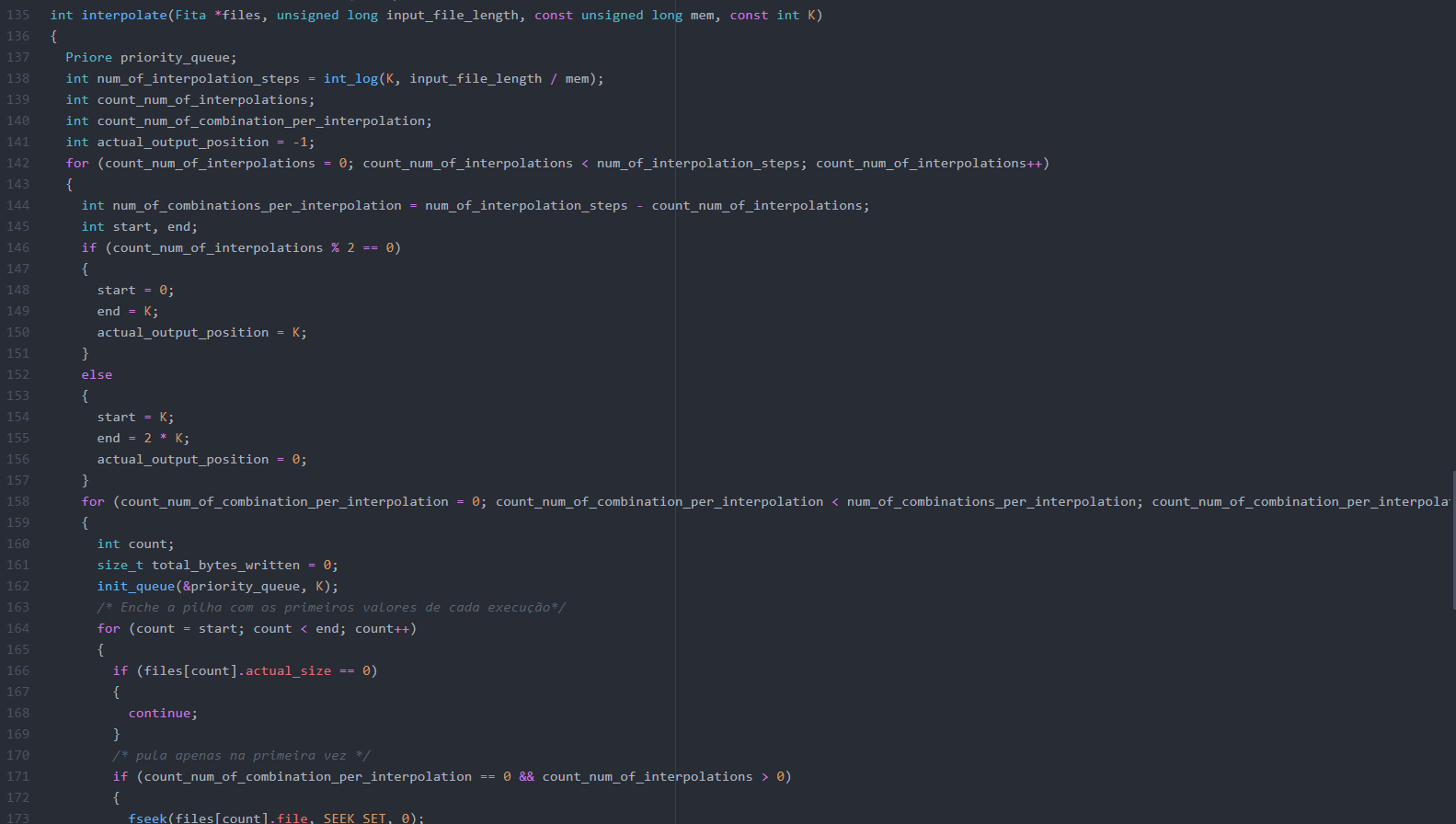
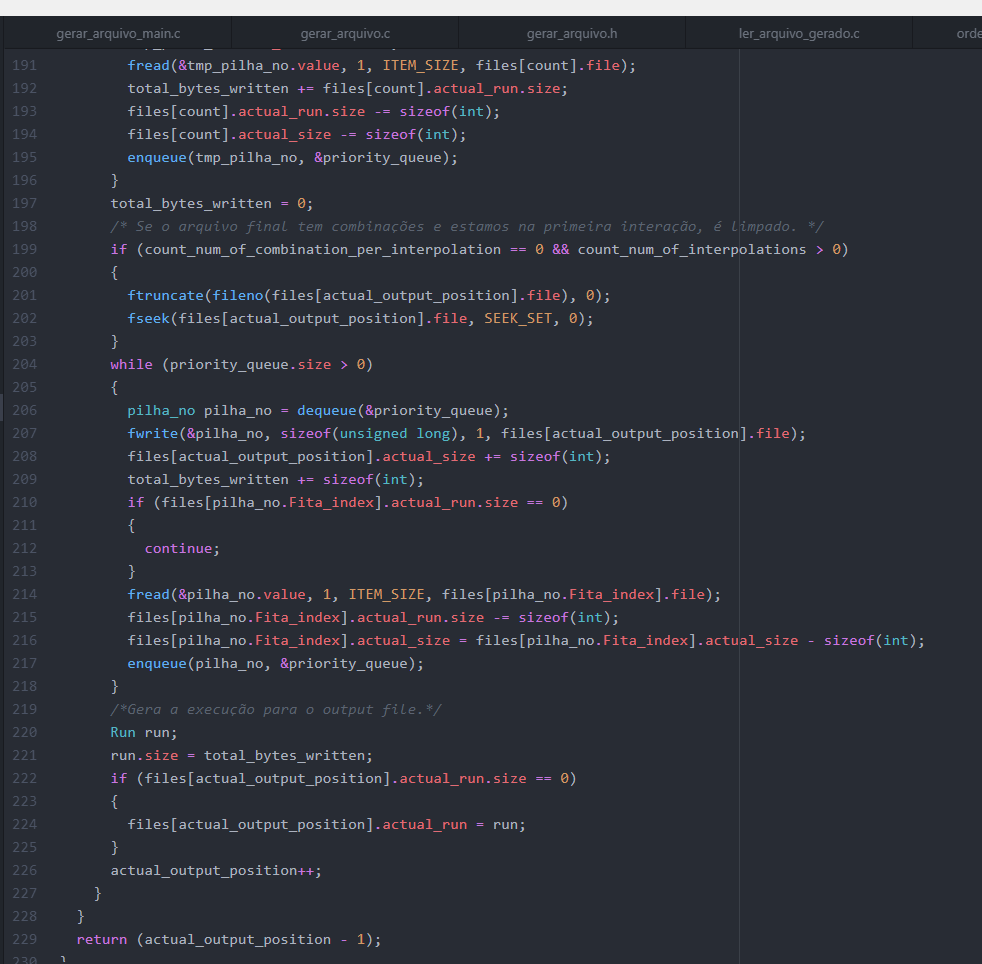


Imagem: Arquivo ordena\_arquivo.c

**2.5.3 Método de Intercalação**

Conforme explicado no tópico da intercalação balanceada de vários caminhos, a função de intercalação é realizada da seguinte maneira:





Imagens: Arquivo ordena\_arquivo.c

**2.6 Complexidade**

Tem-se que cada passagem requer a leitura e escrita de 2 arquivos, cada um com aproximadamente n/2 registros. O número de acessos em cada passagem é aproximadamente ~= 4(n/2) ~= 2n. O número de leituras e escritas de blocos em cada passagem é em torno de 2n/b, onde b é o tamanho do bloco, então, o número total de leitura e escritas de blocos em todo o processo de ordenação é em torno de (2n log n) / b, ou seja é de uma ordem O(n log n).

**2.7.      Testes**

Máquina I: (MacOS)  
Configurações: i7 2,8 GHz | 16 Gb de RAM

Máquina II: (Linux)  
Configurações: i7 2,8 GHz | 16 Gb de RAM

Máquina III: (Windows)

Configurações: i5 2,8 GHz | 8 Gb de RAM

**Tempo para gerar o arquivo de 475MB:**

|  |  |  |
| --- | --- | --- |
| **TESTES** | **MÁQUINA** | **TEMPO (em milissegundos)** |
| **1** | **I** | 5322.2800 |
| **2** | **I** | 5002.9000 |
| **3** | **II** | 5570.45900 |
| **4** | **II** | 5779.48900 |
| **5** | **III** | 6790.21000 |
| **6** | **III** | 8800.45600 |

**Tempo para ordenação:**

|  |  |  |
| --- | --- | --- |
| **TESTES** | **MÁQUINA** | **TEMPO (em milissegundos)** |
| **1** | **I** | 43692.0210 |
| **2** | **I** | 42311.6000 |
| **3** | **II** | 45205.93600 |
| **4** | **II** | 50980.40200 |
| **5** | **III** | 74566.01220 |
| **6** | **III** | 65233.89650 |

**2.6. Resultados**

**2.6.1 Resultado geral para leitura**

|  |  |  |
| --- | --- | --- |
| **MÁQUINA** | **MÉDIA** | **DESVIO PADRÃO** |
| **I** | 5150.02000 | 159,690000 |
| **II** | 5648.25000 | 104,515000 |
| **III** | 7750.26000 | 1005,123000 |
| **GERAL** | 6166.57000 | 1126,840580 |

**2.6.1 Resultado geral para ordenação**

|  |  |  |
| --- | --- | --- |
| **MÁQUINA** | **MÉDIA** | **DESVIO PADRÃO** |
| **I** | 43001.8100 | 690,210500 |
| **II** | 48093.1700 | 2887,233000 |
| **III** | 7750.26000 | 4666,057850 |
| **GERAL** | 57667.2304 | 1793,861000 |

**3.      Conclusão**

Pode-se concluir que algoritmos de ordenação externa são eficientes para dados muito grande em quantidade de Bytes, mas para uma baixa quantidade de dados são pouco eficientes pela velocidade de acesso aos dados externos.