Relatório do Trabalho Prático 2
AEDS II (CSI104)

Iago C. Nuvem[2], Igor Marques Passos[2],

[1]Instituto de Ciências Exatas e Aplicadas(ICEA) – UFOP
91.501-970 – João Monlevade – MG – Brazil

[2]Departamento de Computação e Sistemas
Universidade Federal de Ouro Preto – João Monlevade, MG – Brazil

18.2.8003

22.2.8118

1. Método de ordenação em disco

Foi escolhido o método de Merge Sort, o qual necessitou de duas funções para sua implementação: Função para dividir e ordenar e função de mesclagem.

Abaixo o exemplo de execução da função de merge com os logs de tempo para bases com 1000, 10000 e 10000 entidades desordenadas, conforme o arquivo de log.

```
Entidade: Medico (Ordenacao em disco MergeSort)
Tamanho da base: 1000
Arquivos Criados: 1
Tempo de execucao: 0.001995 segundos

Entidade: Medico (Ordenacao em disco MergeSort)
Tamanho da base: 10000
Arquivos Criados: 10
Tempo de execucao: 0.033661 segundos

Entidade: Medico (Ordenacao em disco MergeSort)
Tamanho da base: 100000
Arquivos Criados: 100
Tempo de execucao: 0.676877 segundos
```

2. Método de geração de partições ordenadas

Foi escolhido o método de Seleção natural, que necessitou da implementação dos métodos auxiliares: criaMinHeap, insereHeap, removeMin e liberaHeap. Sendo MinHeap a seguinte estrutura:

```
typedef struct {
   TMedico **dados; // Array de ponteiros para TMedico
   int capacidade;
   int tamanho;
} MinHeap;
```

Quando o heap está cheio, ele é esvaziado em um novo arquivo de partição. Abaixo o teste que realizamos com 1000, 10000 e 100000 entidades.

```
Entidade: Medico (Particoes ordenadas: Seleo Natural)
Tamanho da base: 1000
Particoes Criadas: 1
Tempo de execucao: 0.001942 segundos

Entidade: Medico (Particoes ordenadas: Seleo Natural)
Tamanho da base: 10000
Particoes Criadas: 10
Tempo de execucao: 0.014653 segundos

Entidade: Medico (Particoes ordenadas: Seleo Natural)
Tamanho da base: 100000
Particoes Criadas: 100
Tempo de execucao: 0.150299 segundos
```

3. Método de Intercalação

Foi escolhido o método de árvore de vencedores, o qual necessitou da seguinte estrutura auxiliar para os nós da árvore:

```
typedef struct NoArvoreVencedores {
   TMedico *medico;
   int indiceArquivo; // indice do arquivo de onde o medico veio
} NoArvoreVencedores;
```

Quando o heap está cheio, ele é esvaziado em um novo arquivo de partição. Abaixo o teste que realizamos com 1000 e 10000 entidades.

```
Entidade: Medico (Intercalacao: Arvore Binaria de Vencedores)
Tamanho da base: 1000
Numero de particoes: 10
Tempo de execucao: 0.009799 segundos

Entidade: Medico (Intercalacao: Arvore Binaria de Vencedores)
Tamanho da base: 10000
Numero de particoes: 100
Tempo de execucao: 0.094248 segundos
```

4. Análise de Resultados

Vamos analisar a diferença de tempo de execução nos métodos implementados, caso a caso.

4.1. Método de ordenação em disco

Para facilitar a visualização criamos um grafico do tempo de execução utilizando o arquivo de log, que pode ser visto na figura 1: Podemos perceber que o gráfico tem um comportamento que tende a ser exponencial.

4.2. Método de geração de partições ordenadas

Para facilitar a visualização criamos um grafico do tempo de execução utilizando o arquivo de log, que pode ser visto na figura 2: Podemos perceber que o gráfico tem um comportamento que tende a ser exponencial.

4.3. Método de Intercalação

Para facilitar a visualização criamos um grafico do tempo de execução utilizando o arquivo de log, que pode ser visto na figura 3:

Como não conseguimos realizar testes com 100000 entidades nao podemos afirmar certamente o comportamento exponencial da função.

Após a visualização dos gráficos, torna-se visivel a semelhança no comportamento das funções implementadas, as quais tendem à ser exponenciais devido ao alto custo computacional da manipulação em disco.

Figure 1. Análise Merge Sort

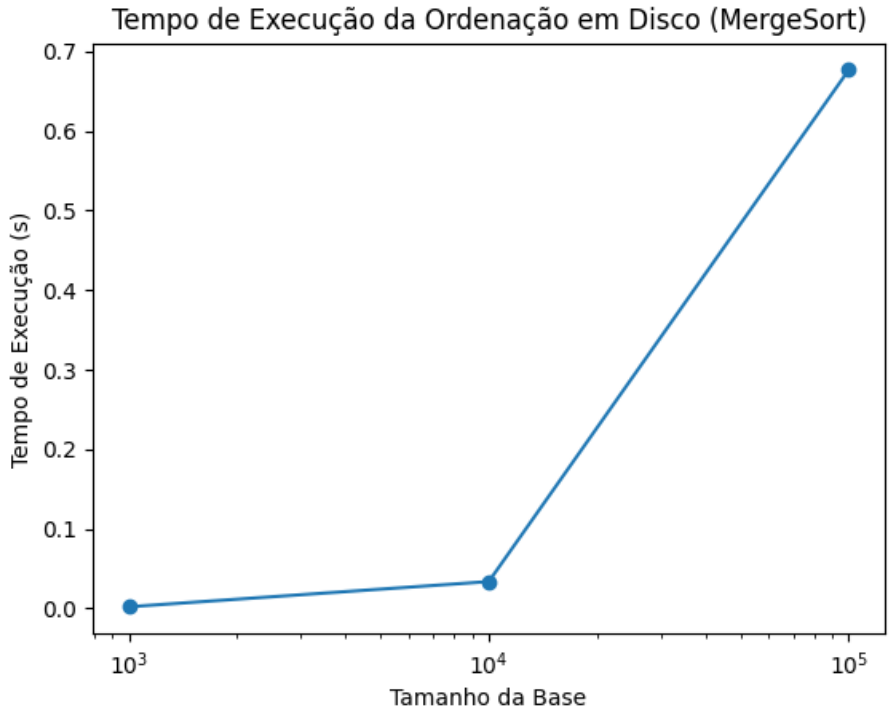

Figure 2. Análise Seleção Natural

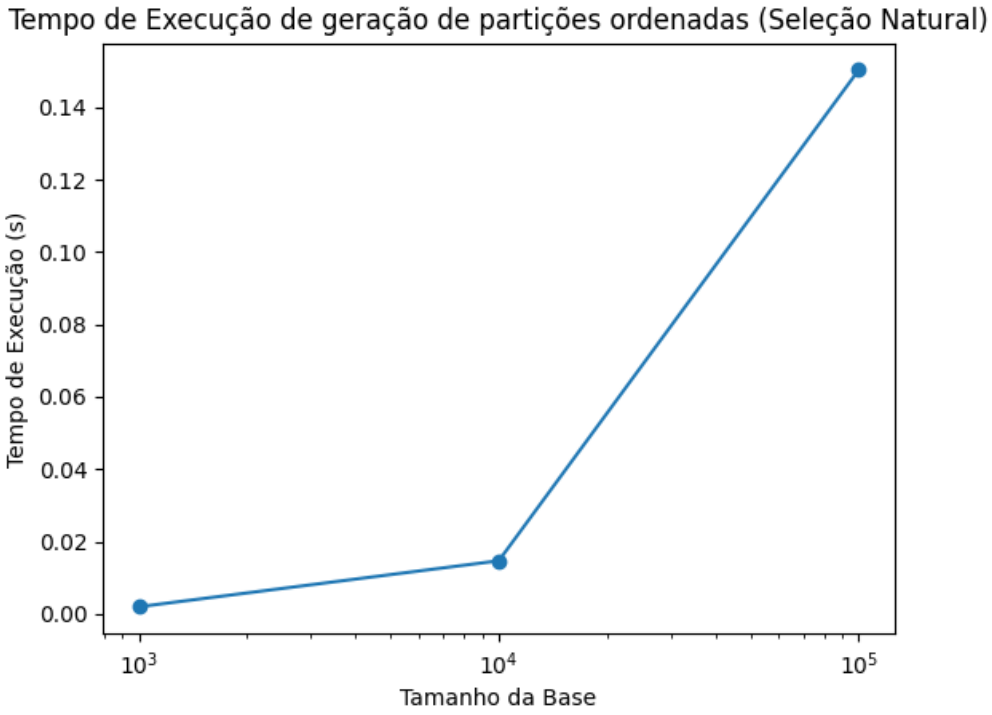

Figure 3. Análise Arvore de Vencedores