Relatório do Trabalho Prático 2 - BCC203

👥 **Alunos**
- Lucas de Araújo | 18.2.4049
- Fabio Henrique Alves Fernandes | 19.1.4128
- João Francisco Bittencourt | 19.1.4016
- Fábio Henrique Soares Dias | 19.1.4008

🎓 **Professor Doutor**
- Guilherme Tavares De Assis

🏫 **Instituição**
- Universidade Federal de Ouro Preto | Departamento de Computação

📅 **Data De Entrega e Apresentação**
- 12 de Agosto de 2021

Introdução

O relatório a seguir, referente ao segundo trabalho prático da disciplina de Estrutura de Dados II (BCC-203) ministrada pelo Professor Doutor Guilherme Tavares De Assis, tem como principal objetivo relatar a análise dos algoritmos e técnicas de ordenação externa apresentadas previamente em aulas assíncronas durante o ano de 2021.

Algoritmos e Técnicas

Os algoritmos e técnicas apresentadas são:

- Intercalação Balanceada de Vários Caminhos - 2F Fitas
- Intercalação Balanceada de Vários Caminhos - F + 1 Fitas (Não Finalizado)
- Quicksort Externo

Ordem dos Arquivos

Os arquivos derivados do "PROVAO.TXT" que cada um destes métodos poderia ler possuía a seguinte possibilidade de ordem:

- Ordem Ascendente
- Ordem Decrescente
- Ordem Aleatória

Organização do Projeto

As pastas com os arquivos deste trabalho prático foram organizadas da seguinte forma:

- **Src**: Possui as implementações (.cpp), os cabeçalhos (.hpp) e também é a saída dos arquivos binários gerados
- **Input**: Possui os inputs utilizados para geração de testes e observados nos resultados das análises
- **Scripts**: Possui os programas em Python e Bash utilizados para facilitar o processo de geração de testes e observação das análises

Objetivos

Realizar a implementação dos métodos intercalação balanceada de vários caminhos (2f fitas), intercalação balanceada de vários caminhos (f+1 fitas) e quicksort externo, sendo que para os métodos de intercalação balanceada de vários caminhos a técnica adotada para a geração de blocos ordenados deve ser a técnica de seleção por substituição. Além de realizar estudos de complexidade de desempenho destes métodos, utilizando como parâmetros as transferências e leituras realizadas entre as memórias principal e secundária, a quantidade de comparações entre valores do campo de ordenação dos registros e o tempo de execução.

Processo de Desenvolvimento

Durante o desenvolvimento dos algoritmos mencionados anteriormente, foram tomadas uma série de decisões que visavam como objetivo facilitar o processo de desenvolvimento e também o processo de análise dos mesmos.

Scripts

Como forma de facilitar a realização e visualização do resultado dos testes, foram utilizados dois programas implementados na linguagem *Python* em ambientes virtuais. A utilização de

ambientes virtuais permite a facilitação de instalação de bibliotecas externas utilizadas no computador daquele que deseja executar os programas e também reduz a quantidade de problemas relacionados ao versionamento do interpretador Python que se tenha utilizado durante a criação do programa. No caso deste trabalho, foi utilizado o Python na versão 3.8

- **Gerador de Inputs**: Responsável por gerar de forma combinatória e também escrever nos arquivos, os inputs que foram utilizados nos testes e análises para cada método
- **Compiladores**: Possui dois arquivos (.bash e .bat) responsáveis por realizar o processo de compilação automaticamente do programa de acordo com sistema operacional que o usuário estiver utilizando em sua máquina

Intercalação balanceada de vários caminhos (2F fitas)

Para o início do método de intercalação de vários caminhos com 2f fitas com seleção por substituição (Heap), é selecionada uma quantidade de linhas do `PROVAO.TXT` ou dos arquivos binários previamente já ordenados (`PROVAO_CRESCENTE` ou `PROVAO_DECRESCENTE`). Após selecionado o arquivo, é copiada a determinada quantidade de linhas deste arquivo para um novo arquivo binário chamado `temp` e este será o arquivo acessado pelo método.

Intercala2F

O inicio do processo de intercalação se dá pela função `intercala2F` que recebe o arquivo a ser ordenado (temp), a quantidade de linhas do mesmo e a estrutura de estátisticas (responsável por fazer as medições de transferências, comparações e afins).

Dentro desta mesma função, será criado dois vetores de fitas de entrada e fitas de saida, com a determinada quantidade de fitas especificadas no enunciado do trabalho. Estes vetores são os responsáveis por armazenar: Um ponteiro para um arquivo, a quantidade de blocos dentro desta respectiva fita (arquivo) e o número de registros que estarão dentro destes blocos.

Junto a estes vetores, também é criado um vetor de registros do tipo Aluno, vetor este que na verdade é um Heap e que será responsável por realizar as ordenações em memória principal durante a execução do programa

As fitas de saída e entrada são então inicializadas pela função de `inicializaFitas` e é chamada a função de criação de blocos para intercalação

criaBlocosIntercalação

Nessa função será feita a criação dos blocos utilizados no processo de intercalação. Cada bloco é criado assim que um vetor (heap) estiver com todos os seus valores marcados (no caso desse programa, quando todos os valores de notas estiverem acima de um outro determinado valor).

Primeiramente é feito um loop do tamanho da quantidade disponível de memória interna para os registros (REG) é feita a leitura de um aluno no arquivo binário e salvo no vetor. Após isso, a variável marcado recebe a nota desse respectivo aluno lido (essa variável será responsável por dizer se determinado valor estar marcado ou não na heap)

Logo após, é iniciado um while loop que será executado até que todas as linhas do arquivo binário tenham sido lidas, ou seja, até que a variável seja igual a zero.

No começo do mesmo, é chamado a função de heapsort com heap para ordenar os valores presentes na heap.

Após isto, é feito um heap que inicia a criação de fitas mediante a quantidade de linhas do arquivo que está sendo ordenado. A variável count serve para contar esta quantidade de arquivos. O nome do arquivo então é gerado dinamicamente, armazenado na variável nomeArq e depois atribuído a respectiva fita. Porém caso o arquivo já exista, será feita apenas a atribuição dele para a determinada fita e o posicionamento do ponteiro no início do mesmo

É feita então a verificação se todos os elementos da heap já se encontram marcados:

- Caso sim:
    - Todos serão desmarcados
    - A quantidade de blocos será incremententada
    - Um vetor auxiliar será criado, com o tamanho do contador dividido pelo número de fitas mais um
    - Esse vetor terá o tamanho correto de blocos presentes em cada fita e servirá para que o número de registros de cada fita possua o devido valor
- Caso não:
    - É criado uma heap individual com o menor registro do vetor de heaps
    - O registro dessa heap é escrito na fita
    - Caso a quantidade de linhas NÃO seja menor que a quantidade disponível no vetor de registros, será feito o processo de marcação do valor da heap (atribuição de valor length) ou simplesmente a atribuição de nota
    - Caso a quantidade de linhas SEJA menor que a quantidade disponível no vetor de registros, será pego o último registro e o mesmo será colocado na primeira posição do vetor de heaps (regHeaps)

São formados assim, os blocos ordenados nas fitas de entrada para a intercalação

Intercalação

O método da intercalação propriamente dita recebe as fitas já inicializadas e construídas pela função anterior, um vetor de alunos e a variável que define quais fitas serão usadas como entrada e saída.

A função começa verificando a possibilidade de ser uma fita única e já ordenada, já encerrando o algoritmo, e caso contrário, iniciando as fitas e variáveis necessárias para a intercalação dos blocos ordenados.

Em seguida, o algoritmo lê o primeiro itens de cada bloco com a ajuda de um contador, armazenando-os em um registro, cria as fitas de saída e as organiza para receber os itens, lendo então a menor nota do registro e escrevendo-a numa fita de saída, posteriormente lendo um próximo registro da fita de entrada, até concluir a leitura de todas as fitas de entrada.

Quicksort externo

O QuickSort externo tem o mesmo princípio do método feito para ordenação em memória interna. O método particiona o arquivo a ser ordenado em "sub arquivos" a partir de chamadas recursivas e para a partição do arquivo, é utilizada uma área de memória interna para o armazenamento do pivô, que no caso é um vetor de tamanho 10. O pivô apresenta os valores ordenados em sua posição final dentro da ordenação geral do arquivo, enquanto que a área esquerda apresenta os valores menores que o pivô, e a área direita representa os valores maiores que os contidos no pivô. Tal particionamento é essencial para a execução do algoritmo.

Gráficos e Tabelas dos Métodos

Intercalação balanceada de vários caminhos (2F fitas)

Tabela de Testes

INTERCALAÇÃO 2F FITAS - TEMPO TOTAL (ms)				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
296.0	320.0	1796.0	857.0	3757.0

Tempo Final de Execução na Ordem Ascendente

INTERCALAÇÃO 2F FITAS - COMPARAÇÕES TOTAL				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
5102	56402	569402	5762996	27282579

Comparações Totais na Ordem Ascendente

INTERCALAÇÃO 2F FITAS - TRANSFERÊNCIAS TOTAL				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
200	2000	20000	200000	943410

Transferências Totais na Ordem Ascendente

INTERCALAÇÃO 2F FITAS - TEMPO TOTAL (ms)				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
14.0	40.0	137.0	1069.0	6134.0

Tempo Total na Ordem Descendente

INTERCALAÇÃO 2F FITAS - COMPARAÇÕES TOTAL				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
10702	117505	1177420	10923664	51363001

Comparações Totais na Ordem Descendente

INTERCALAÇÃO 2F FITAS - TRANSFERÊNCIAS TOTAL				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
200	6000	60000	600000	3773640

Transferências Totais na Ordem Descendente

INTERCALAÇÃO 2F FITAS - TEMPO TOTAL (ms)				
ARQUIVO ALEATORIO				
100	**1000**	**10000**	**100000**	**471705**
12.0	60.0	199.0	1268.0	5660.0

Tempo Total na Ordem Aleatória

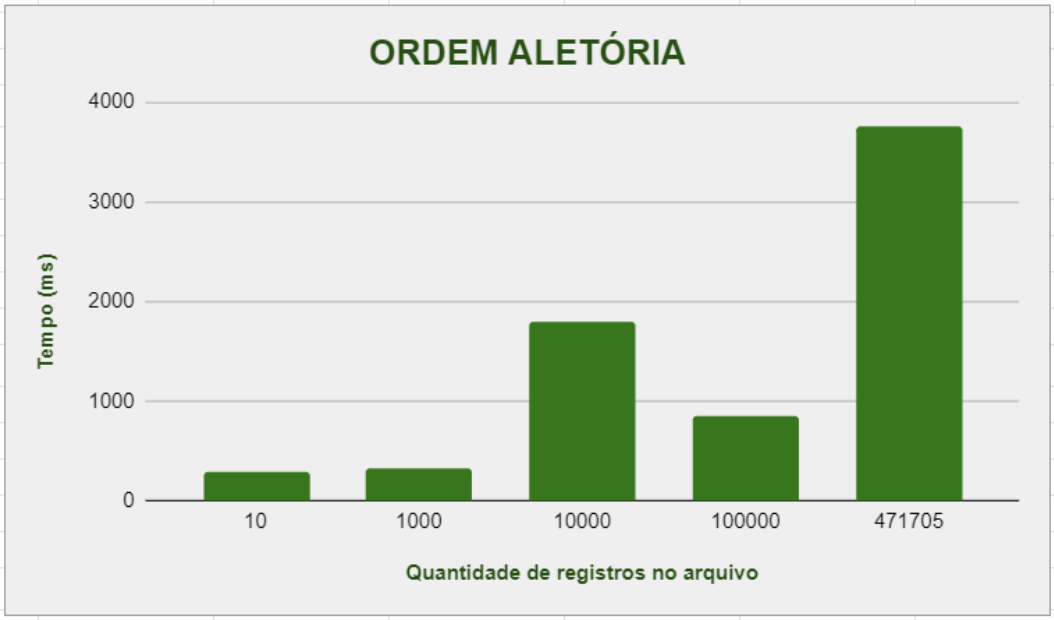
INTERCALAÇÃO 2F FITAS - COMPARAÇÕES TOTAL				
ARQUIVO ALEATORIO				
100	**1000**	**10000**	**100000**	**471705**
11592	146185	1591050	17489312	87913036

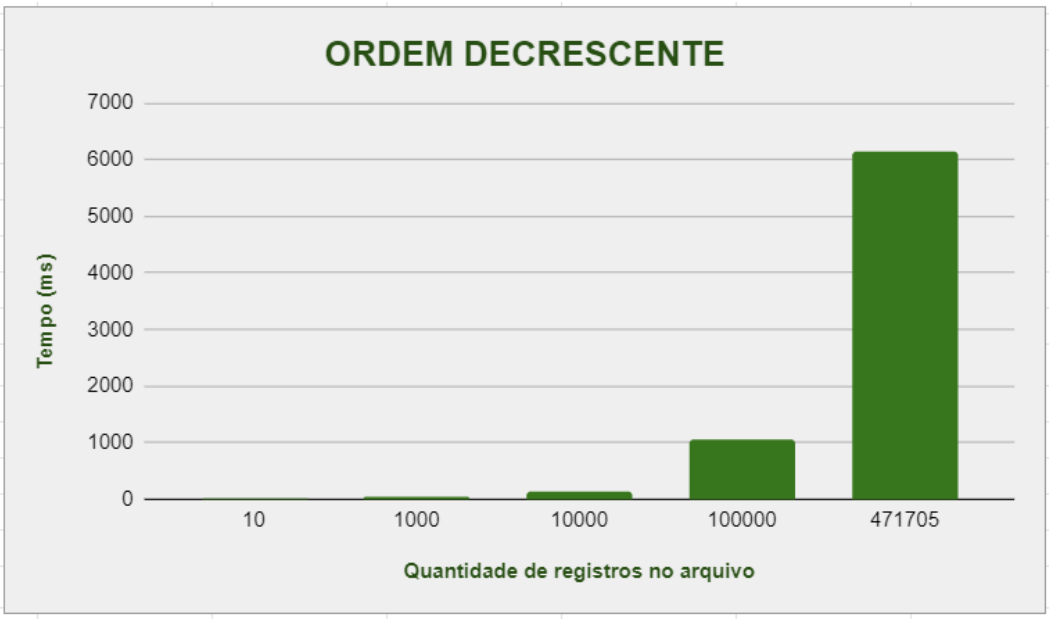
Comparações Totais na Ordem Aleatória

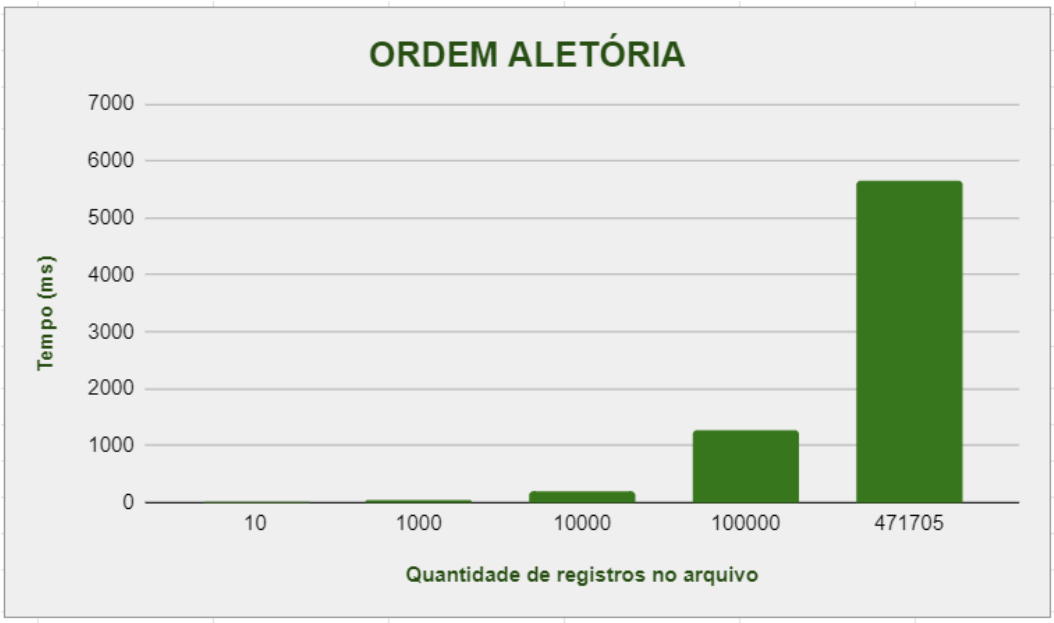
INTERCALAÇÃO 2F FITAS - TRANSFERÊNCIAS TOTAL				
ARQUIVO ALEATORIO				
100	**1000**	**10000**	**100000**	**471705**
200	6000	60000	800000	4717050

Transferências Totais na Ordem Aleatória

Gráficos Comparativos







Quicksort externo

Tabela de Testes

QUICKSORT - TEMPO TOTAL (ms)				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
0.0	7.0	53.0	397.0	1867.0

Tempo Total na Ordem Ascendente

QUICKSORT - COMPARAÇÕES TOTAL				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
3393	35343	354843	4046159	19094564

Comparações Totais na Ordem Ascendente

QUICKSORT - TRANSFERÊNCIAS TOTAL				
ARQUIVO ASCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
200	2000	20000	200000	943410

Transferências Totais na Ordem Ascendente

QUICKSORT - TEMPO TOTAL (ms)				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
1.0	7.0	63.0	405.0	1861.0

Tempo Total na Ordem Descendente

QUICKSORT - COMPARAÇÕES TOTAL				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
4051	40811	405875	4050393	19097304

Comparações Totais na Ordem Descendente

QUICKSORT - TRANSFERÊNCIAS TOTAL				
ARQUIVO DESCENDENTEMENTE				
100	**1000**	**10000**	**100000**	**471705**
200	2000	20000	200000	943410

Transferências Totais na Ordem Descendente

QUICKSORT - TEMPO TOTAL (ms)				
ARQUIVO ALEATORIO				
100	1000	10000	100000	471705
6.0	37.0	298.0	3102.0	16571.0

Tempo Total na Ordem Aleatória

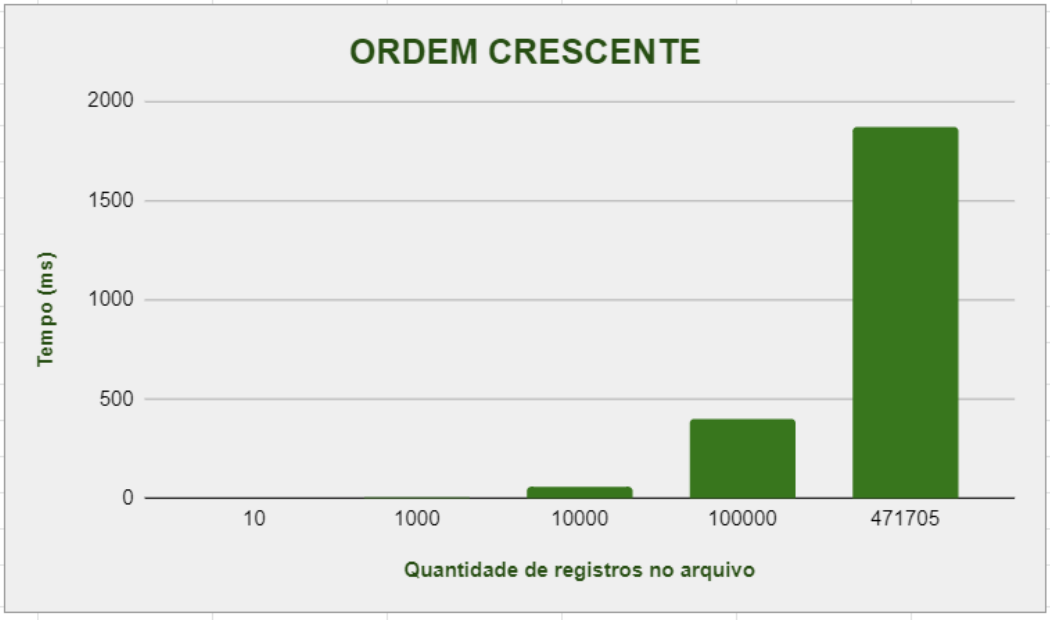
QUICKSORT - COMPARAÇÕES TOTAL				
ARQUIVO ALEATORIO				
100	1000	10000	100000	471705
5304	68402	779838	8607981	43478012

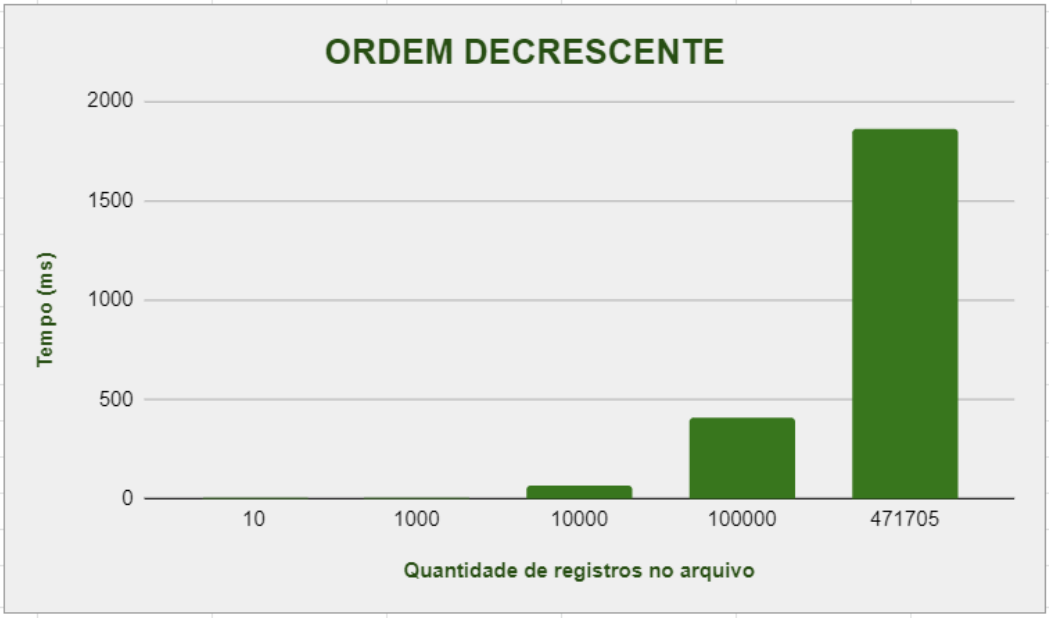
Comparações Totais na Ordem Aleatória

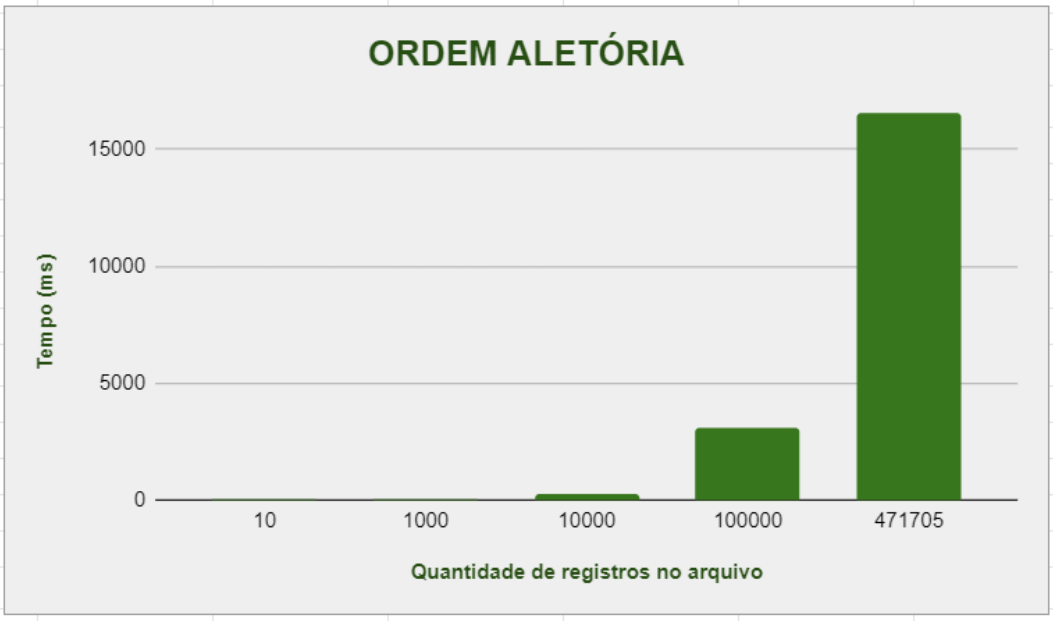
QUICKSORT - TRANSFERÊNCIAS TOTAL				
ARQUIVO ALEATORIO				
100	1000	10000	100000	471705
582	10952	151390	1774800	8747120

Transferências Totais na Ordem Aleatória

Gráficos Comparativos







Análises

Durante as análises de ordenação externa, é sempre importante observar que o hardware dos computadores onde estão sendo executados os testes exerce uma influência sobre os resultados observados, principalmente o tempo gasto durante a execução do método. Dispositivos como SSD's podem acarretar em menores tempos de transferência para a memória principal durante a execução dos métodos por conta da sua arquitetura de funcionamento baseada em células elétricas, diferente do disco rígido onde é utilizada uma gravação magnética. Além disso, os testes podem ser influenciados pelo armazenamento da cache quando executados de maneira consecutiva, como foi o caso deste trabalho.

Feita essas observações, a seguir estão relatados de maneira individual as análises observadas em cada método.

Intercalação balanceada de vários caminhos (2F fitas)

A partir de uma análise dos testes do método em questão, podemos notar que se tratando do tempo gasto, ele possui um desempenho melhor quando se tem um arquivo ordenado descendentemente e aleatoriamente para até 10000 registros, acima disso a ordenação de um arquivo ascendentemente se torna mais rápida.

Se tratando da quantidade de comparações e transferências que o algoritmo realiza durante sua execução, o melhor desempenho se deu com um arquivo ordenado crescentemente, justamente por ele já estar ordenado.

Quicksort externo

O QuickSort externo quando recebe um arquivo ordenado seja de forma ascendente ou descendente tem um número de transferências iguais, porém, quando descrescente, o número de comparações é maior. Isso se dá por conta do sistema de particionamento pois ele separa porções maiores e menores que os elementos contidos no pivô. Em questão de tempo, quando o arquivo está na forma aleatória, o algoritmo leva mais tempo para completar a ordenação comparado a arquivos crescentes ou decrescentes. Um outro fato interessante é que os arquivos ordenados de forma aleatória possuem um número maior de comparações e transferência

Conclusão

Após feita as análises dos métodos descritos anteriormente, podemos citar algumas conclusões a respeito dos mesmos.

É possível concluir que o método mais rápido, tendo como base as análises do tempo total de execução, quando temos um arquivo ordenado de forma ascendente ou descendente, é o Quicksort Externo. No entanto, o mesmo não se mostrou tão eficiente em questão de tempo quando era solicitado a ordenação de um arquivo ordenado de forma aleatória.

Ao contrário do método Quicksort Externo, a Intercalação Balanceada de Vários Caminhos (2f fitas), por mais que seja o método principal de ordenação de pesquisas, se mostrou mais eficiente em termos de tempo de execução, apenas para arquivos ordenados aleatoriamente para os casos em que os regitros fossem maiores que 10000.

Dado o objetivo principal do trabalho, que basicamente consistia em executar os conhecimentos aprendidos em sala e implementar os métodos de pesquisa externa, a fim de

satisfazer a problemática do trabalho. Pode-se inferir que, com a exceção dos problemas enfrentados com a Intercalação Balanceada com f + 1 fitas, este objetivo foi concluído, já que não saímos apenas sabendo como implementar os demais métodos, mas passamos a entender melhor o seu funcionamento e as vantagens e desvantagens de cada um.

Referências

Tavares, Guilherme. **Ordenação Externa**. Disponível em: <http://www.decom.ufop.br/guilherme/BCC203/geral/ed2_ordenacao-externa.pdf>. Acesso em 03 de agosto de 2021.