UNIVERSIADE FEDERAL DE OURO PRETO – *UFOP*

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Fernandes Niquini

Filipe Ramos

Rafael Diniz

Trabalho Prático I – Pesquisa Externa

Ouro Preto

Outubro de 2020

Gabriel Fernandes Niquini

Filipe Ramos

Rafael Diniz

Trabalho Prático I – Pesquisa Externa

Ouro Preto

Outubro de 2020

RESUMO

O trabalho consiste em implementar e desenvolver a lógica de um programa em C/C++ de ordenação externa, com o intuito de exemplificar e praticar o conteúdo, além de um estudo sobre o desempenho dos métodos: Intercalação balanceada de vários caminhos, Substituição por seleção e Quicksort.

# 

# INTRODUÇÃO DOS METODOS

**1.1 - Ordenação Externa**

Consiste em ordenar de arquivos maior que a memória interna possível. Seus métodos são diferentes dos que ordenação interna, seus algoritmos devem diminuir o número de acessos às unidades externa.

- Ordenação por intercalação:

É o método mais importante de ordenação externa é o de ordenação por intercalação. Intercalar significa combinar dois ou mais blocos ordenados em um bloco único. Utilizada para auxiliar na ordenação;

- Foco dos algoritmos:

Reduzir os números de vezes que se passa por um arquivo, sendo assim, uma boa medida de complexidade dos algoritmos é o número de vezes que um item é lido ou escrito na memória interna. Um bom número é >= 10;

- Estratégia geral dos métodos:

1 - Quebrar o arquivo em blocos no espaço de memória interna disponível;

2 – Ordenar cada bloco na memória interna;

3 – Intercalar os blocos ordenados, fazendo várias passadas sobre o arquivo. \*Cada passada cria blocos ordenados cada vez maiores, até que o arquivo esteja totalmente ordenado.

**1.2 - Intercalação balanceada de Vários Caminhos**

- Fase de criação dos blocos ordenados envolvem:

1- Quebra do arquivo em blocos do tamanho da memória interna disponível;

2- Ordenação de cada bloco na memória interna.

- Fase de intercalação envolve:

1- Leitura do primeiro registro de cada fita;

2- Retirada do registro contendo a menor chave, armazenando-o em uma fita de saída;

3- Leitura de um novo registro da fita de onde o registro é proveniente.

3-1 Ao ler o terceiro registro de um dos blocos, a fita correspondente fica inativa;

3-2 A fita é reativada quando os terceiros registros das outras fitas forem lidos;

3-3 Neste momento, um bloco de nove registros ordenados foi formado na fita de saída.

4- Repetir o processo para os blocos restantes.

**1.3 - Implementação por meio de substituição por seleção**

A implementação do método anterior (intercalação balanceada) pode ser feita utilizando filas de prioridade. As fases de quebra e intercalação podem ser implementadas de forma eficiente e elegante, substituindo o menor item existente na memória interna pelo próximo item da fita de entrada.

Estrutura ideal para a implementação: heap

Operação:

- Retirar o menor item da fila;

- Colocar um novo item em seu lugar;

- Reconstituir a propriedade do heap.

Processo de funcionamento para gerar os blocos ordenados:

- M itens são inseridos na fila de prioridades inicialmente vazia;

- O menor item da fila de prioridade é substituído pelo próximo item de entrada. \*Se o próximo item é menor do que o que está saindo, então ele deve ser marcado como membro do próximo bloco, sendo tratado como o maior tem do bloco atual.

- Quando o item marcado vai para o início da fila, o bloco atual é encerrado e um novo bloco de ordenação é criado.

Após gerados os blocos ordenados, faz-se intercalação deles utilizando fila de prioridades:

- Monte uma fila de prioridade de tamanho F a partir dos primeiros itens de cada um do F blocos ordenados;

- Repita o processo abaixo até que não haja mais itens nos blocos ordenados:

1- Substitua o item do topo da fila de prioridades, escrevendo-o em uma fita de saída, pelo próximo item do mesmo bloco do item que está sendo substituído;

2- Reconstitua a propriedade da fila de prioridades.

**1.4 Quicksort externo:**

- Proposto em 1980 por Monard, o algoritmo utiliza o paradigma de divisão e conquista. Ele ordena *in situ* um arquivo A = {R1, ..., Rn} de n registros, esses registros se encontram em memória secundária de acesso randômico.

- O algoritmo utiliza somente O (log n) unidades de memória interna, não necessitando de qualquer memória externa adicional.

Ordenação**:**

- Para ordenar o arquivo A, o algoritmo:

-Divide A em:

{R1, ..., Ri} <=Ri+1<=Ri+2<=...<=Rj-2<=Rj-1<= {Rj, ..., Rn}

-E chama recursivamente os arquivos gerados:

A1 = {R1, ..., Ri} e A2 = {Rj, ..., Rn}

- Os registros {Ri+1, ..., Rj-1} ordenados são o pivô do algoritmo, encontrando-se na memória interna durante a execução do mesmo, os subarquivos gerados possuem os registros maiores que o último registro e menores que o primeiro.

- Para a partição do arquivo, é utilizada uma área de memória interna para armazenar o pivô, e essa área é >=3.

- Considerar que, deve ser ordenado o subarquivo de menor tamanho inicialmente, subarquivos vazios ou com registro único são ignorados e caso os arquivos de entrada possuam no máximo (j – i – 1) registros, ele é ordenado em etapa única.

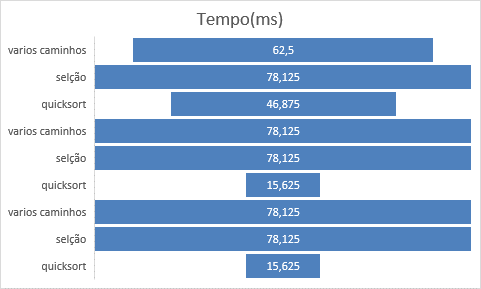
# Metodos

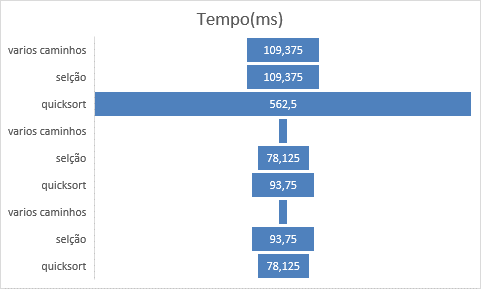
## Tabela:

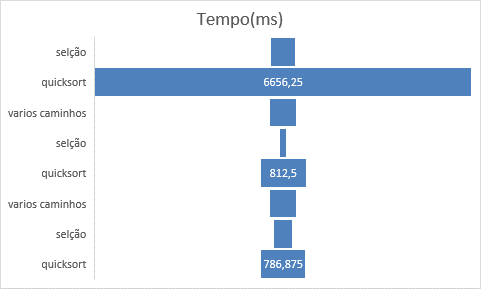
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Elementos | Ordenação | Método | Tempo (ms) | Comparações | Trasferencias |
| 100 | aleatório | vários caminhos | 78,125 | 2838 | 390 |
| 100 | aleatório | seleção | 78,125 | 1053 | 400 |
| 100 | aleatório | quicksort | 15,625 | 5174 | 582 |
| 100 | crescente | vários caminhos | 62,5 | 2490 | 390 |
| 100 | crescente | seleção | 78,125 | 306 | 200 |
| 100 | crescente | quicksort | 15,625 | 4436 | 200 |
| 100 | decrescente | vários caminhos | 62,5 | 2512 | 390 |
| 100 | decrescente | seleção | 78,125 | 828 | 400 |
| 100 | decrescente | quicksort | 0 | 4296 | 200 |
| 1000 | aleatório | vários caminhos | 62,5 | 37386 | 5900 |
| 1000 | aleatório | seleção | 78,125 | 19105 | 6000 |
| 1000 | aleatório | quicksort | 46,875 | 70078 | 10952 |
| 1000 | crescente | vários caminhos | 78,125 | 24900 | 5900 |
| 1000 | crescente | seleção | 78,125 | 3006 | 2000 |
| 1000 | crescente | quicksort | 15,625 | 46736 | 2000 |
| 1000 | decrescente | vários caminhos | 78,125 | 30140 | 5900 |
| 1000 | decrescente | seleção | 78,125 | 10568 | 6000 |
| 1000 | decrescente | quicksort | 15,625 | 45684 | 2000 |
| 10000 | alheatório | vários caminhos | 109,375 | 464454 | 79000 |
| 10000 | alheatório | seleção | 109,375 | 268485 | 88000 |
| 10000 | alheatório | quicksort | 562,5 | 914153 | 15390 |
| 10000 | crescente | vários caminhos | 12,5 | 339000 | 79000 |
| 10000 | crescente | seleção | 78,125 | 30006 | 20000 |
| 10000 | crescente | quicksort | 93,75 | 469736 | 20000 |
| 10000 | decrescente | vários caminhos | 12,5 | 348605 | 79000 |
| 10000 | decrescente | seleção | 93,75 | 136715 | 88000 |
| 10000 | decrescente | quicksort | 78,125 | 459698 | 20000 |
| 100000 | aleatório | vários caminhos | 468,75 | 5546624 | 990000 |
| 100000 | aleatório | seleção | 453,125 | 3612946 | 1000000 |
| 100000 | aleatório | quicksort | 6656,25 | 10864527 | 1774800 |
| 100000 | crescente | vários caminhos | 484,375 | 3840301 | 990000 |
| 100000 | crescente | seleção | 125 | 303570 | 200000 |
| 100000 | crescente | quicksort | 812,5 | 4695619 | 200000 |
| 100000 | decrescente | vários caminhos | 468,75 | 3943044 | 990000 |
| 100000 | decrescente | seleção | 343,75 | 1346389 | 800000 |
| 100000 | decrescente | quicksort | 786,875 | 4599974 | 200000 |
| 471705 | aleatório | vários caminhos | 2375 | 27821736 | 5613289 |
| 471705 | aleatório | seleção | 2140,645 | 19959385 | 5660460 |
| 471705 | aleatório | quicksort | 3645,3125 | 55607228 | 8747120 |
| 471705 | crescente | vários caminhos | 3781,25 | 19864287 | 5613289 |
| 471705 | crescente | seleção | 1812,5 | 1424543 | 933410 |
| 471705 | crescente | quicksort | 5250 | 23866054 | 1355966 |
| 471705 | decrescente | vários caminhos | 2406,25 | 19361777 | 5673289 |
| 471705 | decrescente | seleção | 1406,25 | 7545670 | 3773640 |
| 471705 | decrescente | quicksort | 3843,75 | 21699684 | 943410 |

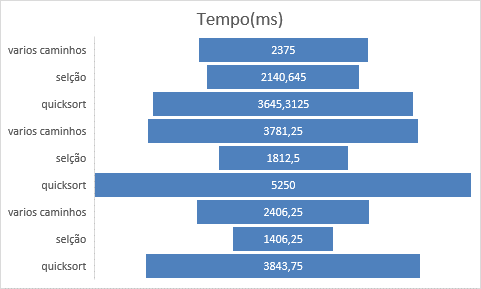
## Gráficos:

## 









# CONSIDERAÇÕES FINAIS

Todos os métodos têm suas utilidades e funções, mesmo que uns sejam melhores que outros para funções específicas, sendo assim, não é possível descartar nenhum dos métodos para atividades diversas.

Na teoria as arvores B e B\* tem a vantagem sobre os outros métodos, por serem mais uteis na maioria das vezes, já que são capazes de lidar melhor com maiores quantidades de dados do que os outros.

O acesso sequencial indexado é o “pior” deles por não ser otimizado para grandes volumes de informações, mesmo que seja uma ótima maneira de separar dados simples de forma fácil.

Já a árvore binaria, a mais conhecida de todas, é utilizada em situações onde as outras árvores são de mais e quando o acesso sequencial indexado é insuficiente.