**FUNDAÇÃO DE ASSISTÊNCIA E EDUCAÇÃO - FAESA**

**FACULDADES INTEGRADAS ESPÍRITO-SANTENSES**

**CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO**

**KEOMA KLAVER KLEIN**

**WILIAM TRANCOSO DAL CIN JUNIOR**

**MÉTODOS DE PESQUISA E ORDENAÇÃO**

**VITÓRIA**

**2016**

**KEOMA KLAVER KLEIN**

**WILIAM TRANCOSO DAL CIN JUNIOR**

**MÉTODOS DE PESQUISA E ORDENAÇÃO**

Trabalho de Pesquisa e Ordenação do Curso de Graduação em Ciências da Computação apresentado às Faculdades Integradas Espírito-santenses, sob orientação da professora Cinthia Caliari.

**VITÓRIA**

**2016**

**SUMÁRIO**

**1. INTRODUÇÃO**

**2. DESENVOLVIMENTO**

[**2.1. 500 registros:**](#_3znysh7)

[**2.2. 1000 registros:**](#_2et92p0)

[**2.3. 5000 registros:**](#_tyjcwt)

[**2.4. Problemas Encontrados:**](#_3dy6vkm)

**3. REFERÊNCIAS BIBLIOGRÁFICAS**

1. **INTRODUÇÃO**

Esse trabalho é um estudo para medirmos a diferença entre os métodos de pesquisa e ordenação: ShellSort, QuickSort, HeapSort, Quicksort com Inserção Direta, ABB não balanceada, AVL, Busca Binária e Hashing com vetor encadeado.

Método quicksort: Um método de ordenação que adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar o vetor colocando as chaves menores para a esquerda e as maiores para a direita e depois e depois rearranjar as lista menores recursivamente até que todo o vetor esteja ordenado.

Inserção Direta: Um método de ordenação mais simples que o quicksort, mas nem por isso mais eficiente. Inicialmente, são ordenados os dois primeiros elementos do vetor, logo em seguida o 3º elemento é ordenado com os anteriores e assim o processo segue até que todo o vetor esteja ordenado.

Shellsort é um método de pesquisa eficaz, de fácil implementação, não usa recursividade e gera pouco código na implementação, apesar de não se mensurar o motivo da eficiência, o shellsort se mostra uma ótima opção para arquivos de +/- 5000 registros  
  
O algoritmo heapsort é um algoritmo de ordenação generalista, e faz parte da família de algoritmos de ordenação por seleção. Tem um desempenho em tempo de execução muito bom em conjuntos ordenados aleatoriamente, tem um uso de memória bem comportado e o seu desempenho em pior cenário é praticamente igual ao desempenho em cenário médio

Pesquisa Binária: Um método de pesquisa em vetor. Ele consiste em dividir o vetor ao meio e comparar a chave do elemento naquela posição com o elemento da busca, se for o mesmo, elemento encontrado, senão faz a pesquisa novamente com os elementos da esquerda ou da direita.  
  
Árvore AVL e ABB trabalham com um princípio parecido com a da busca binária, nessa estrutura elementos menores do que a raiz são adicionados a esquerda e maiores a direita.

**DESENVOLVIMENTO**

Para esse trabalho usamos arquivos de textos com registros variados, do texto1 para o texto5 em ordem crescente, cada registro virou uma combinação da palavra e do parágrafo que esta estava localizada. registros repetidos(mesma palavra para o mesmo parágrafo) foram ignorados.

Para os testes foi usado um computador com a seguinte configuração: Processador: core i7 1.80Ghz, 16Gb de memória, Windows 10 64bits, Eclipse Oxygen.

Abaixo temos um quadro comparativo com todos os métodos usados para cada arquivo, o tempo dado em milissegundos extraído da seguinte rotina: Ler arquivo > ordenação > salvar arquivo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Comparação do tempo de execução dos algoritmos em milissegundos** | | | | | |
| **Ordenação** | | | | | |
|  | **Texto1** | **Texto2** | **Texto3** | **Texto4** | **Texto5** |
| **Shellsort** | 18 | 18 | 53 | 62 | 63 |
| **Quicksort** | 1 | 3 | 11 | 23 | 42 |
| **Heapsort** | 1 | 2 | 11 | 29 | 52 |
| **Quicksort com inserção** | 1 | 3 | 12 | 19 | 41 |
| **Inserção** | | | | | |
| **ABB** | 1 | 2 | 7 | 15 | 40 |
| **AVL** | 1 | 2 | 8 | 19 | 31 |
| **Busca** | | | | | |
| **Busca Binária** | x | x | x | 29 | x |
| **ABB** | x | x | x | 139 | x |
| **AVL** | x | x | x | 125 | x |
| **Hashing Encadeado** | x | x | x | 113 | x |

1. **Texto1.txt:**

Com relação ao menor arquivo, os métodos de ordenação tiveram um desempenho parecido, exceto pelo shellsort que ficou acima dos demais;  
Inserção de poucos registros nas duas estruturas de árvore tivemos resultados parecidos

1. **Texto3.txt:**

Conforme a quantidade de registro foi aumentando, o tempo de ordenação também, o método shellsort teve uma curva de aumento menor, mas permaneceu menos eficaz que os demais, o Heapsort teve melhor desempenho com o arquivo médio.  
A Inserção na ABB um pouco mais rápida comparada a inserção na ABB

1. **Texto5.txt:**

A curva no aumento do tempo do shellsort aumentou pouco; O quicksort se mostrou mais eficaz, em especial o quicksort com inserção direta.

1. **Métodos de busca**

Para melhorar o desempenho do hashing, adotamos a estratégia de registro com palavras iguais serem armazenadas em uma lista encadeada tornando assim a pesquisa no hashing encadeado o método de pesquisa mais rápido, seguido da pesquisa na AVL e depois ABB

**3. CONCLUSÃO**

Para ordenação com poucos registros, o desempenhos de todos os métodos se parecem e conforme aumentamos a quantidade de registros o quicksort com inserção direta vai se provando mais eficaz. Nos testes de inserção de itens nas estruturas de árvores, mesmo a ABB ir se degenerando, ela ainda teve um desempenho melhor do que na AVL, talvez aumentando a quantidade de registro poderíamos visualizar os efeitos da degeneração da ABB. Em contrapartida ao melhor desempenho da ABB na inserção, a AVL teve menor tempo de busca nos registros, sendo superada em eficiência apenas pela busca no Hashing encadeado.