**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LAED1 - Trabalho Pratico 2**

**Aluno: Igor Miranda Oliveira**

**Matricula: 201312040080**

**Data: 10/11/2017**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

Neste trabalho, foram implementados algoritmos não-eficientes (BubbleSort, InsertionSort e SelectionSort) e eficientes (QuickSort, HeapSort, ShellSort e MergeSort) para ordenar sequências de elementos. A linguagem C foi utilizada em todos os algoritmos, o sistema operacional utilizado foi Windows, em um dispositivo com um processador Intel i5-7200u.

Todos os elementos das sequências correspondem a valores inteiros e diferentes sequências iniciais: ordenadas (crescentemente), inversamente ordenadas (decrescentemente), quase ordenadas (metade do vetor foi ordenada) e aleatórias (para cada execução de um algoritmo). Também foram analisadas sequências sem repetições. Os resultados (número de itens e tempo de execução) foram escritos em arquivos csv (Comma separated values) e exportados para gráficos, que podem ser visualizados nos arquivos em pdf anexados.

A biblioteca geraVetor.h foi criada para gerar diferentes tipos de vetores, que foram utilizados pelos demais algoritmos. As implementações do livro texto da disciplina ordenaram os vetores criados.

**Análise de Complexidade**

|  |  |
| --- | --- |
| **Algoritmo** | **Complexidade** |
| Bubblesort |  |
| Inserção |  |
| Seleção |  |
| Heapsort |  |
| Mergesort |  |
| Quicksort |  |
| Shellsort |  |

**Análise dos algoritmos não-eficientes (BubbleSort, InsertionSort e SelectionSort)**

Em todos testes, exceto o de vetores sem elementos repetidos, o InsertionSort foi mais constante, sendo o mais eficiente de todos, seguido pelo SelectionSort e por fim o BubbleSort. O InsertionSort apresentou uma queda no tempo de execução para sequências aleatórias com mais de 600 elementos. O BubbleSort apresentou um comportamento crescente.

Em testes com vetores sem elementos repetidos, o mais eficiente foi o SelectionSort, seguido pelo InsertionSort e por fim o BubbleSort. O SelectionSort manteve-se abaixo de 100ms exceto para n próximo de 1700. O BubbleSort apresentou um comportamento crescente.

**Análise dos algoritmos eficientes (QuickSort, HeapSort, ShellSort e MergeSort)**

Nos testes com vetores aleatórios e sem repetição os algoritmos com melhor resultado foram o MergeSort seguido pelo QuickSort e o HeapSort, que apresentaram comportamentos similares e por fim o ShellSort, que apresentou tempos de execução dispersos.

Em testes com vetores ordenados, quase ordenados e inversamente ordenados os algoritmos tiveram desempenho similar, entretanto o que performou melhor foi o QuickSort e por fim o HeapSort.

**Referências**

* Ziviani, N. Projeto de algoritmos: com implementações em Java e C++. 3 ed. Editora Cengage Learning, 2007.
* Loureiro, A. A. F. Projeto e Análise de Algoritmos: Análise de Complexidade. Notas de aula, 2010.
* <https://stackoverflow.com/questions/5064379/generating-unique-random-numbers-in-c>
* <http://www2.dcc.ufmg.br/livros/algoritmos/implementacoes.php>
* <https://linustechtips.com/main/topic/856289-analysing-different-sorting-algorithms/>