Algoritmos de Ordenação

Millena Suiani Costa
Disciplina – Algoritmos II (CIX56)
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
millena.costa@ufpr.br

***Resumo*—A ordenação de dados é imprescindível para diversas aplicações, e para executá-la da melhor forma possível, com base nos objetivos de cada operação, existem vários algoritmos que consistem nas mais diversas e variadas formas de fazê-lo. O presente trabalho visou, e teve como resultado, a implementação recursiva de dois tipos de busca – Iterativa (ou Ingênua) e Binária – e cinco tipos de algoritmos de ordenação – Insertion Sort, Selection Sort, Merge Sort, Quick Sort e Heap Sort. No presente relatório, foi discorrido mais à respeito de testes realizados com cada vetor, quantidades de comparações feitas entre seus elementos e tempo levado para as suas execuções, além dos respectivos resultados de cada tópico.**

***Index Terms*—Algoritmos. Ordenação. Vetores. Recursão.**

I. Introdução

O presente trabalho consiste em arquivos de código que testam o funcionamento de diferentes algoritmos recursivos de busca e ordenação em vetores de diferentes tamanhos e configurações. Este foi desenvolvido e testado em um sistema operacional Linux de distribuição Ubuntu em uma máquina Dell Inspiron5575, de forma que há possibilidade de margem de erro se testado em outras máquinas.

II. Algoritmos de Busca

Primeiramente, ao se tratar das buscas, a busca binária se mostrou mais rápida e eficiente que a busca sequencial, conforme mostra o gráfico de desempenho das duas:

III. Insertion Sort

O Insertion Sort, se mostrou, de acordo com os testes, bom para ordenar vetores pequenos e na baixa utilização de memória (visto que não necessita da criação de vetores auxiliares). Já no caso de vetores grandes, especialmente os que são inversamente ordenados, o algoritmo levou muito mais tempo e um maior número de comparações para ordená-los corretamente.

IV. Selection Sort

O Selection Sort apresentou nos testes resultados muito parecidos com o do Insertion Sort. Esse, por sua vez, e também se mostrou muito eficiente na ordenação de pequenos vetores e na falta de necessidade de criação de vetores auxiliares. Esse algoritmo, devido ao seu crescimento quadrático, não é recomendado para aplicações com um número muito grande de dados, pois levará muito tempo para a execução.

V. Merge Sort

O Merge Sort demonstrou um ótimo desempenho na ordenação de grandes vetores, demandando de poucas comparações e um tempo pequeno para a sua execução. Entretanto, o Merge não seria útil para situações onde é disponibilizada pouca memória de máquina, visto que esse faz a criação de vetores auxiliares.

VI. Quick Sort

O Quick Sort teve um desempenho razoável na maioria das execuções, mesmo as com vetores maiores. Contudo, como era de se esperar, apresentou certas dificuldades em determinados tipos de vetor – como, por exemplo, os metade ordenados – e devido ao seu pior caso ser similar ao caso médio de outros algoritmos, nesses casos não oferece resultados muito bons.

VII. Heap Sort

O Heap Sort, devido ao seu comportamento n log n em todos os casos, apresentou bom desempenho e um tempo pequeno na ordenação da grande maioria dos vetores, exceto nos vetores menores, nos quais se mostrou muito complexo e com desempenho reduzido.

VIII. Conclusão

Concluiu-se à partir da realização do presente trabalho que, os mais diversos algoritmos, tanto de busca quanto de ordenação, possuem diferentes finalidades. É possível indicar qual o melhor para uso com base nos resultados esperados para tal. O trabalho foi realizado conforme os critérios pré-estabelecidos.