Atividade Prática 07

Aluno: Raphael Henrique Braga Leivas

Matrícula: 2020028101

1 OBJETIVO

O objetivo da prática é comparar o desempenho de execução de um código de ordenação usando ShellSort com um algoritmo eficiente, no caso, o HeapSort.

2 METODOLOGIA

2.1 DESCRIÇÃO GERAL

Será usado como regra de decaimento do h passando por todos os números pares desde o valor inicial 40 até 1. Isso é feito pois os tamanhos dos vetores analisados também serão pares, e assim temos tamanhos de vetores compatíveis com os tamanhos dos h escolhidos.

Serão usados 10 tamanhos de vetores N:

$$N = [10,\ 50,\ 100,\ 500,\ 1000,\ 5000,\ 10000,\ 50000,\ 100000,\ 200000]$$

Os vetores serão preenchidos aleatoriamente com números inteiros a serem ordenados.

O algoritmo eficiente de ordenação de referência será o HeapSort, implementado como visto durante as aulas da disciplina e com base em referências disponíveis na internet.

A comparação de eficiência será feita através do tempo gasto entre a ordenação do mesmo vetor com o ShellSort e com o HeapSort. Usando a biblioteca CustomTime.h desenvolvida de autoria própria na aula prática AP02, é possível extrair o tempo gasto na ordenação de um vetor usando a diferença entre os timestamp UNIX em microssegundos antes chamada da função (*before*) e depois da ordenação (*after*).

2.2 DESCRIÇÃO DO CÓDIGO

Os códigos desenvolvidos podem ser vistos na pasta TP, dentro da pasta compactada enviada no moodle em que esse arquivo PDF se encontra. Consiste no arquivo principal que executa os testes (AP07.cpp) e a biblioteca CustomTime.h de autoria própria.

A função shellSort foi extraída dos slides da AP 07 disponibilizado no moodle.

A função heapSort foi adaptada de um código disponível na internet. É importante destacar que algoritmos de ordenação heapSort são amplamente utilizados e possuem

várias referências e tutoriais disponíveis na web para consulta, não sendo necessário programá-los do zero por uma questão de tempo.

A função getUnixTimestamp extrai o número de milissegundos, microsegundos ou nanosegundos (dependendo de seu argumento de entrada) desde 1 de janeira de 1970 (época UNIX) e possui implementação disponível na biblioteca CustomTime.h na pasta include. O tempo gasto por uma função é dado pela diferença entre o timestamp Unix antes e depois de sua execução, e essa metodologia é a utilizada para extrair o tempo gasto na execução da função.

A função fillArrayWithRandom preenche um vetor com números inteiros aleatórios, usando o algoritmo exibido nos slides da AP 07 disponível no moodle.

O código na função main essencialmente executa o procedimento descrito na seção 2.1, chamados as funções descritas acima.

3 ANÁLISE DOS RESULTADOS

A Figura 3.1 exibe o tempo em microssegundos gasto para ordenar vetores aleatórios de tamanho N para o ShellSort e o HeapSort.

Figura 3.1. Tempos de execução gastos para a ordenação de vetores aleatórios de tamanho N.

```
N = 10 : ShellSort = 0 us -- HeapSort = 2 us
N = 50 : ShellSort = 4 us -- HeapSort = 13 us
N = 100 : ShellSort = 11 us -- HeapSort = 29 us
N = 500 : ShellSort = 87 us -- HeapSort = 208 us
N = 1000 : ShellSort = 337 us -- HeapSort = 473 us
N = 5000 : ShellSort = 2232 us -- HeapSort = 2474 us
N = 10000 : ShellSort = 6344 us -- HeapSort = 6851 us
N = 50000 : ShellSort = 136827 us -- HeapSort = 33345 us
N = 100000 : ShellSort = 548727 us -- HeapSort = 69361 us
N = 200000 : ShellSort = 1958594 us -- HeapSort = 136546 us
```

Fonte: elaboração própria.

Analisando os dados da Figura 3.1, observamos que o ShellSort é capaz de ordenar vetores pequenos com melhor desempenho que o HeapSort. Contudo, para valores grandes de tamanho do vetor de entrada $N > 10000$, o algoritmo HeapSort fica significativamente mais eficiente que o ShellSort, chegando a ser quase 20 vezes mais rápido para $N = 200000$.

Esse comportamento pode ser explicado pela complexidade do HeapSort, que é $O(nlog(n))$ e portanto cresce praticamente linearmente com o tamanho do vetor de entrada (aqui, estamos identificando que a função $log(n)$ cresce bastante devagar), o que não acontece com o ShellSort.

4 CONCLUSÃO

Tendo em vista o objetivo da prática, foi possível comparar o desempenho de dois algoritmos de ordenação que, apesar de conseguirem cumprir a mesma função de ordenar dois vetores de inteiros, possuem eficiências diferentes dependendo do tamanho do vetor de entrada

5 REFERÊNCIAS

GEEKS FOR GEEKS. C++ program for HeapSort. Disponível em C++ Program for Heap Sort - GeeksforGeeks. Acesso em 23 de mai. de 2023.