Tarefa Prática 7

1. Planejar uma bateria de testes com diferentes valores de tamanho para os vetores a serem gerados pelo programa, assim como uma forma de analisar a performance do método de maneira significativa para o conjunto de valores de h's escolhidos.

2. Implementar o heapsort (ordenação por heap) e o shellsort com sua estratégia de escolha e atualização dos h.

3. Implementar um gerador de vetores aleatórios

4. Realizar os testes descritos no ponto 1.

5. Apresentar um relatório em PDF com os resultados obtidos.

Resposta:

1. Bateria de testes:

    a) tamanho vetor: 100 // 50 vetores

        h1:75

        h2:37

        h3:18

        h4:9

        h4:1

    b) tamanho vetor: 250 // 125 vetores

        h1:188

        h2:94

        h3:47

        h4:23

        h5:11

        h6:5

        h7:1

    c) tamanho vetor: 500 // 250 vetores

        h1:375

        h2:187

        h3:93

        h4:46

h5:23

h6:11

h7:5

h8:1

d) tamanho vetor: 1000 // 500 vetores

h1:750

h2:370

h3:180

h4:90

h5:45

h6:22

h7:11

h7:5

h8:1

A regra de h escolhida é separar em 4 h's sendo o primeiro h n-(n/4) ,n/4 é aproximado para o inteiro maior (2,3 arrendoda para 3). Os próximos h's são (h_anterior/2) arrendondando para baixo e depois quando o h for menor que 10 o próximo é 1. A lógica que eu pensei foi um vetor de n dividido em 4 partes são tamanhos que separam o vetor em uma quantia razoável, se você divide por 2 ainda é um tamanho muito próximo do original do n. por 8 é uma fragmentação muito grande portanto peguei a potência de dois intermediária. Para os próximos valores escolhi dividir eles por 2, pois pensei que a margem dentre as ordenações precisava ser menor que a divisão do vetor inicial para o caso h=1 trabalhar menos quando chegar seu momento. Por fim, acredito que h menores que 10 seriam muito custosos, então só executarei um desses para aliviar o h=1, mas não quero exagerar na quantidade desses casos.

Para validar os testes, irei testar n/2 vetores para mostrar que de fato a regra se aplica para qualquer tamanho e para qualquer sequência com o tamanho indicado.Em relação ao tempo de execução em relação ao tamnho usarie 4 tamanhos distintos. Utilizarei o tempo de execução de cada um desses casos e compararei para tirar as conclusões.

Seguindo para os resultados.

Devido ao tempo muito pequeno fiz a bateria de testes de duas formas, a primeira o tempo de execução para cada um dos vetores tanto no shellsort quanto no heapsort e depois a soma dos tempos de todos os vetores. Cada

análise tem sua vantagem. A individual é possível garantir que o tempo analisado do heapsort e do shellsort se referem ao mesmo vetor, já o da soma não.

Tempo individual de cada vetor:

Tamanho 100: O tempo de execução do shellsort era, em média, 6 vezes maior que o tempo do heapsort. Muita oscilação.

Tamanho 250: O tempo de execução do shellsort era, em média, 6 vezes maior que o tempo do heapsort. Muita oscilação.

Tamanho 500: O tempo de execução do shellsort era, em média, 7 vezes maior que o tempo do heapsort. Muita oscilação.

Tamanho 1000: O tempo de execução do shellsort era, em média, 7 vezes maior que o tempoo do heapsort. Curiosamente, o com menos oscilação.

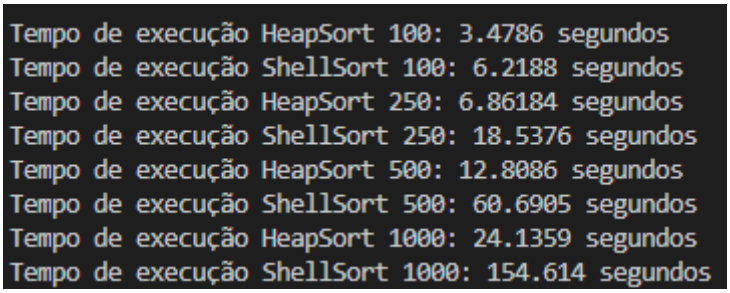
Tempo somados de todos os vetores:

Tamanho 100: O tempo de execução do shellsort foi 1,78 vezes maior que o tempo do heapsort.

Tamanho 250: O tempo de execução do shellsort foi 2,70 vezes maior que o tempo do heapsort.

Tamanho 500: O tempo de execução do shellsort foi 4,75 vezes maior que o tempo do heapsort.

Tamanho 1000: O tempo de execução do shellsort foi 6,41 vezes maior que o tempo do heapsort.



Os resultados indicam duas coisas. A primeira sendo que os valores individuais são muito pequenos e oscilam muito de caso a caso não sendo bons para determinar qual método é melhor e quão melhor esse método seria. A segunda afirmação é baseada no resultado mais confiável, o da soma, o qual foi repetido com 1.000.000 de vetores para cada tamanho propostos e mantiveram a mesma proporção, o tempo de execução do shellsort cresce mais rápido que o do heapsort de acordo com a entrada.

O makefile irá compilar individualmente e na última execução irá dar o tempo individual e o tempo total.