Aula Prática 7

Lucas Albano Olive Cruz | 2022036209

Roteiro:

Irei testar vetores com elementos pseudo aleatórios de tamanho 20, 50, 100, 500 e 1000.

Para cada tamanho serão testados 10 vetores com elementos diferentes.

Cada vetor será ordenado por Shellsort e Heapsort.

Irei usar duas sequências de h's. A sequência de Knuth e a sequência das potências de dois.

Ao fim serão calculadas as métricas: média de Tempo de CPU gasto, média do número de instruções, média de leituras na memória, média de escritas na memória.

Também serão extraídas as mesmas métricas para vetores de tamanho 100 já ordenados e em formato decrescente.

Resultados:

ShellSort com Sequência de Knuth (Média de 10 tentativas):

Tamanho	20	50	100	500	1000
Tempo de CPU (nanosegundos)	940	2500	5650	40600	95150
Instruções	3105,8	10390	25984	191165	436796
Leitura	1616	5483	13813	102214	234009
Escrita	239,2	746	1775	12422	27830

ShellSort com Potências de dois (Média de 10 tentativas):

Tamanho	20	50	100	500	1000
Tempo de CPU (nanosegundos)	1100	3200	7300	55500	136300
Instruções	4274	13598	31367	313301	695069
Leitura	2240	7154	16509	168047	372821
Escrita	331	1032	2387	19679	43667

HeapSort (Média de 10 tentativas):

Tamanho	20	50	100	500	1000
Tempo de CPU (nanosegundos)	1720	4750	10800	65700	152300
Instruções	5766,6	19978	48680	346478	787496
Leitura	2414,3	8387,5	20466	145952	331885
Escrita	857,5	2902,5	6988	49116	111293

Vetor de tamanho 100 ordenado:

Método	Tempo de CPU (ns)	Instruções	Leitura	Escrita
Shellsort (Knuth)	1600	14716	7533	1344
Shellsort (Ptc de 2)	2500	22358	11504	2039
HeapSort	9600	53356	22445	7789

Vetor de tamanho 100 em ordem decrescente:

Método	Tempo de CPU (ns)	Instruções	Leitura	Escrita
Shellsort (Knuth)	2600	20278	10643	1574
Shellsort (Ptc de 2)	3300	27335	14269	2243
HeapSort	8000	43530	18289	6191

Conclusões:

Conforme observado o algoritmo de ShellSort utilizando a sequência de Knuth gasta cerca da metade do tempo de CPU de um algoritmo de HeapSort até vetores de tamanho 100. Para vetores maiores a diferença tende a diminuir. No entanto, as escritas em memória aparentam ser uma ordem de grandeza menores para o ShellSort. As demais métricas aparentam terem um crescimento semelhante embora o ShellSort ainda gaste menos recursos de leitura e instruções.

O ShellSort com potências de dois se mantêm entre o ShellSort com a sequência de Knuth e o HeapSort. A característica de ter os passos de escrita consideravelmente menores ao HeapSort se perpetua.

Para os melhores e piores casos, ambos métodos de ShellSort desempenham melhor que o HeapSort. No entanto, é interessante notar que o HeapSort desempenha melhor para vetores em ordem decrescente do que para vetores já ordenados.

De fato, embora não se saiba exatamente a complexidade do ShellSort ele se mantêm entre $O(n)$ e $O(n^2)$. Existem conjecturas que o número de comparações para a sequência de Knuth é $O(n^{1,25})$ e para a sequência de potências de dois é $O(n^{1,5})$. Enquanto isso, o HeapSort possui complexidade $O(n \cdot log(n))$.

Observações sobre o método de análise:

Para analisar o tempo de CPU gasto foi usado a biblioteca chrono e para as demais métricas a ferramenta cachegrind do Valgrind.

Os testes foram feitos em várias execuções do programa, não sendo possível replicar somente com o comando make run. Para alterar o tamanho do vetor basta alterar o atributo size. Para alterar a sequência de h 's basta alterar os passos “calcula h” e “atualiza h”. O método de geração de vetores também pode ser alterado para gerar um vetor ordenado ou em forma decrescente.