Aula Prática 7 – Estrutura de Dados

Iago da Silva Rodrigues Alves – 2022035881

Contextualização

A variação escolhida para o ShellSort utiliza a sequência de números de Fibonacci para definir os intervalos de comparação e troca dos elementos no vetor. Assim, a variavel h é inicializada com Fibonacci referente ao tamanho do vetor passado como parâmetro, e é atualizada como o valor de Fibonacci de h – 1;

Testes de Execução

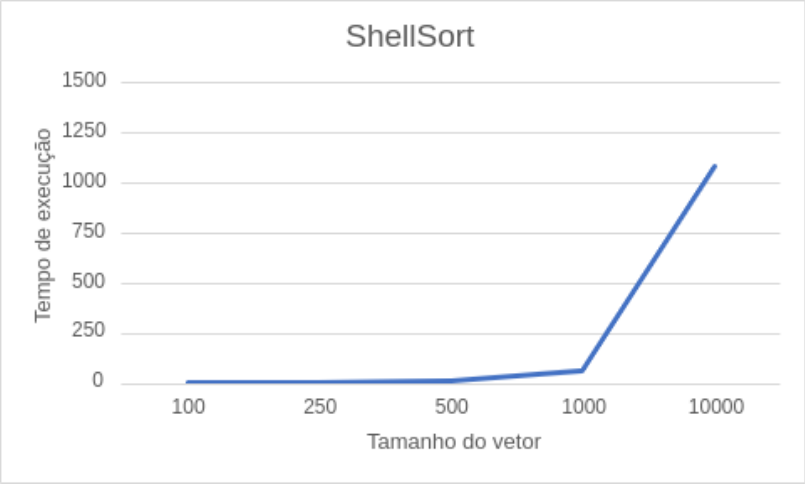
Para analisar a performance do algoritmo de maneira significativa para esse conjunto de h escolhido, testei a ordenação ShellSort para vetores de tamanho 100, 250, 500, 1000 e 10000 elementos. Vale ressaltar que foram feitos 100 testes para cada tamanho de vetor e tirada a média do tempo de execução.

O mesmo teste foi feito para o algoritmo de ordenação tipo HeapSort.

Resultado ordenação ShellSort

Vale ressaltar que o tempo é calculado em microssegundos e o vetor é um vetor desordenado gerado aleatoriamente.

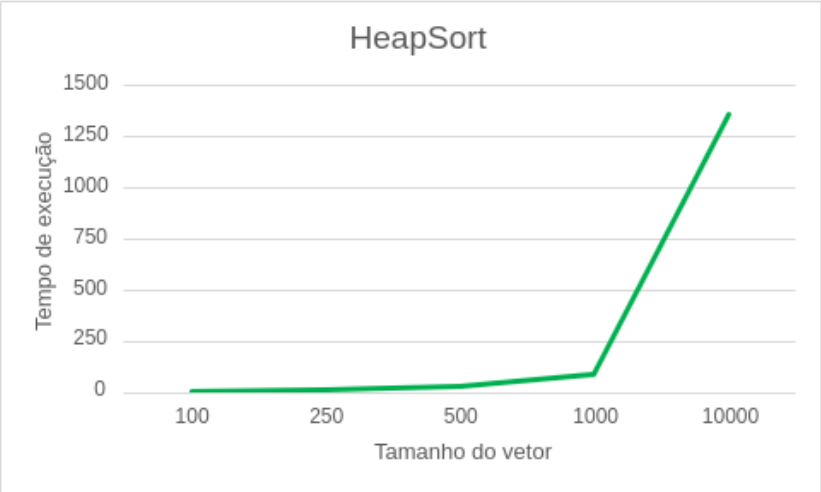
- Tamanho do vetor: **100** Tempo de execução: **2.93** microssegundos
- Tamanho do vetor: **250** Tempo de execução: **7.33** microssegundos
- Tamanho do vetor: **500** Tempo de execução: **17.35** microssegundos
- Tamanho do vetor: **1000** Tempo de execução: **68.79** microssegundos
- Tamanho do vetor: **10000** Tempo de execução: **1086.45** microssegundos



Podemos observar que à medida que o tamanho do vetor desordenado aumenta, o tempo de execução do algoritmo Shellsort também aumenta significativamente. No entanto, também é importante notar que a sequência de Fibonacci usada para determinar os intervalos de ordenação no Shellsort pode não ser a melhor escolha para obter o desempenho ideal.

Resultado ordenação HeapSort

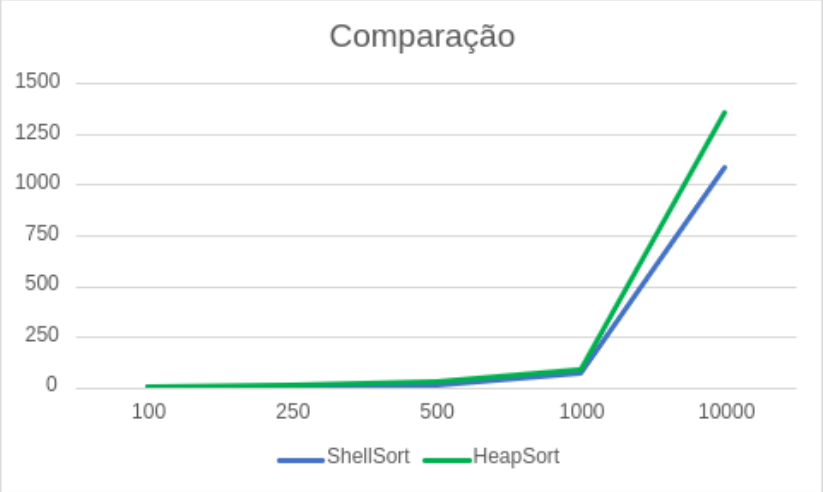
- Tamanho do vetor: **100** Tempo de execução: **5.04** microssegundos
- Tamanho do vetor: **250** Tempo de execução: **13.63** microssegundos
- Tamanho do vetor: **500** Tempo de execução: **30.59** microssegundos
- Tamanho do vetor: **1000** Tempo de execução: **89.74** microssegundos
- Tamanho do vetor: **10000** Tempo de execução: **1363.57** microssegundos



Podemos observar que, assim como no caso do Shellsort, à medida que o tamanho do vetor desordenado aumenta, o tempo de execução do Heapsort também aumenta. No entanto, os tempos de execução do Heapsort parecem ser um pouco mais altos em comparação com os tempos do Shellsort para os mesmos tamanhos de vetor.

Lembrando que ele possui uma complexidade de tempo médio de O(nlogn) e é geralmente mais eficiente do que o Shellsort em grandes conjuntos de dados. No entanto, em casos de conjuntos de dados menores, o Shellsort pode ter um desempenho semelhante ou até melhor do que o Heapsort.

Comparação e conclusões



Comparando os resultados obtidos para o algoritmo Shellsort com a sequência de Fibonacci e o algoritmo Heapsort, podemos tirar algumas conclusões:

1. Tempo de execução: Em geral, o Shellsort com a sequência de Fibonacci tem tempos de execução mais curtos do que o Heapsort para os mesmos tamanhos de vetores desordenados. Isso sugere que o Shellsort pode ser mais eficiente em termos de tempo de execução em comparação com o Heapsort para esses casos específicos.
2. Crescimento do tempo de execução: Ambos os algoritmos mostram um crescimento no tempo de execução à medida que o tamanho do vetor desordenado aumenta. No entanto, o crescimento é mais pronunciado para o Heapsort, indicando que ele pode ser mais sensível ao aumento do tamanho do vetor em comparação com o Shellsort com a sequência de Fibonacci.
3. Sequência de intervalos: O uso da sequência de Fibonacci no Shellsort pode não ser a melhor escolha em termos de desempenho. Outras sequências de intervalos podem ser exploradas para melhorar ainda mais o desempenho do Shellsort.

Em resumo, os resultados indicam que o Shellsort com a sequência de Fibonacci tem tempos de execução mais curtos em comparação com o Heapsort para os tamanhos de vetores desordenados apresentados.