Um Limite Superior para a Complexidade do ShellSort

Raquel M. Souza[1]*, **Fabiano S. Oliveira**[1]†, **Paulo E. D. Pinto**[1]†

[1]Universidade do Estado do Rio de Janeiro, Brasil

raquelmarcolino25@gmail.com, {fabiano.oliveira,pauloedp}@ime.uerj.br

Abstract. *The worst-case time complexity of the ShellSort algorithm is known only for some specific sequences (a sequence is a parameter of the algorithm). Relating the algorithm to the Frobenius number concept, we present an algorithm for determining the maximum number of comparisons for any sequence and array to be ordered. We apply this method together with the empirical determination of complexity to analyze several sequences whose worst case complexity are known. We show that the empirical approach succeeded in determining the same complexities which are analytically known and presented its results for sequences with unknown worst-case time complexity.*

1. Introdução

O algoritmo ShellSort é um algoritmo de ordenação por comparação criado por [Shell 1959] com o objetivo de ter complexidade de pior caso de tempo ótima de $\Theta(n \log n)$, como aquela do MergeSort, e de fazer uso de espaço auxiliar constante. O algoritmo utiliza uma sequência de inteiros, chamados de *passos*, que influencia na complexidade do algoritmo. A complexidade de pior caso para a sequência de Shell não atingiu tal objetivo ([Frank 1960] mostraram posteriormente que o pior caso resultante da sequência de Shell é de $\Theta(n^2)$), mas abriu caminho para que diversos trabalhos estudassem outras sequências de passos, que resultaram em complexidades de pior caso de tempo tais como $\Theta(n^{3/2})$, $\Theta(n \log^2 n)$ e $O(n^{4/3})$ [Frank 1960, Pratt 1972, Sedgewick 1986].

Embora a relação entre o algoritmo ShellSort e o número de Frobenius já tenha sido abordada [Incerpi 1983], este trabalho apresenta uma nova relação. Tal relação conduz a um algoritmo para determinar um limite superior para o número de comparações numa execução do ShellSort dados uma sequência de passos e um vetor. Com tal limite, é realizado um estudo empírico a fim de encontrar complexidades de pior caso para sequências clássicas, algumas cuja complexidade de pior caso são desconhecidas.

2. ShellSort

ShellSort é um algoritmo de ordenação por comparação [Shell 1959] de um vetor V de n elementos que executa o algoritmo InsertionSort em diferentes subsequências de V. Tais subsequências são definidas por uma *sequência de passos* $p_1, p_2, \ldots, p_k$ com $p_1 = 1$, $p_i < p_{i+1}$ para todo $1 \leq i < k$ e $p_k < n$. O ShellSort é descrito pelo Algoritmo 1, no qual InsertionSort(V, S) representa a ordenação por inserção da subsequência de V definida pela sequência de índices S. Diversos trabalhos estudaram a complexidade de diferentes sequências, sumarizadas na Tabela 1.

O menor limite superior conhecido que independa de sequência é dado a seguir.

*Projeto financiado por CAPES.

†Projeto parcialmente financiado por FAPERJ.

Entrada: $V[1..n]$ de inteiros, sequência de passos $p_1, p_2, \ldots, p_k$
Resultado: V ordenado
para $j \leftarrow k$ **até** 1 **incremento** -1 **faça**
  para $i \leftarrow 1$ **até** p_j **faça**
    InsertionSort(V, $S_{i,j}$), com $S_{i,j} = i, i + p_j, i + 2p_j, \ldots, i + \left\lfloor \frac{n-i}{p_j} \right\rfloor p_j$

Algoritmo 1: Algoritmo do ShellSort.

Tabela 1: Complexidade analítica para algumas sequências de passos.

[Autor Ano]	**Passos** $\{p_1, \ldots, p_k\}$	**Pior caso**
[Shell 1959]	$\{\lfloor n/2^i \rfloor : 0 < i \leq \lfloor \log_2 n \rfloor\}$	$\Theta(n^2)$
[Frank 1960]	$\{2 \cdot \lfloor n/2^i \rfloor + 1 : 0 < i \leq \lfloor \log_2 n \rfloor\}$	$\Theta(n^{3/2})$
[Hibbard 1963]	$\{2^i - 1 : 0 < i \leq \lfloor \log_2 n \rfloor\}$	$\Theta(n^{3/2})$
[Papernov 1965]	$\{1\} \cup \{2^{i-1} + 1 : 1 < i \leq \lfloor \log_2 n \rfloor\}$	$\Theta(n^{3/2})$
[Pratt 1972]	$\{q = 2^r 3^s : r, s \in \mathbb{N} \mid q < n\}$	$\Theta(n \log^2 n)$
[Knuth 1973]	$\{q = (3^i - 1)/2 : i \in \mathbb{N} \mid q \leq \lceil n/3 \rceil\}$	$\Theta(n^{3/2})$
[Sedgewick 1986]	$\{q = 4^i + 3 \cdot 2^{i-1} + 1 : 1 < i \in \mathbb{N} \mid q < n\}$	$O(n^{4/3})$
[Gonnet 1991]	$\{\lfloor (0,45454)^i \cdot n \rfloor : 0 < i \leq \lfloor \log_{2,2} n \rfloor\}$	em aberto
[Tokuda 1992]	$\{q = \lceil (9^i - 4^i)/(5 \cdot 4^{i-1}) \rceil : 1 < i \in \mathbb{N} \mid q < n\}$	em aberto

Teorema 1. *A complexidade de pior caso do algoritmo ShellSort é* $O(n^2 \log \log n)$ *para qualquer constante* c *e sequência com* $O(\log^c n)$ *passos.*

3. Relação com Número de Frobenius

Considere o conjunto $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{N}$, $|A| > 1$, tal que $a_1, a_2, \ldots, a_n > 1$ e MDC$(a_1, a_2, \ldots, a_n) = 1$. Seja $F(A) \subset \mathbb{N}$ tal que, para cada elemento $f \in F(A)$, não existem $k_1, k_2, \ldots, k_n \in \mathbb{N}$ de modo que $k_1 a_1 + k_2 a_2 + \ldots + k_n a_n = f$, isto é, f não pode ser representado como uma combinação linear dos elementos do conjunto A. O *número de Frobenius*, denotado por $g(A)$, é o maior número de F. O problema de determinar $g(A)$ possui solução algébrica para $|A| = 2$, a saber, $g(\{a_1, a_2\}) = a_1 a_2 - a_1 - a_2$, mas sua solução algébrica é um problema em aberto quando $n \geq 3$. Contudo, elaboramos um algoritmo pseudo-polinomial como uma variação do algoritmo da mochila que é eficiente na prática. Tal algoritmo determina o número de Frobenius dado um limite superior $t \in \mathbb{N}$ para $g(A)$. Mais especificamente, dados $A = \{a_1, a_2, \ldots, a_n\}$ e $t \in \mathbb{N}$, o algoritmo determina $g(A, t) = \max\{f \in F(A) : f \leq t\}$, que é igual a $g(A)$ quando $t \geq g(A)$. No contexto do ShellSort, este algoritmo pode ser estendido para determinar também $g_m(A, t) = \max\{f \in F(A) : f \leq t, f \bmod m = 0\}$. Durante o processamento do passo p_s, pode ser mostrado que o número máximo de comparações para cada elemento do vetor neste passo é de $g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)$. Tal demonstração se baseia em mostrar que não ocorrem comparações de elementos cuja diferença de posições seja acima deste valor, pois os elementos já estariam ordenados entre si no processamento de algum dos passos anteriores. Baseado nestas ideias, o Teorema 2 permite estabelecer um limite superior de comparações de um elemento arbitrário para qualquer sequência utilizada. Este resultado é utilizado na elaboração do Algoritmo 2.

Teorema 2. *Durante a ordenação do vetor* V *pelo ShellSort, o número de comparações realizadas na iteração relativa ao passo* p_s *para cada elemento de* V *é, no máximo,* $\lfloor g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)/p_s \rfloor + 1$.

Demonstração (Esboço). Mostra-se que o limite superior do intervalo percorrido por um elemento do vetor na ordenação com a iteração relativa ao passo p_s é $g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)$. Para percorrer tal intervalo são necessárias $\lfloor g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\}, n)/p_s \rfloor + 1$ comparações. □

4. Algoritmo de Limite Superior para o Número de Comparações

Com base no Teorema 2, é possível enunciar o Algoritmo 2, que determina o número máximo de comparações de qualquer sequência do ShellSort.

Entrada: Sequência de passos $p_1, p_2, \ldots, p_k$ e inteiro n
Resultado: Número máximo de comparações realizadas pelo ShellSort com n elementos e com a sequência de passos $p_1, p_2, \ldots, p_k$
ncomp $\leftarrow 0$
para $s \leftarrow k$ **até** 1 **incremento** -1 **faça**
  limite $\leftarrow \lfloor g_{p_s}(\{p_k, p_{k-1}, \ldots, p_{s+1}\})/p_s \rfloor + 1$
  para $i \leftarrow 1$ **até** n **faça**
    limite$_i \leftarrow \lfloor (n-i)/p_s \rfloor$
    ncomp $\leftarrow$ ncomp + $\min\{$limite, limite$_i\}$
retorna *ncomp*

Algoritmo 2: Determinação do número máximo de comparações do ShellSort.

O algoritmo determina tal limite com o suporte do Teorema 2, observando-se que o limite dado pelo teorema é comparado com o tamanho dos subvetores a serem ordenados por InsertionSort. Em alguns casos, o limite dado pelo teorema pode ser maior que o número de elementos, e contabiliza-se o menor entre os dois. Como resultado, o algoritmo computa um limite superior para a complexidade de pior caso, em número de comparações, dados uma sequência de passos e o número n de elementos do vetor a ser ordenado. Este algoritmo é utilizado com o método empírico de se escolher diversos valores de n e de se determinar o limite superior para tais valores pelo Algoritmo 2. O resultado é analisado então com a ferramenta EMA [Oliveira 2016] para estimar a complexidade assintótica correspondente a tal conjunto de pontos. Apesar de não haver uma garantia de que o pior caso assim obtido coincida com os resultados da literatura, este foi o observado nos experimentos (ver Seção 5).

5. Resultados e Conclusão

A Tabela 2 mostra que os limites superiores para as complexidades retornados pelo algoritmo são justos para as sequências cujas complexidades de pior caso são conhecidas. Além disso, para a sequência [Sedgewick 1986], para a qual a complexidade de pior caso exata é desconhecida, o limite encontrado pelo método proposto neste trabalho coincide com aquele analítico. Portanto, os limites conhecidos analíticos foram todos obtidos através do método empírico apresentado. Para as sequências de complexidade analítica desconhecidas, apresentamos os resultados desta abordagem. Finalmente, observamos que o melhor limite superior analítico para sequências gerais, dado pelo Teorema 1, não é justo para nenhuma sequência específica, enquanto que a abordagem empírica obteve limites justos para todas elas.

Tabela 2: Tabela de complexidades retornadas pelo EMA para cada sequência.

[Autor Ano]	**Literatura**	**Empírico**
[Shell 1959]	$\Theta(n^2)$	$O(n^2)$
[Frank 1960]	$\Theta(n^{1,5})$	$O(n^{1,5})$
[Hibbard 1963]	$\Theta(n^{1,5})$	$O(n^{1,5})$
[Papernov 1965]	$\Theta(n^{1,5})$	$O(n^{1,5})$
[Pratt 1972]	$\Theta(n \log^2 n)$	$O(n \log^2 n)$
[Knuth 1973]	$\Theta(n^{1,5})$	$O(n^{1,5})$
[Sedgewick 1986]	$O(n^{1,333...})$	$O(n^{1,329})$
[Gonnet 1991]	em aberto	$O(n^{1,145})$, $O(n \log^3 n)$
[Tokuda 1992]	em aberto	$O(n^{1,255})$, $O(n \log^{5,5} n)$

Referências

Frank, R. M. Lazarus, R. B. (1960). A high-speed sorting procedure. *Communications of the ACM*, 3(1):20–22.

Gonnet, G. H. Baeza-Yates, R. A. (1991). *Handbook of Algorithms and Data Structures: in Pascal and C*. Addison-Wesley.

Hibbard, T. N. (1963). An empirical study of minimal storage sorting. *Communications of the ACM*, 6(5):206–213.

Incerpi, J. Sedgewick, R. (1983). Improved upper bounds on shellsort. *Anais do 24º Annual Symposium on Foundations of Computer Science*, p. 48–55.

Knuth, D. E. (1973). *The Art of Computer Programming 3: Sorting and Searching*. Addison-Wesley.

Oliveira, F. S. (2016). EMA - WebPage. `http://fabianooliveira.ime.uerj.br/ema`. [Último acesso: 10 de Março de 2018].

Papernov, A. A. Stasevich, G. V. (1965). A method of information sorting in computer memories. *Problemy Peredachi Informatsii*, 1(3):81–98.

Pratt, V. R. (1972). Shellsort and sorting networks. Relatório técnico, Documento DTIC.

Sedgewick, R. (1986). A new upper bound for shellsort. *Journal of Algorithms*, 7(2):159–173.

Shell, D. L. (1959). A high-speed sorting procedure. *Communications of the ACM*, 2(7):30–32.

Tokuda, N. (1992). An improved shellsort. *Anais do 12º IFIP World Computer Congress on Algorithms, Software, Architecture-Information Processing*, 1:449–457.