UNISINOS

Estruturas Avançadas de Dados I
(Shellsort)

Prof. Gilberto Irajá Müller

Última atualização 17/10/2017

Introdução

- Proposto por Ronald Shell em 1959;
- Explora as seguintes características do método de **inserção direta**:
  - desempenho aceitável quando o número de chaves é pequeno;
  - desempenho aceitável quando as chaves já possuem uma ordenação parcial.
- É considerado uma **extensão do método de inserção**, pois se diferencia deste apenas no número de segmentos considerados;
- Realiza classificações parciais do array a cada iteração, favorecendo o desempenho dos passos seguintes.

Introdução (cont.)

- Artigo "A High-Speed Sorting Procedure" disponível em http://penguin.ewu.edu/cscd300/Topic/AdvSorting/p30-shell.pdf;

- Primeiro algoritmo a "quebrar" a barreira quadrática, mas alguns anos depois, provou-se ser subquadrático. Ex.: ($O^{3/2}$);

- A complexidade de tempo do algoritmo é algo desconhecido, embora existam várias definições;

- Shellsort tem um bom desempenho porque explora o que tem de melhor no Insertion Sort: boa ordenação quando o array está parcialmente ordenado.

Funcionamento

- No **primeiro** passo, o array **A[n]** é dividido em **h** segmentos (também chamado de **gap**), de tal forma que cada um possua **n / h** chaves;

- Os elementos de um segmento são identificados por:
  Segmento i: A[i], A[h + i], A[2h + i], A[3h + i], …, p/ i = 1, 2, …, h;

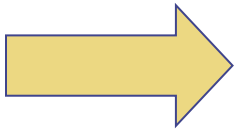
- Cada um dos segmentos é classificado por **inserção direta** separadamente;

- No **passo seguinte**, o **h** é decrementado (a metade do valor anterior, se este era potência inteira de 2 – baseado no artigo original);
- No último passo, **h = 1**

$h=2^2=4$

$h=2^1=2$

$h=2^0=1$



3 passos

Funcionamento (cont.)

- Testes empíricos sugerem que a melhor escolha para os valores de h é a sequência $(3^t-1)/2$, ..., 13, 4, 1; onde:
- $h_t = 3h_{t-1}+1$

$3*0+1 = 1$

$3*1+1 = 4$

$3*4+1 = 13$

$3*13+1 = 40$

- Sequência original do Shellsort: N/2 , N/4 , ..., 1;
- Sequência de Hibbard: 1, 3, 7, ..., $2^k - 1$;
- Sequência de Knuth: 1, 4, 13, ..., $(3^k - 1) / 2$;
- Sequência de Sedgewick: 1, 5, 19, 41, 109, ...
- Outras https://en.wikipedia.org/wiki/Shellsort#Gap_sequences

Implementação do Shellsort

```
public static <T extends Comparable<? super T>> void shellSort(T[] a) {
   int h = 1;
   while (3 * h + 1 < a.length) h = 3 * h + 1;
   while (h > 0) {
      for (int i = h; i < a.length; i++) {
         for (int j = i; j >= h  && a[j - h].compareTo(a[j]) > 0; j -= h) {
          exchange(a, j - h, j);
         }
      }
      h /= 3;
   }
}
```

Baseado no *h* do Knuth

Exemplo

0	1	2	3	4	5	6	7
7	12	5	4	2	1	8	9

Primeiro passo h = 4

```
while (h * 3 + 1 < a.length) h = 3 * h + 1;
```

0	1	2	3	4	5	6	7
2	12	5	4	7	1	8	9

Troca 7 > 2

0	1	2	3	4	5	6	7
2	1	5	4	7	12	8	9

Troca 12 > 1

0	1	2	3	4	5	6	7
2	1	5	4	7	12	8	9

Não troca 5 < 8

0	1	2	3	4	5	6	7
2	1	5	4	7	12	8	9

Não troca 4 < 9

Exemplo

0	1	2	3	4	5	6	7
2	1	5	4	7	12	8	9

Segundo passo h = 1

```
h /= 3;
```

0	1	2	3	4	5	6	7
1	2	5	4	7	12	8	9

0	1	2	3	4	5	6	7
1	2	4	5	7	12	8	9

0	1	2	3	4	5	6	7
1	2	4	5	7	8	12	9

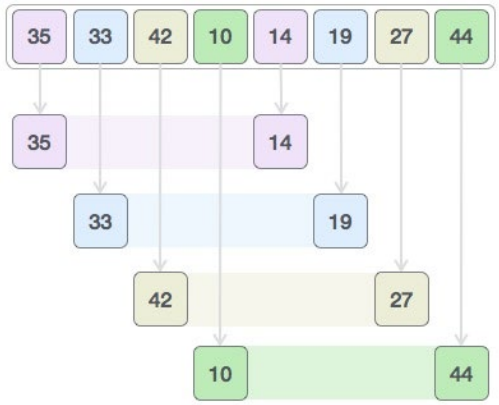
0	1	2	3	4	5	6	7
1	2	4	5	7	8	9	12

Com *h = 1* ocorre o processo do Insertion Sort tradicional.
O exemplo acima contém apenas os elementos trocados.

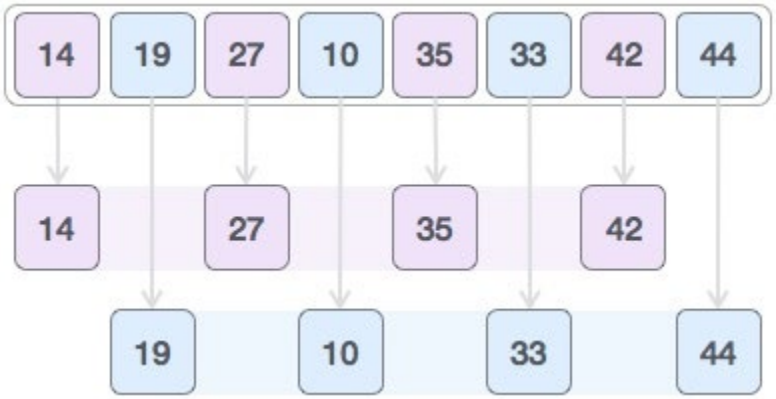
Exemplo com h = n / 2.

- Ao ser considerado o gap de n/2 conforme artigo original, teríamos três passos para o array abaixo:

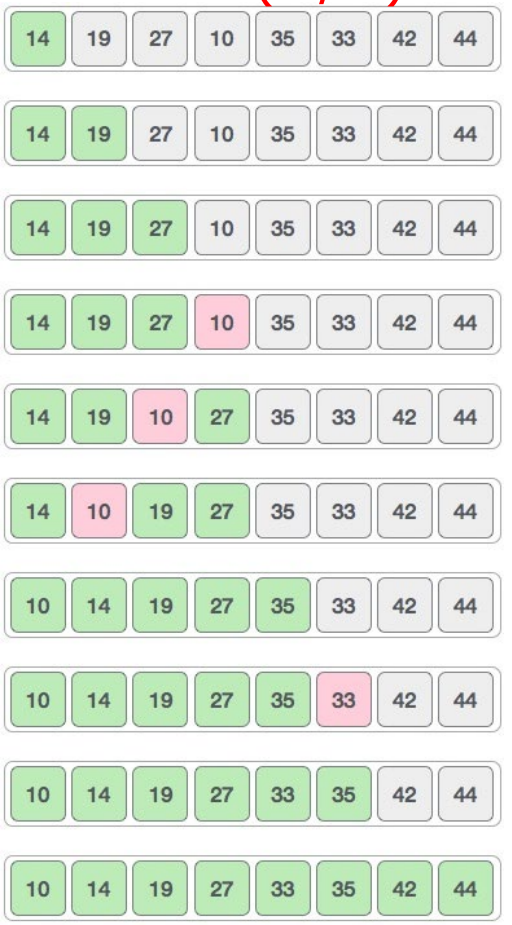
h = 4 (8 / 2)



h = 2 (4 / 2)



h = 1 (2 / 2)



Complexidade

- A análise do desempenho do método é complexa, envolvendo problemas matemáticos difíceis, alguns deles ainda não resolvidos; por quê? A complexidade depende do cenário e do gap (h);
- Por exemplo, um dos problemas é determinar o efeito que a ordenação dos segmentos em um passo produz nos passos subsequentes;
- Também não se conhece a melhor sequência de incrementos que produz o melhor resultado;
- Acredita-se que seja qualquer coisa entre $O(n \log n)$ e $O(n^{3/2})$.

Complexidade (cont.)

Método	Caso médio	Melhor caso	Pior caso	Complexidade de Espaço	Estável	Interno	Recursivo	Comparação
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Shell Sort	**$O(n^{7/6})$ – depende do gap**	**O(nlog(n))**	**O(nlog(n)) a $O(n^{3/2})$**	**In-place = O(1)**	**Não**	**Sim**	**Não**	**Sim**

Exercícios Teóricos

- Exercício 1. Considerando o seguinte array:

a) Aplique o Shellsort (passo-a-passo) somente para as chaves que mudarem.

Referências Bibliográficas

- CORMEN, Thomas H. et al. **Introduction to algorithms.** 3. ed. Cambridge: MIT, 2009. xix. 1292 p.

- https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm. Acessado em 17/10/2017.