BCC202 - Estruturas de Dados I

Aula 16: Ordenação: ShellSort

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

2021

Conteúdo

Conteúdo

ShellSort

- Proposto por **Donald Shell** em 1959.
- É uma extensão do ***InsertionSort***.
- Problemas com o algoritmo de *ordenação por inserção*:
  - Troca itens adjacentes para determinar o ponto de inserção.
    - São efetuadas $\boldsymbol{n-1}$ comparações e movimentações quando o menor item está na posição mais à direita do vetor.
  - Ineficiente para $\boldsymbol{n}$ grande
- O método de **Shell** contorna este problema permitindo trocas de registros distantes um do outro.

Sequência *h – ordenada*

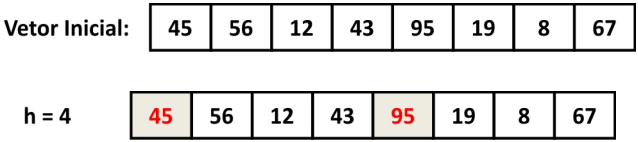
- Os itens separados de ***h*** posições são rearranjados, gerando sequências ordenadas.
- É dito que cada seqüência está **h-ordenada**.
- Depois, o valor de ***h*** é reduzido progressivamente, até atingir o valor 1, que resultará no vetor completamente ordenado.

Conteúdo

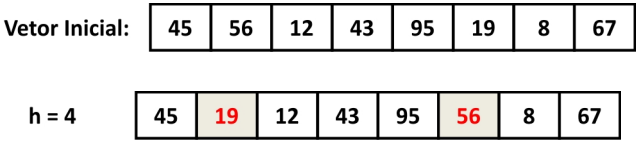
Exemplo I

Vetor Inicial:	45	56	12	43	95	19	8	67

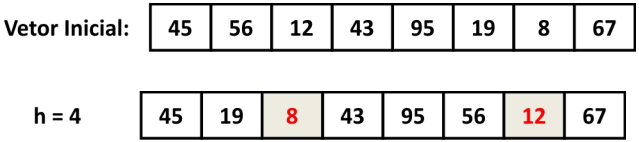
Exemplo I



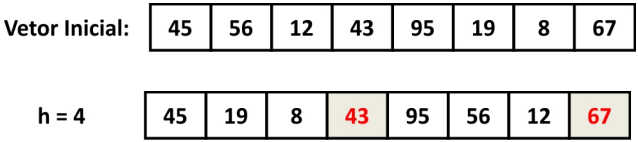
Exemplo I



Exemplo I



Exemplo I

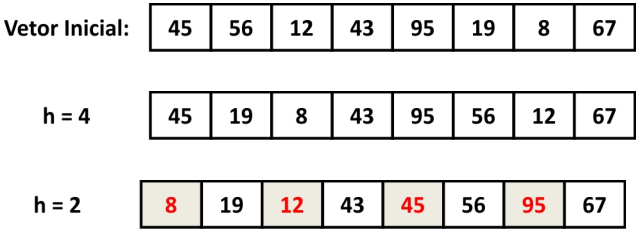


Exemplo I

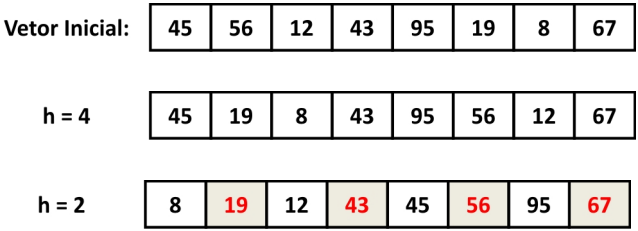
Vetor Inicial:	45	56	12	43	95	19	8	67

h = 4	45	19	8	43	95	56	12	67

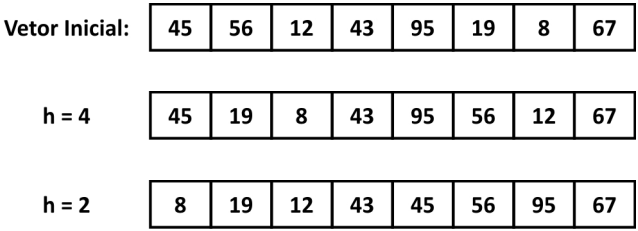
Exemplo I


Vetor Inicial:
45 56 12 43 95 19 8 67
h = 4
45 19 8 43 95 56 12 67
h = 2
8 19 12 43 45 56 95 67

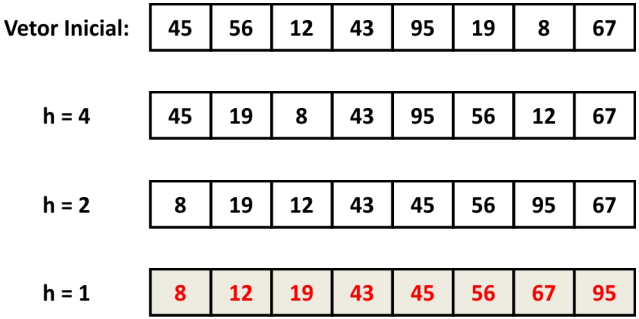
Exemplo I


Vetor Inicial:
45 56 12 43 95 19 8 67
h = 4
45 19 8 43 95 56 12 67
h = 2
8 19 12 43 45 56 95 67

Exemplo I


Vetor Inicial:
45 56 12 43 95 19 8 67
h = 4
45 19 8 43 95 56 12 67
h = 2
8 19 12 43 45 56 95 67

Exemplo I


Vetor Inicial:
45 56 12 43 95 19 8 67
h = 4
45 19 8 43 95 56 12 67
h = 2
8 19 12 43 45 56 95 67
h = 1
8 12 19 43 45 56 67 95

Exemplo I

Vetor Inicial:	45	56	12	43	95	19	8	67
h = 4	45	19	8	43	95	56	12	67
h = 2	8	19	12	43	45	56	95	67
h = 1	8	12	19	43	45	56	67	95

Exemlo II

- A lista tem nove itens. Se usarmos um incremento de três ($h = 3$), em vez de quebrar a lista em sublistas de itens contíguos:
  - Três *"sublistas*
    , cada uma sendo submetida à ordenação por inserção.

Figura: Elementos 3 – *ordenados*

Exemplo II

Os itens $3-$ *distantes* ficam mais próximos de onde pertencem no vetor ordenado.

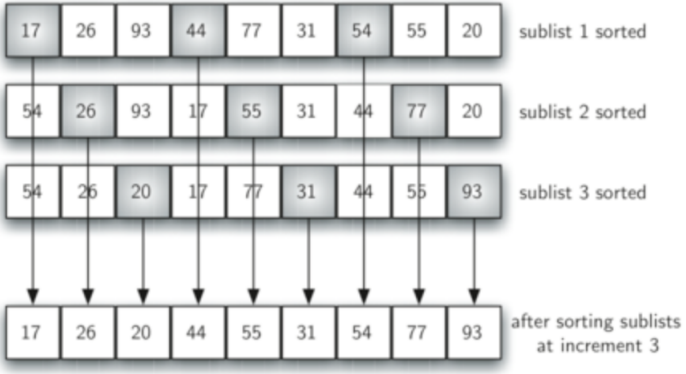
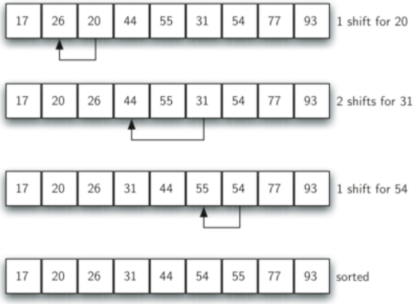


Figura: Elementos mais próximos da ordenação final

Exemplo II

- Ordenação por inserção final usando um incremento de $h = 1$; em outras palavras, uma ordenação por inserção convencional.
- Reduzido o número total de operações de deslocamento (4) necessárias para chegar ao vetor ordenado.



A seguir, mais detalhes sobre o passo a passo do *shellsort*.


Animação


Vídeo

Algumas observações sobre o *shellsort*.

- À medida que $\boldsymbol{h}$ decresce, o vetor vai ficando cada vez mais próximo da ordenação.
- Com $\boldsymbol{h} = \mathbf{1}$, o algoritmo se comporta exatamente igual ao **InsertionSort**.
- Mas, como podemos escolher uma boa sequência de valores para $\boldsymbol{h}$?

Conteúdo

Escolha de h

- Como escolher o valor de $\boldsymbol{h}$?
  - Para $\boldsymbol{s = 1}$: $\boldsymbol{h(s) = 1}$;
  - Para $\boldsymbol{s > 1}$: $\boldsymbol{h(s) = 3h(s-1)+1}$
- A sequência para $\boldsymbol{h}$ corresponde a:

$$\mathbf{1, 4, 13, 40, 121, 364, 1.093, 3.280, ...}$$

- *Knuth (1973, p. 95)* mostrou experimentalmente que esta seqüência é difícil de ser batida por mais de 20% em eficiência.
- Para conhecer outras sequências, clique aqui.

```
1  Algorithm: SHELLSORT
   Input: int* v, int n
2  begin
3      for h ← 1 to h < n do
4          h = 3 * h + 1 // h inicial
5      end
6      do
7          h ← (h − 1)/3 // atualiza h
8          for i ← h to i < n do
9              aux ← v[i] j ← i
10             while v[j − h] > aux do
11                 v[j] ← v[j − h]
12                 j = j − h
13                 if j < h then
14                     interrompe - sair do bloco de repetição
15                 end
16             end
17             v[j] ← aux
18         end
19     while h ≠ 1;
20 end
```

Conteúdo

Tempo, Comparação e Espaço

Depende também dos valores de h escolhidos.

- Conjecturas referentes ao número de comparações para a sequência de Knuth:
  - Conjectura 1: $\boldsymbol{C(n) = O(n^{1.25})}$.
  - Conjectura 2: $\boldsymbol{C(n) = O(n\ (ln\ n)^2)}$.
  - Complexidade de espaço/consumo de espaço
    - Extra: $O(1)$

Vantagens e Desvantagens

Vantagens

- Shellsort é uma ótima opção para arquivos de tamanho moderado.
- Sua implementação é simples e requer uma quantidade de código pequena.

Desvantagens

- O tempo de execução do algoritmo é sensível à ordem inicial do arquivo.
- O método **não é estável**.

Conteúdo

- *Algoritmo de ordenação* chamado **Shellsort**: tende a melhorar a ordenação por inserção.
- Sua complexidade não é bem conhecida, mas, em geral, é adequado para ordenar pequenas quantidades de dados.

Quadro Comparativo

- Quadro comparativo dos métodos de ordenação:

Algoritmo	Comparações			Movimentações			Espaço	Estável	In situ
	Melhor	Médio	Pior	Melhor	Médio	Pior			
Bubble	$O(n^2)$			$O(n^2)$			$O(1)$	**Sim**	**Sim**
Selection	$O(n^2)$			$O(n)$			$O(1)$	**Não***	**Sim**
Insertion	$O(n)$	$O(n^2)$		$O(1)$	$O(n^2)$		$O(1)$	**Sim**	**Sim**
Merge	$O(n \log n)$			$O(n \log n)$			$O(n)$	**Sim**	**Não**
Quick	$O(n \log n)$		$O(n^2)$	–			$O(n)$	**Não***	**Sim**
Shell	$O(n^{1.25})$ ou $O(n\ (ln\ n)^2)$			–			$O(1)$	**Não**	**Sim**

* Existem versões estáveis.

Filas de Prioridade (*Heap* e *HeapSort*)

Conteúdo

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

- O Shell Sort, disponíveil em `https://panda.ime.usp.br/pythonds/static/pythonds_pt/05-OrdenacaoBusca/OShellSort.html`. Acessado em 2021.

Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.
Introduction to Algorithms.
The MIT Press, 2011.

Exercício

- Dada a sequência de números: `3 4 9 2 5 1 8`.
- Ordene em ordem crescente utilizando o algoritmo aprendido em sala (**ShellSort**), apresentando a sequência dos números a cada passo (Teste de Mesa).