Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

Ordenação – ShellSort

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- Em geral, a atividade de ordenação é o processo de rearranjo de um certo conjunto de objetos (elementos) de acordo com um critério (ordem) específico.

- O objetivo da ordenação é facilitar a localização dos membros de um conjunto de objetos.

1. Introdução

- Anteriormente, foram apresentados três algoritmos de ordenação com complexidade $O(n^2)$:
  - *Bubblesort*
    - Na pratica é o pior dos três, raramente usado.
  - *SelectionSort*
    - Bom quando comparações são muito mais baratas que trocas.
  - *InsertionSort*
    - O melhor dos três na prática, com otimizações que melhoraram os resultados empíricos.
- Existem muitos outros, cada um apresentando vantagens e desvantagens em relação aos demais.

1. Introdução

- Pode-se esperar algum nível melhor de eficiência para um algoritmo de ordenação?
  - O limite $O(n^2)$ precisa ser quebrado para melhorar a eficiência e o tempo de execução.
  - Como isso pode ser feito?

2. Algoritmo ShellSort

- Refinamento do método de ordenação por inserção *(InsertionSort)*, proposto em 1959 por Donald L. Shell.

- Explora o fato de que o método de inserção direta apresenta desempenho aceitável quando o número de chaves é pequeno e/ou estas já estão parcialmente ordenadas.

2. Algoritmo ShellSort

- A inserção direta troca itens adjacentes quando está procurando o ponto de inserção na sequência destino.

- ***Shellsort***: troca de registros que estão distantes um do outro:

  - Itens que estão separados **h** posições são rearranjados de tal forma que todo **h-ésimo** item leva a uma sequência ordenada.

  - Também conhecido como ordenação por inserção através de incrementos decrescentes.

2. Algoritmo ShellSort

- O algoritmo consiste em dividir o vetor em **h** segmentos, de tal forma que cada um possua aproximadamente **n/h** chaves e classificar cada segmento separadamente.

- Para implementar o método, o vetor v[**1..n**] é dividido em **h** segmentos:

  - **Segmento 1**: v[1], v[h+1], v[2h+1], v[3h+1], ...
  - **Segmento 2**: v[2], v[h+2], v[2h+2], v[3h+2], ...
  - **Segmento 3**: v[3], v[h+3], v[2h+3], v[3h+3], ...
  - ...
  - **Segmento h**: v[h], v[h+h], v[2h+h], v[3h+h], ...

  - Onde **h** é o incremento.

2. Algoritmo ShellSort

- Como escolher o valor de **h**
  - Sequência de valores para **h**:
    - $h_1 = 1$ ← Primeiro valor de **h**
    - $h_i = 3h_{i-1} + 1,\ h_i < n$ ← Demais valores de **h**

- Dessa forma, a sequência para **h** corresponde a 1, 4, 13, 40, 121, 364, 1.093, 3.280, ...
  - Contudo, diferentes sequências de **h** podem ser usadas.

- Knuth (1973, p. 95) mostrou experimentalmente que esta sequência é difícil de ser batida por mais de 20% em eficiência.

2. Algoritmo ShellSort

- **Passos do algoritmo:**
  - Inicialmente, todos os elementos que estiverem a intervalos de **h** posições entre si na sequência corrente são agrupados e ordenados separadamente
    - **h inicial**: maior valor de **h**, tal que **h** < **n**.
  - Após este primeiro passo, os elementos são reagrupados em grupos com elementos cujo intervalo é de **h**/**3** posições, sendo então ordenados novamente.
  - O processo se repete até que **h** = **1**. Quando for feita a classificação com **h** = **1**, o vetor estará ordenado.

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3; /* h = ( h - 1 ) / 3 */
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

h = **1**

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

h = **13**

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

h = **4**

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
i	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
     ➡ aux = v[i];
     ➡ j = i;
       while (v[j-h].chave > aux.chave)
       { v[j] = v[j - h];
         j -= h;
         if (j < h) break; }
       v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
j, i	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

aux = 6

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
j-h	0	4
	1	8
	2	2
	3	7
j, i	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

aux = 6

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    ➡ v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
j, i	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

aux = 6

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
i	5	3
	6	5
	7	1
	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
     ➡ aux = v[i];
     ➡ j = i;
       while (v[j-h].chave > aux.chave)
       { v[j] = v[j - h];
         j -= h;
         if (j < h) break; }
       v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
j, i	5	3
	6	5
	7	1
	8	9
	9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
j-h	1	8
	2	2
	3	7
	4	6
j, i	5	3
	6	5
	7	1
	8	9
	9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

	n = 10	Pos.	Chave
		0	4
	j-h	1	8
		2	2
		3	7
		4	6
	j, i	5	8
		6	5
		7	1
		8	9
		9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
j	1	8
	2	2
	3	7
	4	6
i	5	8
	6	5
	7	1
	8	9
	9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
      ➡ if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
j	1	8
	2	2
	3	7
	4	6
i	5	8
	6	5
	7	1
	8	9
	9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    ➡ v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
j	1	3
	2	2
	3	7
	4	6
i	5	8
	6	5
	7	1
	8	9
	9	10

aux = 3

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	7
	4	6
	5	8
i	6	5
	7	1
	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
    ➡ aux = v[i];
    ➡ j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	7
	4	6
	5	8
j, i	6	5
	7	1
	8	9
	9	10

aux = 5

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
j-h	2	2
	3	7
	4	6
	5	8
j, i	6	5
	7	1
	8	9
	9	10

aux = 5

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    ➡ v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	7
	4	6
	5	8
j, i	6	**5**
	7	1
	8	9
	9	10

aux = 5

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	7
	4	6
	5	8
	6	5
i	7	1
	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
    ➡ aux = v[i];
    ➡ j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	7
	4	6
	5	8
	6	5
j, i	7	1
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
j-h	3	7
	4	6
	5	8
	6	5
j, i	7	1
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
j-h	3	7
	4	6
	5	8
	6	5
j, i	7	7
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
      ➡ j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
j	3	7
	4	6
	5	8
	6	5
i	7	7
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
      ➡ if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
j	3	7
	4	6
	5	8
	6	5
i	7	7
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    ➡ v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
j	3	1
	4	6
	5	8
	6	5
i	7	7
	8	9
	9	10

aux = 1

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
i	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
     ➡ aux = v[i];
     ➡ j = i;
       while (v[j-h].chave > aux.chave)
       { v[j] = v[j - h];
         j -= h;
         if (j < h) break; }
       v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
j, i	8	9
	9	10

aux = 9

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
j-h	4	6
	5	8
	6	5
	7	7
j, i	8	9
	9	10

aux = 9

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
j, i	8	9
	9	10

aux = 9

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
i	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
    ➡ aux = v[i];
    ➡ j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
j, i	9	10

aux = 10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
    ➡ while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
j-h	5	8
	6	5
	7	7
	8	9
j, i	9	10

aux = 10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
    ➡ v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
j, i	9	10

aux = 10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
	9	10

h = 4

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
	9	10

h = **1**

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	4
i	1	3
	2	2
	3	1
	4	6
	5	8
	6	5
	7	7
	8	9
	9	10

h = 1

3. Implementação

```
void shellsort (Item *v, int n) {
  int i, j, h = 1;
  Item aux;

  do h = h * 3 + 1; while (h < n);
  do {
    h /= 3;
    for (i = h ; i < n ; i++ ) {
      aux = v[i];
      j = i;
      while (v[j-h].chave > aux.chave)
      { v[j] = v[j - h];
        j -= h;
        if (j < h) break; }
      v[j] = aux; }
  } while (h != 1);
}
```

n = 10	Pos.	Chave
	0	1
	1	2
	2	3
	3	4
	4	5
	5	6
	6	7
	7	8
	8	9
	9	10

h = 1

4. Análise

- Como o primeiro incremento usado é grande, os segmentos individuais são pequenos e as ordenações com inserção direta são razoavelmente velozes.
- Cada ordenação parcial dos segmentos favorece o desempenho dos passos seguintes, uma vez que a inserção direta é acelerada quando o vetor já se encontra parcialmente ordenado.
- Embora a cada passo se use incrementos menores, e portanto, segmentos maiores, estes se encontram praticamente ordenados.

4. Análise

- A razão da eficiência do algoritmo ainda não é conhecida.
  - Ninguém ainda foi capaz de analisar o algoritmo.
  - A sua análise contém alguns problemas matemáticos muito difíceis.
    - A começar pela própria sequência de incrementos.
  - O que se sabe é que cada incremento não deve ser múltiplo do anterior.

4. Análise

- Conjecturas referente ao número de comparações para a sequência de Knuth:
  - **Conjectura 1**: $C(n) = \mathbf{O(n^{1,25})}$
  - **Conjectura 2**: $C(n) = \mathbf{O(n \times (\ln n)^2)}$
- Embora essas sejam melhorias significativas em relação à $\mathbf{O(n^2)}$, existem outros algoritmos ainda melhores.

4. Análise

- **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.

5. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- Horowitz, E. & Sahni, S.; Fundamentos de Estruturas de Dados, Editora Campus, 1984.
- Wirth, N.; Algoritmos e Estruturas de Dados, Prentice/Hall do Brasil, 1989.
- Material de aula do Prof. José Augusto Baranauskas, da USP: https://dcm.ffclrp.usp.br/~augusto/teaching.htm
- Material de aula do Prof. Rafael C. S. Schouery, da Unicamp: https://www.ic.unicamp.br/~rafael/cursos/2s2019/mc202/index.html