# Shellsort

- Mejora en la ordenación por inserción.
- Comienza comparando elementos muy separados, luego elementos menos separados,
  y finalmente comparando elementos adyacentes (efectivamente una ordenación por inserción).
  En esta etapa, los elementos se clasifican lo suficiente como para que el tiempo
  de ejecución de la etapa final esté mucho más cerca de O (N) que O ($N^2$).

Shellsort, también conocido como el **tipo de incremento decreciente** , es uno de los algoritmos de clasificación más antiguos, llamado así por su inventor Donald. L. Shell (1959). Es rápido, fácil de entender y fácil de implementar. Sin embargo, su análisis de complejidad es un poco más sofisticado.

La idea de Shellsort es la siguiente:

a) organizar la secuencia de datos en una matriz bidimensional
b) ordenar las columnas de la matriz

El efecto es que la secuencia de datos está parcialmente ordenada.
El proceso anterior se repite, pero cada vez con una matriz más estrecha, es decir, con un número menor de columnas. En el último paso, la matriz consta de solo una columna.

En cada paso, la ordenación de la secuencia aumenta, hasta que en el último paso está completamente ordenada. Sin embargo, el número de operaciones de clasificación necesarias en cada paso es limitado, debido a la preclasificación de la secuencia obtenida en los pasos anteriores.

Ejemplo:

Dejar

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

ser la secuencia de datos que se ordenará Primero, está organizado en una matriz con 7 columnas (izquierda), luego las columnas están ordenadas (derecha):

| | |
|---|---|
| 3 7 9 0 5 1 6<br>8 4 2 0 6 1 5<br>7 3 4 9 8 2 | 3 3 2 0<br>5 1 5<br>7 4 4 0<br>6 1 6<br>8 7 9 9<br>8 2 |

Los elementos de datos 8 y 9 ya han llegado al final de la secuencia, pero un pequeño elemento (2) también está allí. En el siguiente paso, la secuencia se organiza en 3 columnas, que se ordenan de nuevo:

| | | |
|---|---|---|
| 3 3 2 | | 0 0 1 |
| 0 5 1 | | 1 2 2 |
| 5 7 4 | | 3 3 4 |
| 4 0 6 | ® | 4 5 6 |
| 1 6 8 | | 5 6 8 |
| 7 9 9 | | 7 7 9 |
| 8 2 | | 8 9 |

Ahora la secuencia está casi completamente ordenada. Al organizarlo en una columna en el último paso, es solo un 6, un 8 y un 9 que tienen que moverse un poco a sus posiciones correctas.

## 2. Implementación

En realidad, la secuencia de datos no está dispuesta en una matriz bidimensional, sino que se mantiene en una matriz unidimensional que está indexada de manera apropiada.

El algoritmo usa una secuencia de incremento para determinar qué tan separados están los elementos para ser ordenados: $h_1$ , $h_2$ , ..., $h_t$ con $h_1 = 1$

Al principio, los elementos a la distancia $\mathbf{h_t}$ se clasifican, luego los elementos a la distancia $\mathbf{h_{t-1}}$ se ordenan, etc., hasta que finalmente la matriz se ordena usando la ordenación por inserción (distancia $h_1 = 1$).

Una matriz se dice que es $\mathbf{h_k}$ -sorted si todos los elementos espaciados una distancia $\mathbf{h_k}$ aparte se clasifican respecto a la otra.

Shellsort solo funciona porque una matriz que es $\mathbf{h_{k-}}$ ordenada sigue $\mathbf{h_{k-}}$ clasificada cuando $\mathbf{h_{k-1}}$-clasificada. Esto significa que los géneros posteriores con un incremento menor no deshacen el trabajo realizado por fases anteriores.

## 3. Corrección del algoritmo

La corrección del algoritmo se deriva del hecho de que en el último paso (con *h* = 1) se realiza una clasificación de inserción común en toda la matriz. Debido a que los datos se clasifican previamente según los pasos anteriores

( *h* = 3, 7, 21, ...), solo son suficientes unos pocos pasos de clasificación de inserción.

## 4. Código

Aquí, si hacemos caso omiso del bucle externo y reemplazamos la **brecha** con 1, terminamos con la ordenación por inserción. Por lo tanto, el siguiente código es muy similar al código de ordenación de inserción, con las siguientes diferencias:

- hay un bucle adicional (el bucle externo): cada ejecución procesa un incremento.
- el medio y los bucles más internos son los mismos que en el tipo de inserción con brecha = 1.

```
int j, p, gap;
tmp comparable;


for (gap = N / 2; gap> 0; gap = gap / 2)

  para (p = gap; p <N; p ++)
  {
   tmp = a [p];
   para (j = p; j> = espacio && tmp <a [j- gap]; j = j - espacio)

                a [j] = a [j-gap];

   a [j] = tmp;
  }

}
```

**Animación**

## 5. Secuencias incrementales

¿Cuál debería ser la secuencia de incremento?
Hay muchas opciones para la secuencia de incremento. Cualquier secuencia que comience en 1 y siempre aumente lo hará, aunque algunas rindan mejor rendimiento que otras:

1. Secuencia original de Shell: N / 2, N / 4, ..., 1 (divide repetidamente por 2);
2. Incrementos de Hibbard: 1, 3, 7, ..., $2^k - 1$;
3. Incrementos de Knuth: 1, 4, 13, ..., $(3^k - 1) / 2$;

4. Incrementos de Sedgewick: 1, 5, 19, 41, 109, ....
   Se obtiene intercalando los elementos de dos secuencias:

   1, 19, 109, 505, 2161, ... .., $9 (4^k - 2^k) + 1$, k = 0, 1, 2, 3, ...
   5, 41, 209, 929, 3905, ... ..$2^{k+2} (2^{k+2} - 3) + 1$, k = 0, 1, 2, 3, ...

## 6. Análisis

El peor rendimiento de un Shellsort con incrementos de Hibbard es **$\Theta (n^{3/2})$.**
Se cree que el rendimiento promedio es aproximadamente **$O (n^{5/4})$**

La complejidad exacta de este algoritmo aún se debate.

La experiencia muestra que para los datos medianos (decenas de miles de elementos) el algoritmo funciona casi tan bien o mejor que los *n* más rápidos .