ALGORITHM 2.3  Shellsort

```java
public class Shell {
    // Shellsort.
    public static void sort(Comparable[] a) {
        // Sort a[] into increasing order.
        int N = a.length;
        int h = 1;
        while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, 1093, ...
        while (h >= 1) {
            // h-sort the array.
            for (int i = h; i < N; i++) {
                // Insert a[i] among a[i-h], a[i-2*h], a[i-3*h]...
                for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }
}
```

If we modify insertion sort (Algorithm 2.2) to h-sort the array and add an outer loop to decrease h through a sequence of increments starting at an increment as large as a constant fraction of the array length and ending at 1, we are led to this compact shellsort implementation.

```
% java SortCompare Shell Insertion 100000 100
For 100000 random Doubles
    Shell is 600 times faster than Insertion
```

```
input  S H E L L S O R T E X A M P L E
13-sort P H E L L S O R T E X A M S L E
4-sort  L E E A M H L E P S O L T S X R
1-sort  A E E E H L L L M O P R S S T X

Shellsort trace (array contents after each pass)
```