6.10. The Shell Sort

The **shell sort**, sometimes called the "diminishing increment sort," improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort. The unique way that these sublists are chosen is the key to the shell sort. Instead of breaking the list into sublists of contiguous items, the shell sort uses an increment `i`, sometimes called the **gap**, to create a sublist by choosing all items that are `i` items apart.

This can be seen in Figure 6. This list has nine items. If we use an increment of three, there are three sublists, each of which can be sorted by an insertion sort. After completing these sorts, we get the list shown in Figure 7. Although this list is not completely sorted, something very interesting has happened. By sorting the sublists, we have moved the items closer to where they actually belong.

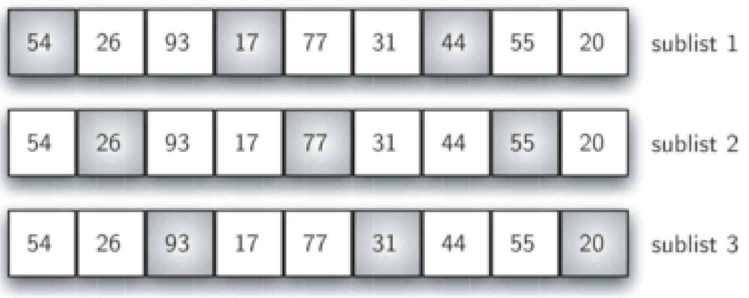

Figure 6: A Shell Sort with Increments of Three

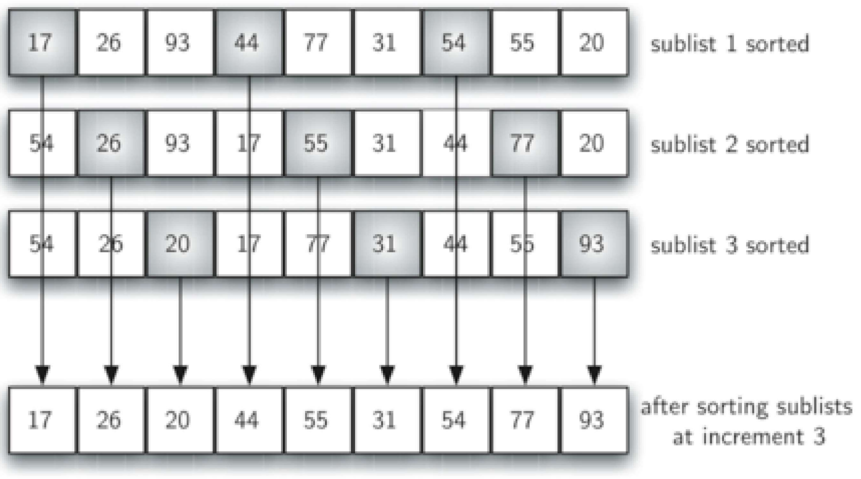

Figure 7: A Shell Sort after Sorting Each Sublist

Figure 8 shows a final insertion sort using an increment of one; in other words, a standard insertion sort. Note that by performing the earlier sublist sorts, we have now reduced the total number of shifting operations necessary to put the list in its final order. For this case, we need only four more shifts to complete the process.

(TheInsertionSort.html)

(TheMergeSort.ht

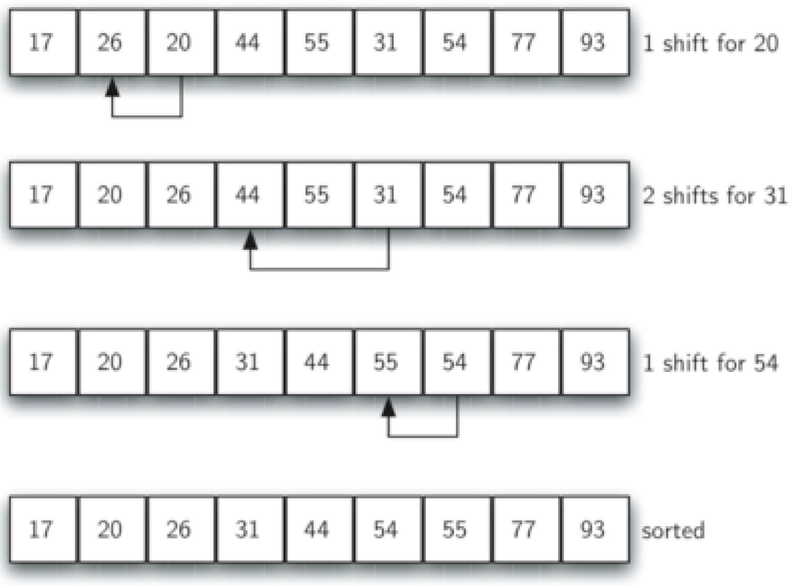

Figure 8: ShellSort: A Final Insertion Sort with Increment of 1

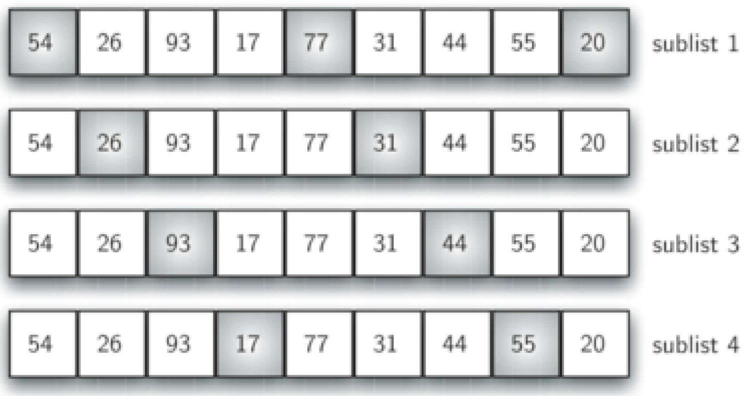

Figure 9: Initial Sublists for a Shell Sort

We said earlier that the way in which the increments are chosen is the unique feature of the shell sort. The function shown in ActiveCode 1 uses a different set of increments. In this case, we begin with $\frac{n}{2}$ sublists. On the next pass, $\frac{n}{4}$ sublists are sorted. Eventually, a single list is sorted with the basic insertion sort. Figure 9 shows the first sublists for our example using this increment.

The following invocation of the `shellSort` function shows the partially sorted lists after each increment, with the final sort being an insertion sort with an increment of one.

Run Load History Show CodeLens

```
def shellSort(alist):
    sublistcount = len(alist)//2
    while sublistcount > 0:

      for startposition in range(sublistcount):
        gapInsertionSort(alist,startposition,sublistcount)

      print("After increments of size",sublistcount,
                                   "The list is",alist)
```

(TheInsertionSort.html)
(TheMergeSo .ht

```

        sublistcount = sublistcount // 2

def gapInsertionSort(alist,start,gap):
    for i in range(start+gap,len(alist),gap):
```

Activity: 6.10.1 Shell Sort (lst_shellSort)

Initialize Run Stop

Beginning Step Forward Step Backward End

At first glance you may think that a shell sort cannot be better than an insertion sort, since it does a complete insertion sort as the last step. It turns out, however, that this final insertion sort does not need to do very many comparisons (or shifts) since the list has been pre-sorted by earlier incremental insertion sorts, as described above. In other words, each pass produces a list that is “more sorted” than the previous one. This makes the final pass very efficient.

Although a general analysis of the shell sort is well beyond the scope of this text, we can say that it tends to fall somewhere between $O(n)$ and $O(n^2)$, based on the behavior described above. For the increments shown in Listing 5, the performance is $O(n^2)$. By changing the increment, for example using $2^k - 1$ (1, 3, 7, 15, 31, and so on), a shell sort can perform at $O(n^{\frac{3}{2}})$.

Self Check

Q-3: Given the following list of numbers: [5, 16, 20, 12, 3, 8, 9, 17, 19, 7] Which answer illustrates the contents of the list after all swapping is complete for a gap size of 3?

(TheInsertionSort.html)

(TheMergeSort.ht

- ○ **A. [5, 3, 8, 7, 16, 19, 9, 17, 20, 12]**
- ○ **B. [3, 7, 5, 8, 9, 12, 19, 16, 20, 17]**
- ○ **C. [3, 5, 7, 8, 9, 12, 16, 17, 19, 20]**

○ **D. [5, 16, 20, 3, 8, 12, 9, 17, 20, 7]**

Check Me Compare me

Activity: 6.10.2 Multiple Choice (question_sort_4)

You have attempted 1 of 3 activities on this page

user not logged in

(TheInsertionSort.html) (TheMergeSort.ht