2 Insertion Sort

Wolfgang P. Kowalk

Carl-von-Ossietzky-Universität Oldenburg, Oldenburg, Germany

Let's sort our books in the bookcase by title so that each book can be accessed immediately if required.

How to achieve this quickly? We can use several concepts. For example, we can look at each book one after the other, and if two subsequent books are out of order we exchange them. This works since finally no two books are out of order, but it takes, on average, a very long time. Another concept looks for the book with the "smallest" title and puts it at first position; then from those books remaining the next book with smallest title is looked for, and so on, until all books are sorted. Also this works eventually; however, since a great deal of information is always ignored it takes longer than it should. Thus let's try something else.

The following idea seems to be more natural than those discussed above. The first book is sorted. Now we compare its title with the second book, and if it is out of order we exchange those two books. Now we look to find the correct position for the next book within the sequence of the first sorted books and place it there. This can be iterated until we have finally sorted all books. Since we can use information from previous steps this method seems to be most efficient.

Let us look more deeply at this algorithm. The first book alone is always sorted. We assume that all books to the left of current book i are sorted. To enclose book i in the sequence of sorted books we search for its correct position and put it there; to do this all, books on the right side of the correct place are shifted one position to the right. This is repeated with the next book at position $i + 1$, etc., until all the books are sorted. This method yields the correct result very quickly, particularly if the "Binary Search" method from Chap. 1 is used to find the place of insertion.

How can we apply this intuitive method so it is useful for any number of books? To simplify the notation we will write a number instead of a book title.

In Fig. 2.1 the five books 1, 6, 7, 9, 11 on the left side are already sorted; book number 5 is not correctly positioned. To place it at the correct position

B. Vöcking et al. (eds.), *Algorithms Unplugged*,
DOI 10.1007/978-3-642-15328-0_2, © Springer-Verlag Berlin Heidelberg 2011

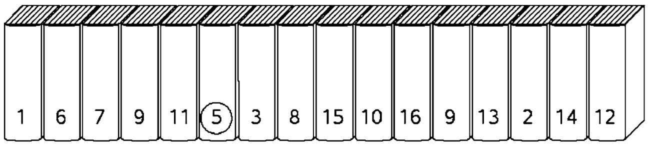

Fig. 2.1. The first five books are sorted

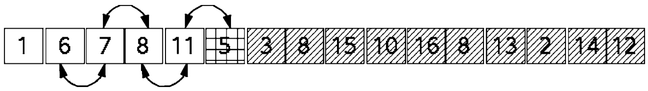

Fig. 2.2. Book "5" is situated at the correct position

we can exchange it with book number 11, then with book number 9, and so on, until it is placed at its correct position. Then we proceed with book number 3 and sort it by exchanging it with the books on the left-hand side. Obviously all books are eventually placed by this method at their correct position (see Fig. 2.2).

How can this be programmed? The following program answers this question. It uses an array of numbers A, where the cells of the array are numbered $1, 2, 3, \ldots$. Then $A[i]$ means the value at position i of array A. To sort n books requires an array of length n with cells $A[1], A[2], A[3], \ldots, A[n-1], A[n]$ to store all book titles. Then the algorithm looks like this:

Subsequent books are exchanged:

```
1   Given: $A$: Array with $n$ cells
2   for $i := 2$ to $n$ do
3       $j := i$;     // book at position $i$ is current
                         as long as correct position not achieved
4       while $j \geq 2$ and $A[j-1] > A[j]$ do
5           $Hand := A[j]$;     // exchange current book with left neighbor
6           $A[j] := A[j-1]$;
7           $A[j-1] := Hand$;
8           $j := j - 1$
9       endwhile
10  endfor
```

How long does sorting take with this algorithm? Lets take the worst case where all books are sorted vice versa, i.e., the book with smallest number is at last position, that with biggest number at first, and so on. Our algorithm changes the first book with the second, the third with the first two books, the fourth with the first three books, etc., until eventually the last book is to be changed with all other $n-1$ books. The number of exchanges is

$$1 + 2 + 3 + \cdots + (n-1) = \frac{n \cdot (n-1)}{2}.$$

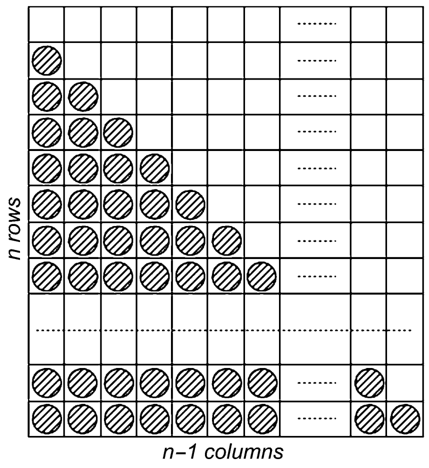

Fig. 2.3. Compute the number of exchanges

This formula is easily derived from Fig. 2.3. In the rectangle are $n \cdot (n-1)$ cells, and half of them are used for compare and exchange. This picture shows the absolute worst case. For the average case we assume that only half as many compares and exchanges are required. If the books are already almost sorted, then much less effort is required; in the best case if all books are sorted only $n-1$ comparisons have to be done.

You may have found that this algorithm is more cumbersome than necessary. Instead of exchanging two subsequent books, we shift all books to the right until the space for the book to be inserted is free.

Instead of exchanging k times two books, we shift $k+1$ times one book, which is more efficient. The algorithm look like this:

Sort books by insertion:

```
Given: $A$: array with $n$ cells;
**for** $i := 2$ **to** $n$ **do**
        // sort book at position $i$ by shifting
    $Hand := A[i]$;      // take current book
    $j := i - 1$;
        // as long as current position not found
    **while** $j \geq 1$ **and** $A[j] > Hand$ **do**
        $A[j+1] := A[j]$;      // shift book right to position $j$
        $j := j - 1$
    **endwhile**
    $A[j] := Hand$      // insert current book at correct position
**endfor**
```

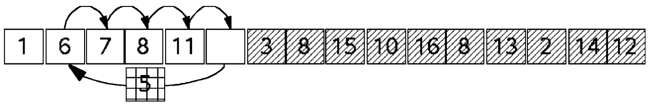

Fig. 2.4. Compute the number of exchanges

Further improvements of this sorting method, like inserting several books at once, and animations of this and other algorithms can be found at the Web site `http://einstein.informatik.uni-oldenburg.de/forschung/animAlgo/`

Considerations about computer hardware that can calculate shifting several books at the same time can be found in Chap. 4.

Even if sorting in normal computers requires a great deal of time, this algorithm is often used when the number of objects like books is not too big, or if you can assume that most books are almost sorted, since implementation of this algorithm is so simple. In the case of many objects to be sorted, other algorithms like MergeSort and QuickSort are used, which are more difficult to understand and to implement. They are discussed in Chap. 3.

To Read on

1. Insertion Sort is a standard algorithm that can be found in most textbooks about algorithms, for example, in Robert Sedgewick: *Algorithms in C++*. Pearson, 2002.
2. W.P. Kowalk: *System, Modell, Programm*. Spektrum Akademischer Verlag, 1996 (ISBN 3-8274-0062-7).