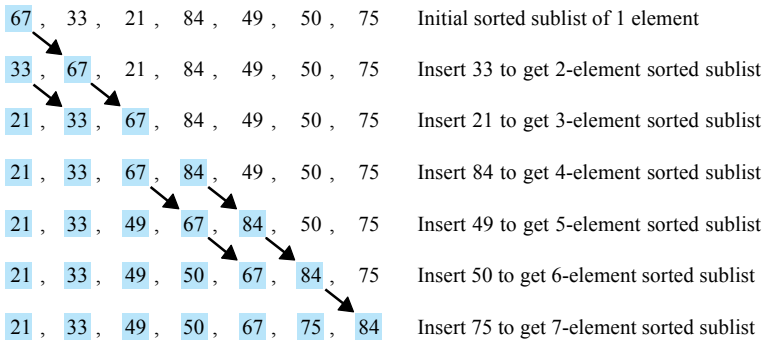
Other Sorting Methods

LINEAR INSERTION SORT

Linear insertion sort is based on the idea of repeatedly inserting a new element into a list of already sorted elements so that the resulting list is still sorted. The following sequence of diagrams demonstrates this method for the list 67, 33, 21, 84, 49, 50, 75. The sorted sublist produced at each stage is highlighted.



The following algorithm describes this procedure for lists stored in arrays. At the ith stage, $x[i]$ is inserted into its proper place among the already sorted $x[0],...,x[i-1]$. We do this by comparing x_i with each of these elements, starting from the right end, and shifting them to the right as necessary.

Linear Insertion Sort Algorithm

For $i = 1$ to $n - 1$ do the following:
- // Insert $x[i]$ into its proper position among $x[0],..., x[i-1]$.
- a. Set *nextElement* equal to $x[i]$.
- b. Set j equal to i.
- c. While j > 0 and *nextElement* < $x[j-1]$ do the following:
  - // Shift element to the right to open a spot
  - i. Set $x[j]$ equal to $x[j-1]$.
  - ii. Decrement j by 1.
- // Now drop *nextElement* into the open spot.
- d. Set $x[j]$ equal to *nextElement*.

As the diagram above indicates, more elements in the already-sorted sublist have to be moved for a smaller value being inserted than for a larger one. In particular, if the list is already sorted, no elements have to move! Thus, unlike simple selection sort, linear insertion sort takes advantage of any partial ordering of the elements that already exists.

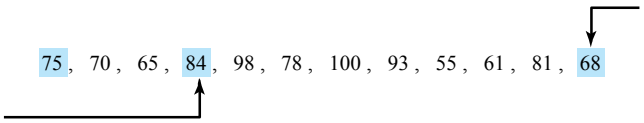
QUICKSORT

The **quicksort** method of sorting is one of the fastest methods of sorting and is most often implemented by a recursive algorithm. The basic idea of quicksort is to choose some element called a **pivot** and then to perform a sequence of exchanges so that all elements that are less than this pivot are to its left and all elements that are greater than the pivot are to its right. This correctly positions the pivot and divides the (sub)list into two smaller sublists, each of which may then be sorted independently in the *same* way. This **divide-and-conquer** strategy leads naturally to a recursive sorting algorithm.

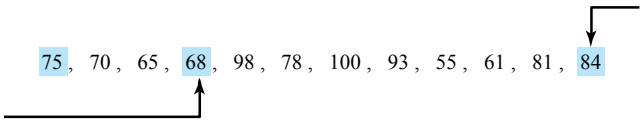
To illustrate this splitting of a list into two sublists, consider the following list of integers:

75, 70, 65, 84, 98, 78, 100, 93, 55, 61, 81, 68

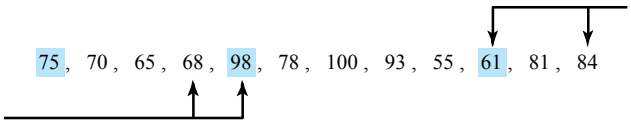
If we select the first number as the pivot, we must rearrange the list so that 70, 65, 55, 61, and 68 are placed before 75, and 84, 98, 78, 100, 93, and 81 are placed after it. To carry out this rearrangement, we search from the right end of the list for an element less than 75 and from the left end for an item greater than 75.



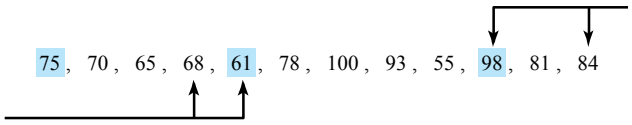
This locates the two numbers 68 and 84, which we now interchange to obtain



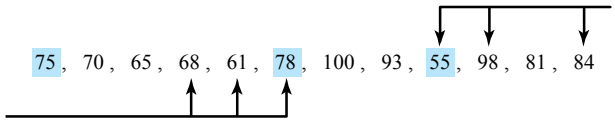
We then resume the search from the right for a number less than 75 and from the left for a number greater than 75:



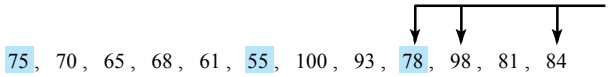
This locates the numbers 61 and 98, which are then interchanged:



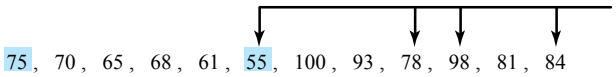
A continuation of the searches locates 78 and 55:



Interchanging these gives



Now, when we resume our search from the right, we locate the element 55 that was found on the previous search from the left:



The "pointers" for the left and right searches have thus met, and this signals the end of the two searches. We now interchange 55 and the pivot 75:

55, 70, 65, 68, 61, 75, 100, 93, 78, 98, 81, 84

Note that all elements to the left of 75 are less than 75 and that all those to its right are greater than 75, and thus the pivot 75 has been properly positioned.

The left sublist

55, 70, 65, 68, 61

and the right sublist

100, 93, 78, 98, 81, 84

can now be sorted *independently, using any sorting scheme desired.* Quicksort uses the same scheme we have just illustrated for the entire list; that is, these sublists must themselves be split by choosing and correctly positioning one pivot element (the first) in each of them.

A recursive method to sort a list is then easy to write and we leave it as an exercise. The anchor case occurs when the list being examined is empty or contains a single element; in this case the list is in order, and nothing needs to be done. The inductive case occurs when the list contains two or more elements, in which case the list can be sorted by:

1. Splitting the list into two sublists;
2. Recursively sorting the left sublist; and
3. Recursively sorting the right sublist.