The ***decrease-and-conquer*** technique is based on exploiting the relationship between a solution to a given instance of a problem and a solution to its smaller instance. Once such a relationship is established, it can be exploited either top down or bottom up. The former leads naturally to a recursive implementation, although, as one can see from several examples in this chapter, an ultimate implementation may well be nonrecursive. The bottom-up variation is usually implemented iteratively, starting with a solution to the smallest instance of the problem; it is called sometimes the ***incremental approach***.
There are three major variations of decrease-and-conquer:
decrease by a constant
decrease by a constant factor
variable size decrease

In the ***decrease-by-a-constant*** variation, the size of an instance is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one (Figure 4.1), although other constant size reductions do happen occasionally.
Consider, as an example, the exponentiation problem of computing *an* where $a \neq 0$ and n is a nonnegative integer. The relationship between a solution to an instance of size n and an instance of size $n - 1$ is obtained by the obvious formula $a^n = a^{n-1} . a$. So the function $f(n) =$ *an* can be omputed either "top down" by using its recursive definition or "bottom up" by multiplying 1 by a n times. (Yes, it is the same as the brute-force algorithm, but we have come to it by a different thought process.)

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0, \\ 1 & \text{if } n = 0, \end{cases}$$

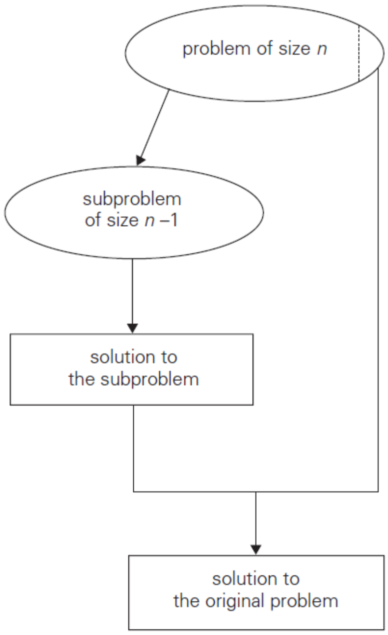

FIGURE 4.1 Decrease-(by one)-and-conquer technique.

The ***decrease-by-a-constant-factor*** technique suggests reducing a problem instance by the same constant factor on each iteration of the algorithm. In most applications, this constant factor is equal to two. For an example, let us revisit the exponentiation problem. If the instance of
size n is to compute a^n, the instance of half its size is to compute $an/2$, with the obvious relationship between the two: $an = (a^{n/2})^2$. But since we consider here instances with integer exponents only, the former does not work for odd n. If n is odd, we have to compute a^{n-1} by using the rule for even-valued exponents and then multiply the result by a.

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

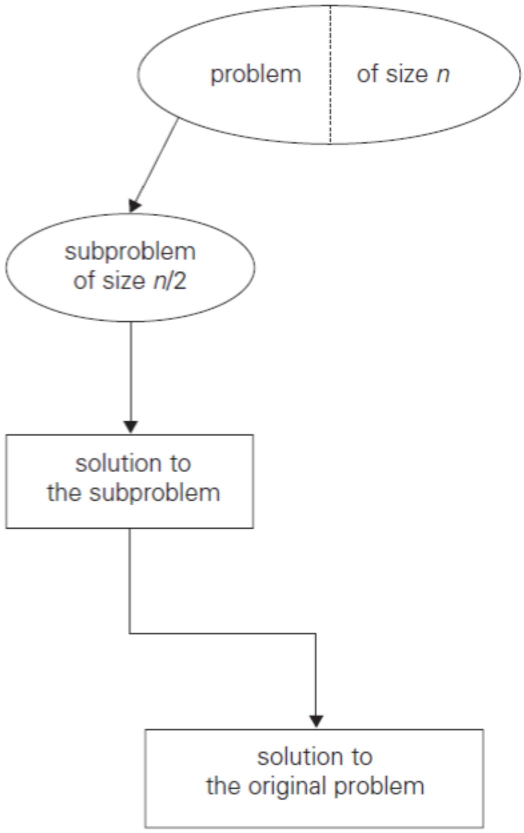

FIGURE 4.2 Decrease-(by half)-and-conquer technique.

Finally, in the ***variable-size-decrease*** variety of decrease-and-conquer, the size-reduction pattern varies from one iteration of an algorithm to another. Euclid's algorithm for computing the greatest common divisor provides a good example of such a situation. Recall that this algorithm is based on the formula gcd(m, n) = gcd$(n, m$ mod $n)$.

Insertion Sort

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

$$\underbrace{A[0] \le \cdots \le A[j]}_{\text{smaller than or equal to } A[i]} < \underbrace{A[j+1] \le \cdots \le A[i-1]}_{\text{greater than } A[i]} \;\big|\; A[i] \cdots A[n-1]$$

FIGURE 4.3 Iteration of insertion sort: $A[i]$ is inserted in its proper position among the preceding elements previously sorted.

89 $\mid$	**45**	68	90	29	34	17
45	89 $\mid$	**68**	90	29	34	17
45	68	89 $\mid$	**90**	29	34	17
45	68	89	90 $\mid$	**29**	34	17
29	45	68	89	90 $\mid$	**34**	17
29	34	45	68	89	90 $\mid$	**17**
17	29	34	45	68	89	90

FIGURE 4.4 Example of sorting with insertion sort. A vertical bar separates the sorted part of the array from the remaining elements; the element being inserted is in bold.

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$$

Thus, in the worst case, insertion sort makes exactly the same number of comparisons as selection sort (see Section 3.1).

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$