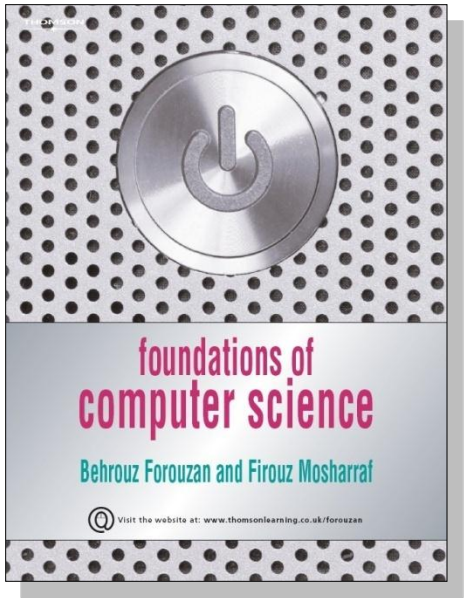
Chapter 8
Algorithms

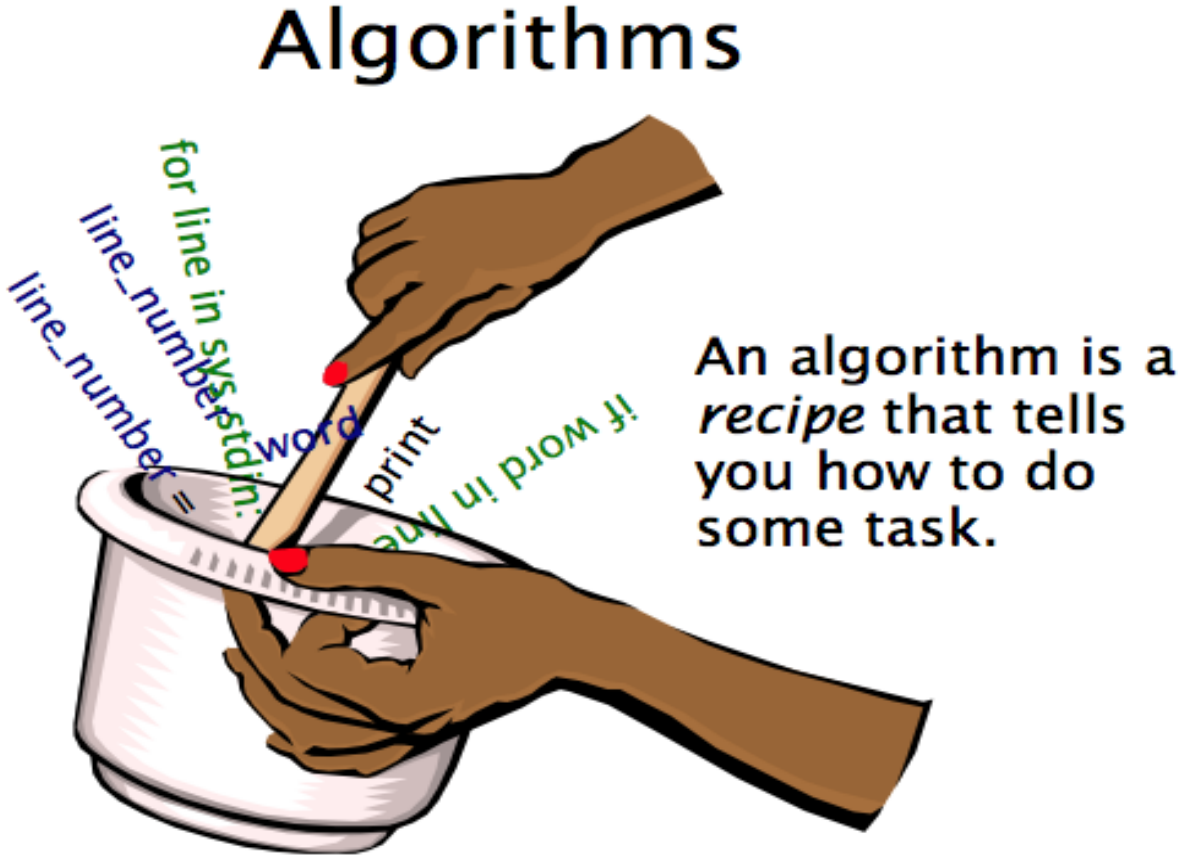


Objectives

After studying this chapter, the student should be able to:

- Define an algorithm and relate it to problem solving.
- Define three construct and describe their use in algorithms.
- Describe UML diagrams and pseudocode and how they are used in algorithms.
- List basic algorithms and their applications.
- Describe the concept of sorting and understand the mechanisms behind three primitive sorting algorithms.
- Describe the concept of searching and understand the mechanisms behind two common searching algorithms.
- Define subalgorithms and their relations to algorithms.
- Distinguish between iterative and recursive algorithms.

8.1 CONCEPT


Algorithms
for line in sys.stdin:
line_number
line_number =
word
print
if word in line
An algorithm is a *recipe* that tells you how to do some task.

Informal definition

An informal definition of an algorithm is:

Algorithm: a step-by-step method for solving a problem or doing a task.

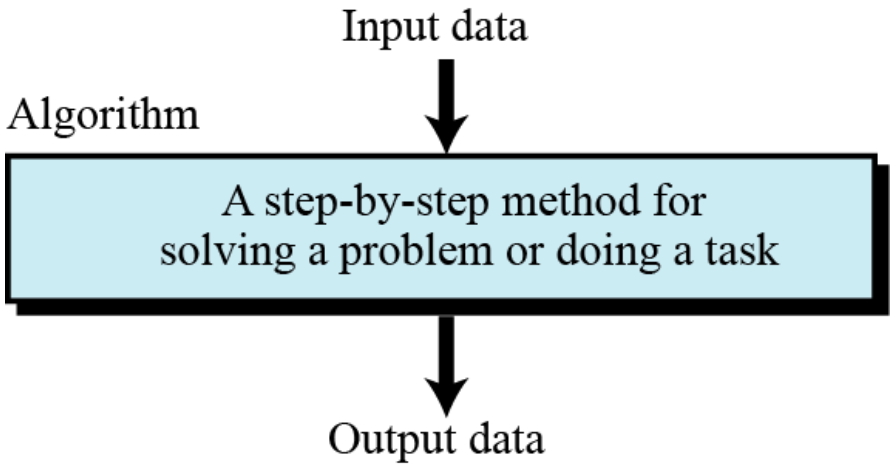


Figure 8.1 **Informal definition of an algorithm used in a computer**

Example:

- ✓ We want to develop an algorithm for finding the largest integer among a list of positive integers.
- ✓ The algorithm should find the largest integer among a list of any values (for example 5, 1000, 10,000, 1,000,000).
- ✓ The algorithm should be general and not depend on the number of integers.

➢ **To solve this problem, we need an intuitive approach.**

First use a small number of integers (for example, five), then extend the solution to any number of integers.

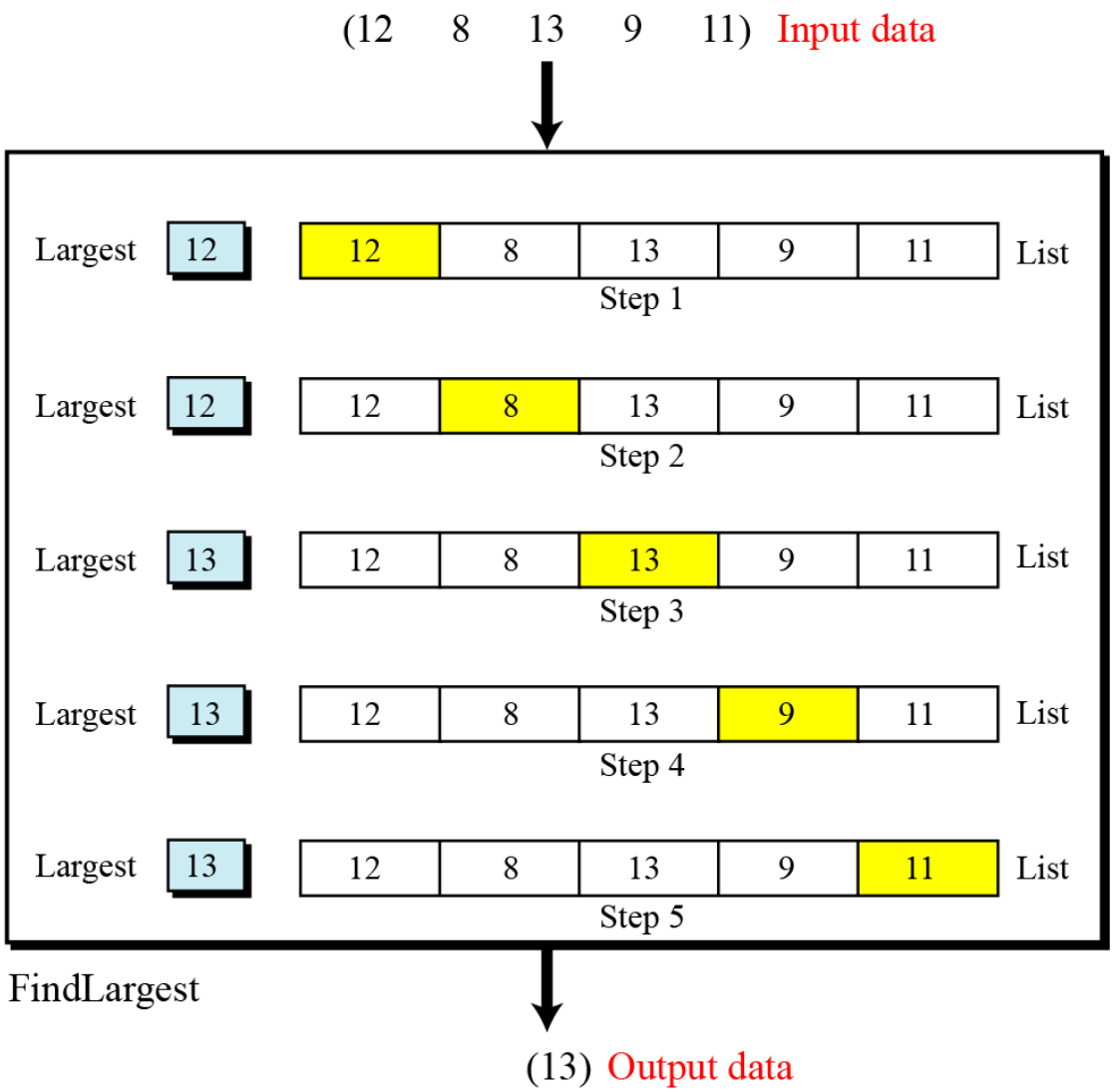


Figure 8.2 **Finding the largest integer among five integers**

Defining actions:

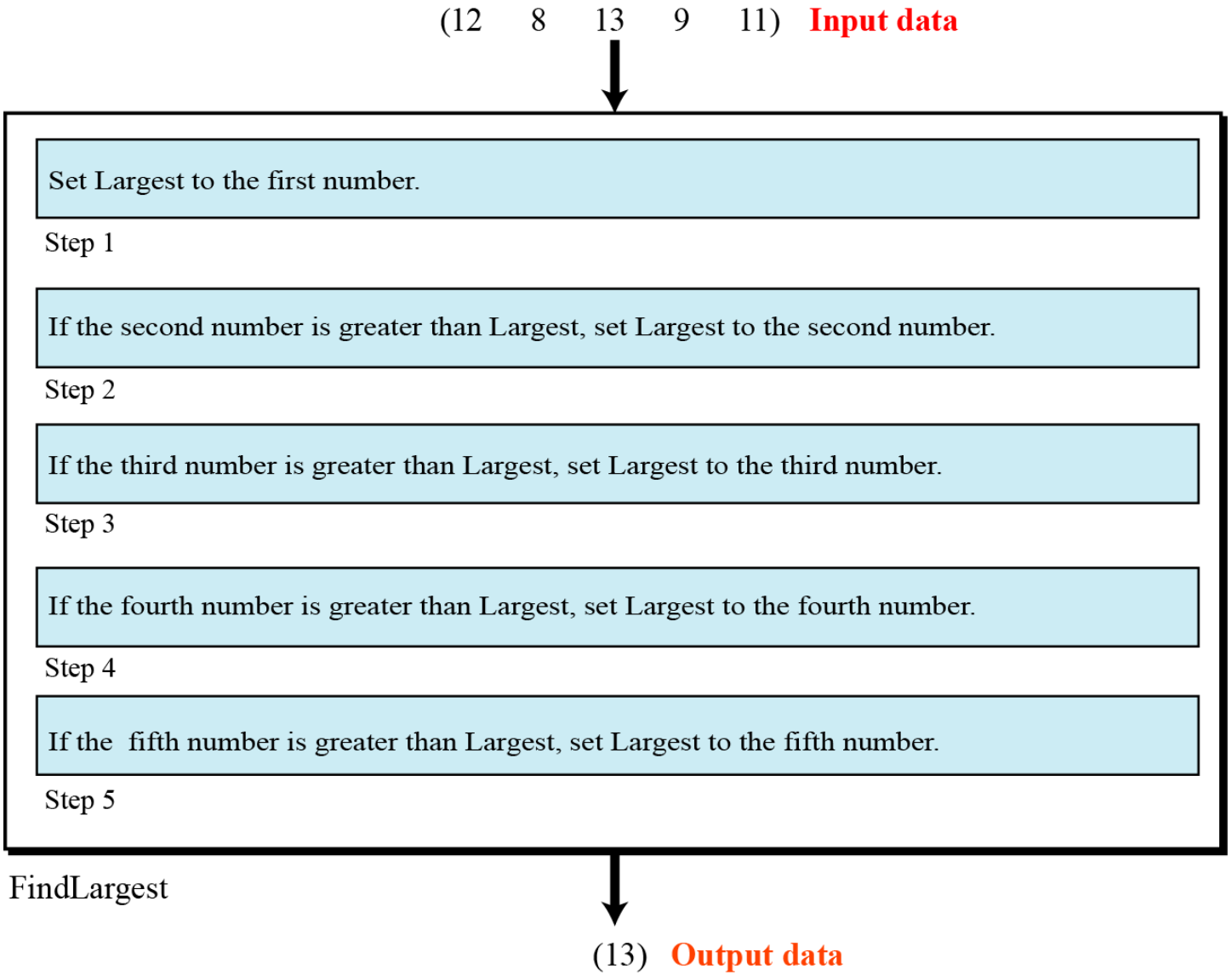


Figure 8.3 Defining actions in FindLargest algorithm

Refinement:

There are two problems:

- ✓ **First,** the action in the first step is different than those for the other steps.
- ✓ **Second,** the wording is not the same in steps 2 to 5.

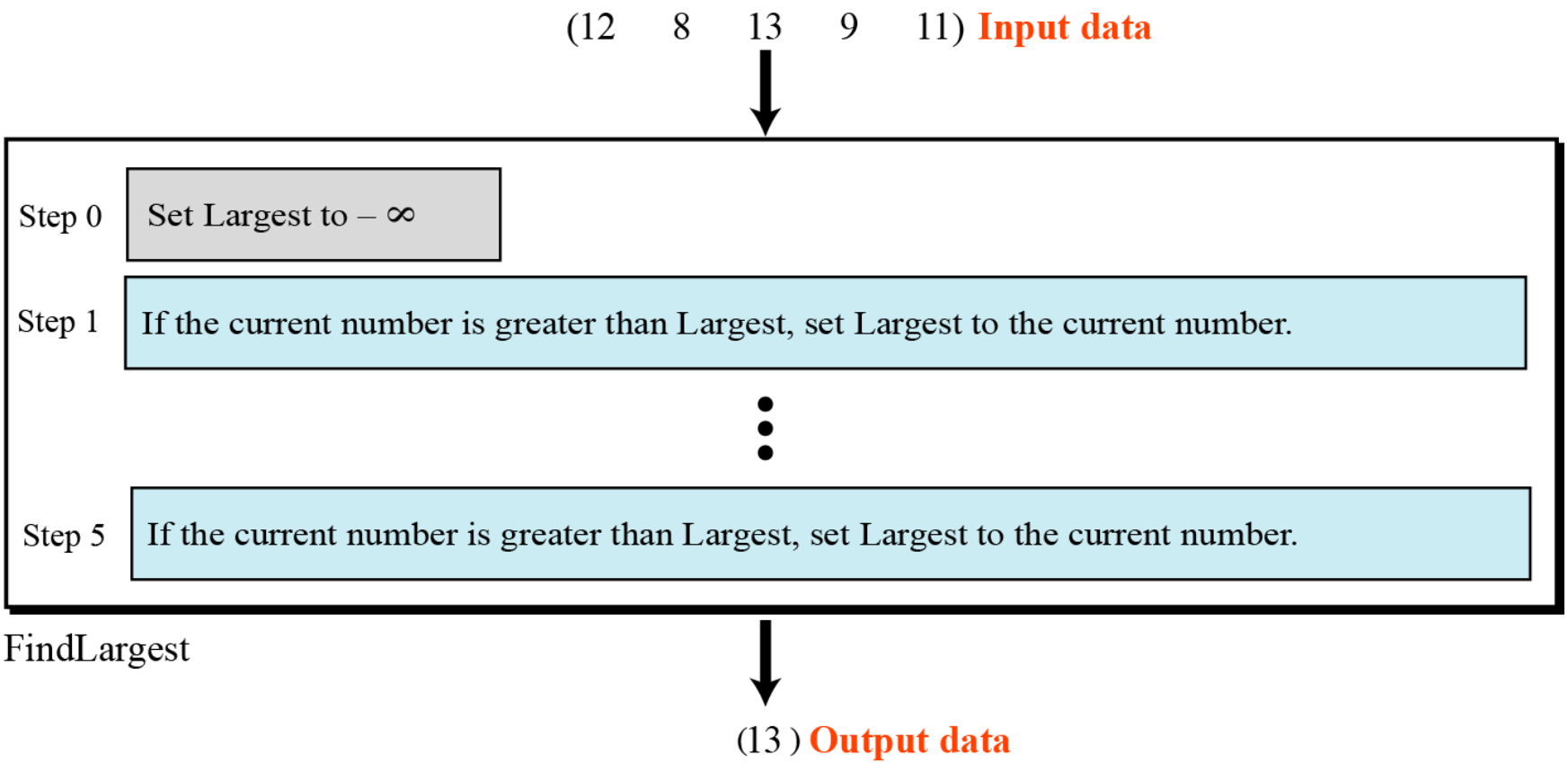


Figure 8.4 **FindLargest refined**

Generalization:

Is it possible to generalize the algorithm?

We want to find the largest of ***n*** positive integers, where *n* can be 1000, 1,000,000, or more.

There is a better way to do this. We can tell the computer to repeat the steps ***n*** times.

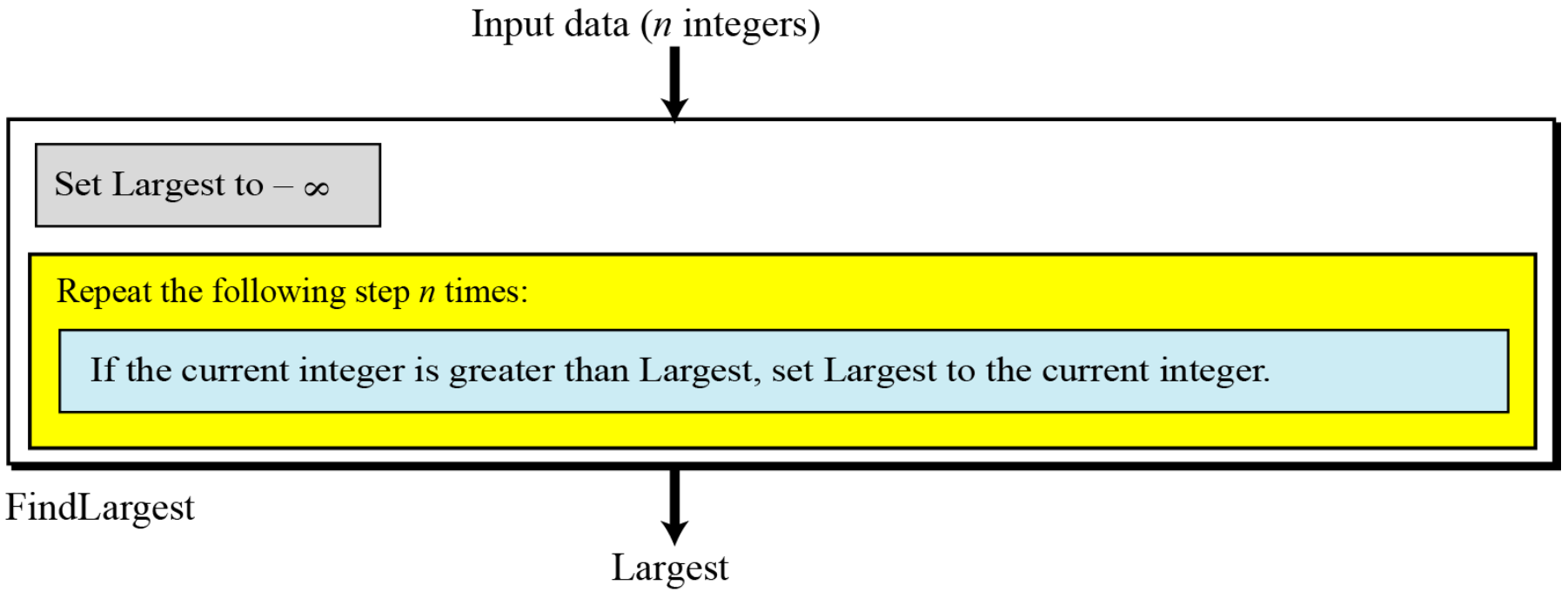


Figure 8.5 **Generalization of FindLargest**

8.2 THREE CONSTRUCTS

- Computer scientists have defined three constructs for a structured program or algorithm.
- The idea is that a program must be made of a combination of only these three constructs: *sequence*, *decision* (selection) and *repetition*.
- It has been proven there is no need for any other constructs. Using only these constructs makes a program or an algorithm easy to understand, debug or change.

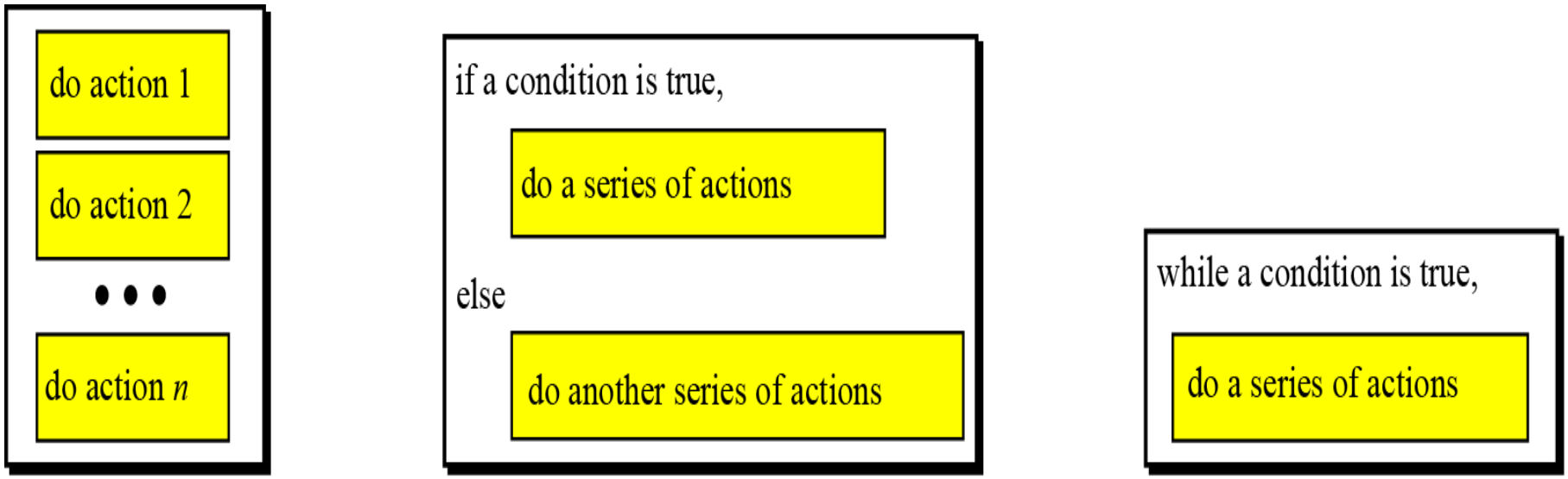


Figure 8.6 **Three constructs**

Sequence:

An algorithm, and eventually a program, is a sequence of instructions, which can be a simple instruction or either of the other two constructs.

Decision (selection):

Some problems cannot be solved with only a sequence of simple instructions. Sometimes we need to test a condition. If the result of testing is true, we follow a sequence of instructions. if it is false, we follow a different sequence of instructions. This is called the decision (selection) construct.

Repetition:

In some problems, the same sequence of instructions must be repeated. We handle this with the repetition or *loop* construct. Finding the largest integer among a set of integers can use a construct of this kind.

8.3 ALGORITHM REPRESENTATION

So far, we have used figures to convey the concept of an algorithm. During the last few decades, tools have been designed for this purpose. Two of these tools, UML and pseudocode, are presented here.

UML

Unified Modeling Language (UML)

- Is a pictorial representation of an algorithm.
- It hides all the details of an algorithm in an attempt to give the "big picture" and to show how the algorithm flows from beginning to end.

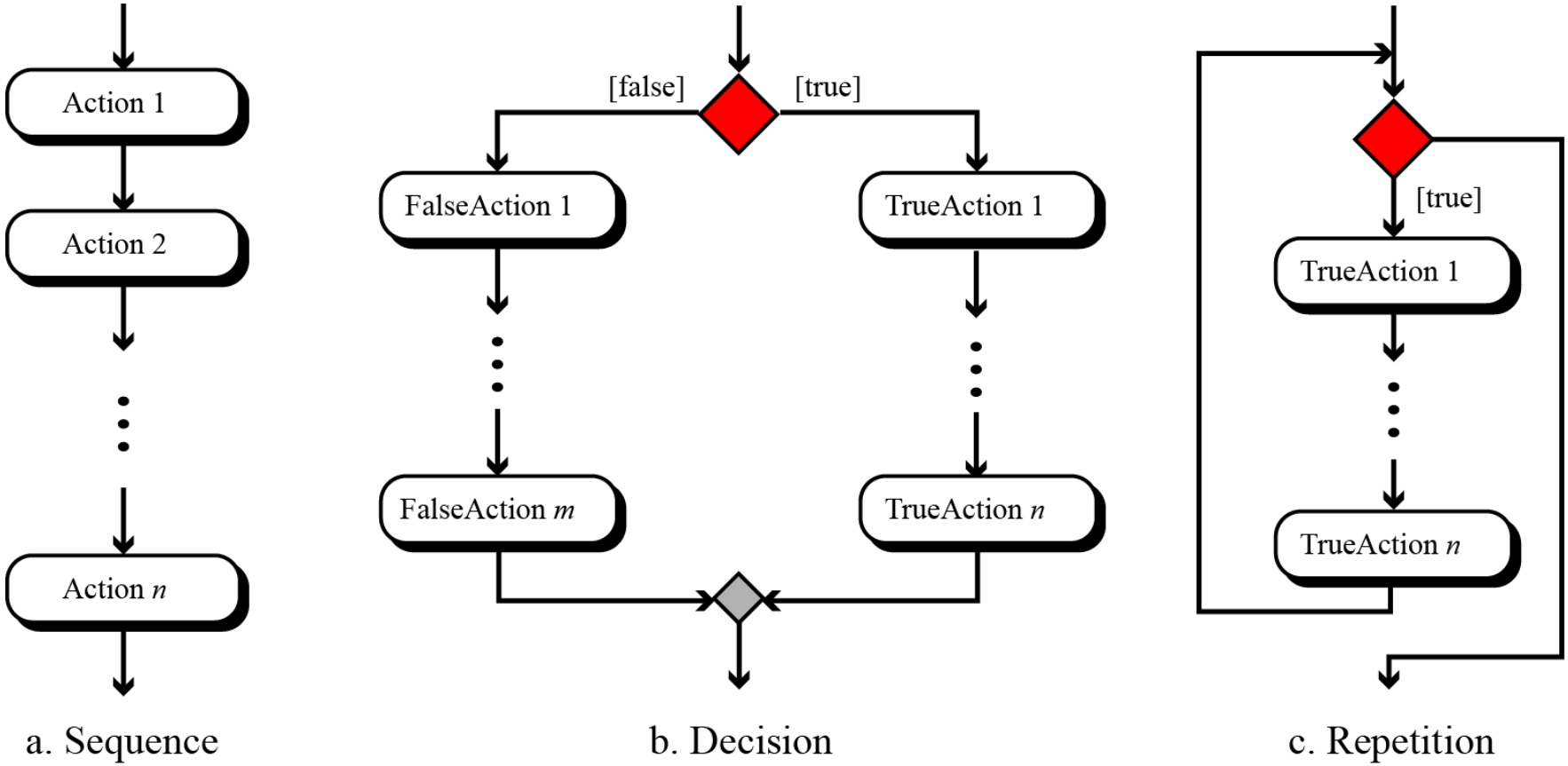


Figure 8.7 **UML for three constructs**

Pseudocode

- Pseudocode is an English-language-like representation of an algorithm.

- There is no standard for Pseudocode—some people use a lot of detail, others use less. Some use a code that is close to English, while others use a syntax like the Pascal programming language.

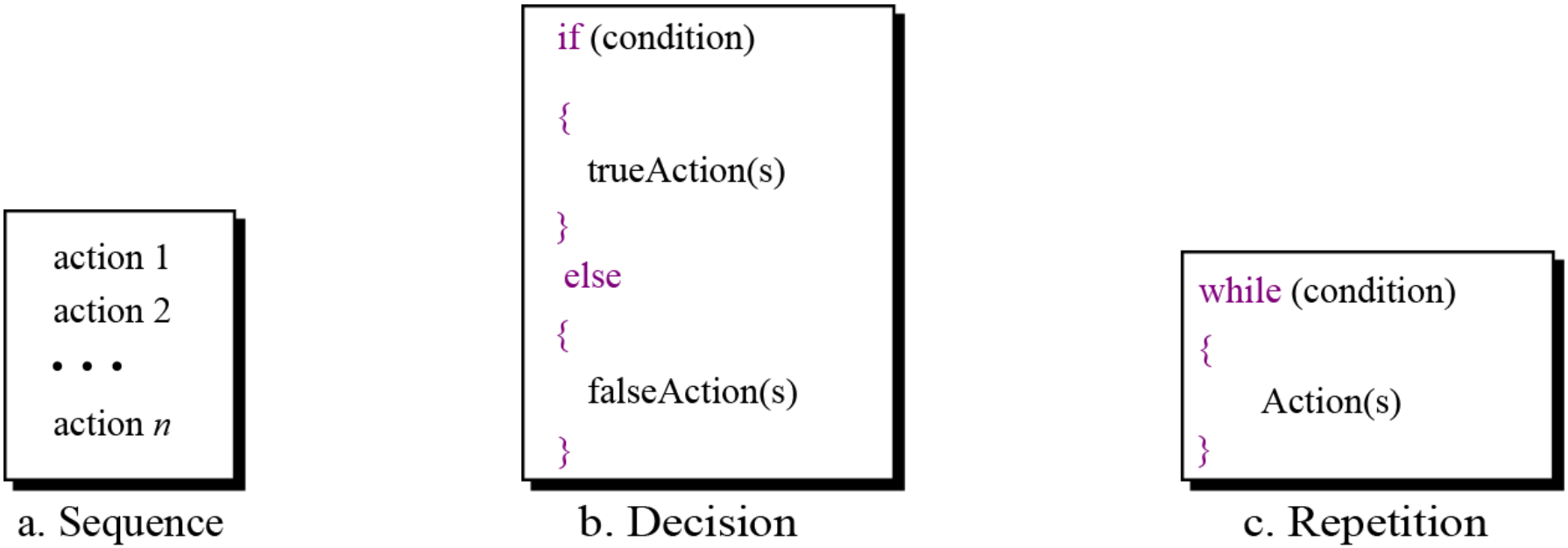


Figure 8.8 **Pseudocode for three constructs**

Example 8.1

- **Write an algorithm in Pseudocode that finds the sum of two integers.**

Example 8.1

Algorithm 8.1 Calculating the sum of two integers

```
Algorithm: SumOfTwo (first, second)
Purpose: Find the sum of two integers
Pre: Given: two integers (first and second)
Post: None
Return: The sum value
{
        sum ← first + second
        return sum
}
```

Example 8.2

- **Write an algorithm to change a numeric grade to a pass/no pass grade.**

Example 8.2

Algorithm 8.2 Assigning pass / no pass grade

```
Algorithm: Pass/NoPass (score)

Purpose: Creates a pass/no pass grade given the score

Pre: Given: the score to be changed to grade

Post: None

Return: The grade

{
        if (score ≥ 70)          grade ← "pass"
        else                     grade ← "nopass"
        return grade
}
```

Example 8.3

- **Write an algorithm to change a numeric grade (integer) to a letter grade.**

Example 8.3

Algorithm 8.3 Assigning a letter grade

Algorithm: **LetterGrade** (score)

Purpose: Find the letter grade corresponding to the given score

Pre: Given: a numeric score

Post: None

Return: A letter grade

```
{
        if (100 ≥ score ≥ 90)    grade ← 'A'
        if (80 ≥ score ≥ 89)     grade ← 'B'
        if (70 ≥ score ≥ 79)     grade ← 'C'
        if (60 ≥ score ≥ 69)     grade ← 'D'
        if (0 ≥ score ≥ 59)      grade ← 'F'
        return grade
}
```

Example 8.4

- **Write an algorithm to find the largest of a set of integers. We do not know the number of integers.**

Example 8.4

Algorithm 8.4 Finding the largest integer among a set of integers

```
Algorithm: FindLargest (list)
Purpose: Find the largest integer among a set of integers
Pre: Given: the set of integers
Post: None
Return: The largest integer
{
        largest ← – ∞
        while (more integers to check)
        {
                current ← next integer
                if (current > largest)          largest ← current
        }
        return largest
}
```

Example 8.5

- **Write an algorithm to find the largest of the first 1000 integers in a set of integers.**

Example 8.5

Algorithm 8.5 Finding the largest integer among the first 1000 integers

```
Algorithm: FindLargest2 (list)
Purpose: Find and return the largest integer among the first 1000 integers
Pre: Given: the set of integers with more than 1000 integers
Post: None
Return: The largest integer
{
        largest ← – ∞
        counter ← 1
        while (counter ≤ 1000)
        {
                current ← next integer
                if (current > largest)          largest ← current
                counter ← counter + 1
        }
        return largest
}
```

8.4 A MORE FORMAL DEFINITION

Now that we have discussed the concept of an algorithm and shown its representation, here is a more formal definition.

Algorithm:

An ordered set of unambiguous steps that produces a result and terminates in a finite time.

- Ordered set:

An algorithm must be a well-defined.

- Unambiguous steps:

Each step in an algorithm must be clearly and unambiguously defined.

- Produce a result:

An algorithm must produce a result, otherwise it is useless. The result can be data returned to the calling algorithm, or some other effect (for example, printing).

- Terminate in a finite time:

An algorithm must terminate (halt). If it does not (that is, it has an infinite loop), we have not created an algorithm.

8.5 BASIC ALGORITHMS

- ✓ Summation.
- ✓ Product.
- ✓ Smallest and largest.
- ✓ Sorting.

Summation

We can add two or three integers very easily, but how can we add many integers? The solution is simple: we use the add operator in a loop.

A summation algorithm has three logical parts:

1. Initialization of the sum at the beginning.
2. The loop, which in each iteration adds a new integer to the sum.
3. Return of the result after exiting from the loop.

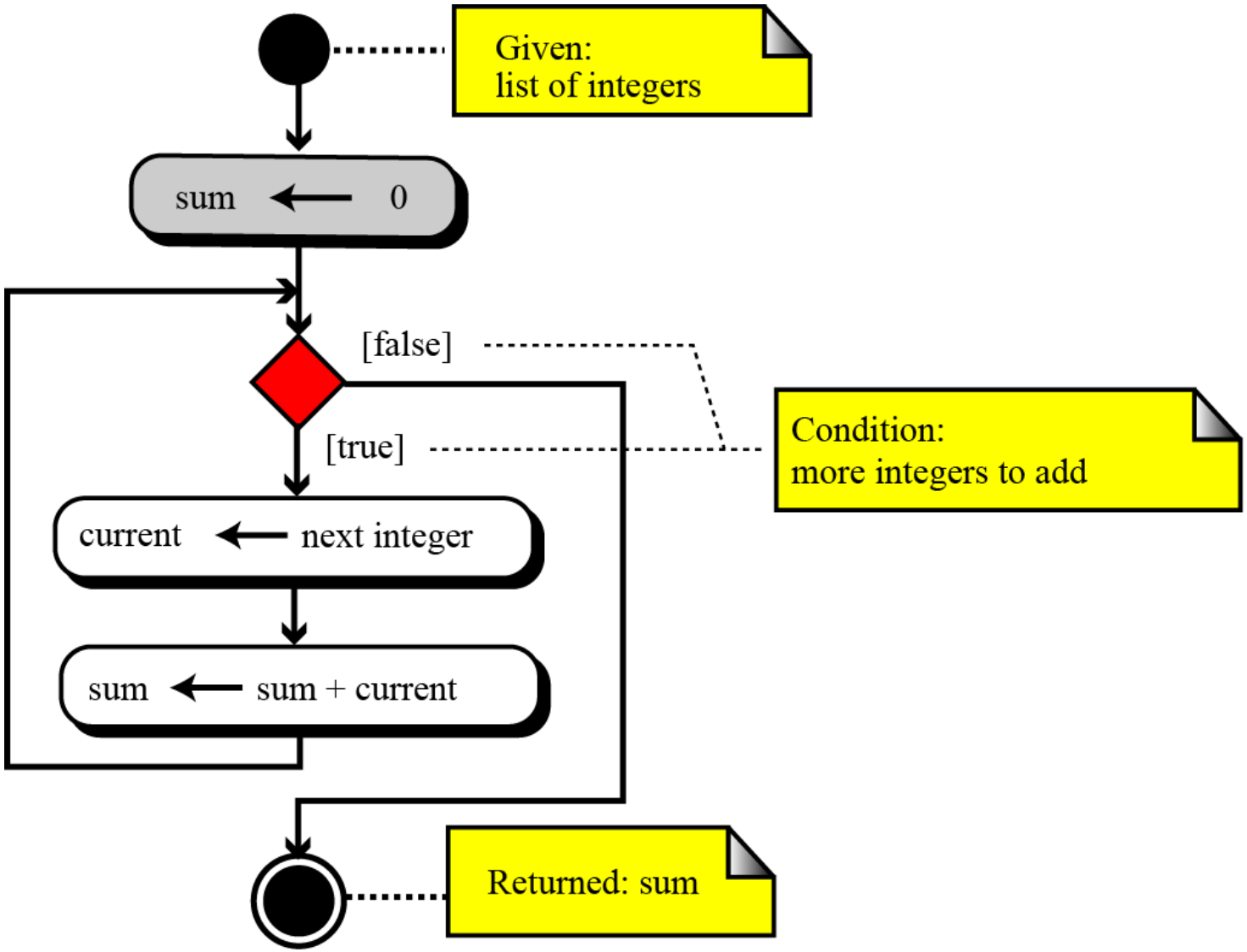


Figure 8.9 Summation algorithm

Product

Another common algorithm is finding the product of a list of integers. The solution is simple: use the multiplication operator in a loop.

A product algorithm has three logical parts:

1. Initialization of the product at the beginning.
2. The loop, which in each iteration multiplies a new integer with the product.
3. Return of the result after exiting from the loop.

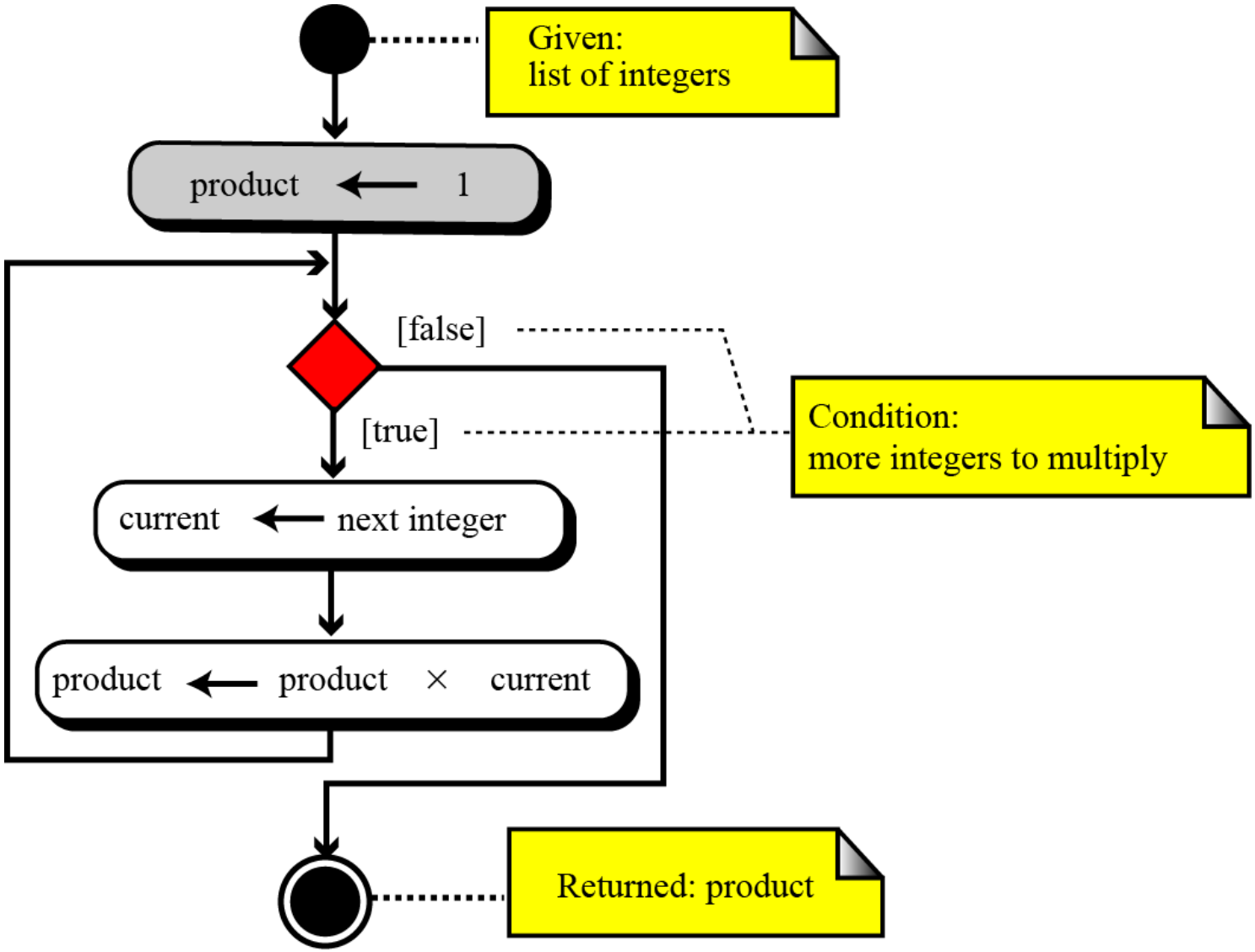


Figure 8.10 **Product algorithm**

Smallest and largest

We discussed the algorithm for finding the largest among a list of integers at the beginning of this chapter. The idea was to write a decision construct to find the larger of two integers. If we put this construct in a loop, we can find the largest of a list of integers.

Finding the smallest integer among a list of integers is similar, with two minor differences. First, we use a decision construct to find the smaller of two integers. Second, we initialize with a very large integer instead of a very small one.

Sorting

One of the most common applications in computer science is sorting, which is the process by which data is arranged according to its values.

- In this section, we introduce three sorting algorithms:

selection sort, bubble sort and insertion sort.

These three sorting algorithms are the foundation of faster sorting algorithms used in computer science today.

• Selection sorts

In a selection sort, the list to be sorted is divided into two sublists—sorted and unsorted—which are separated by an imaginary wall. We find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted sublist. After each selection and swap, the imaginary wall between the two sublists moves one element ahead.

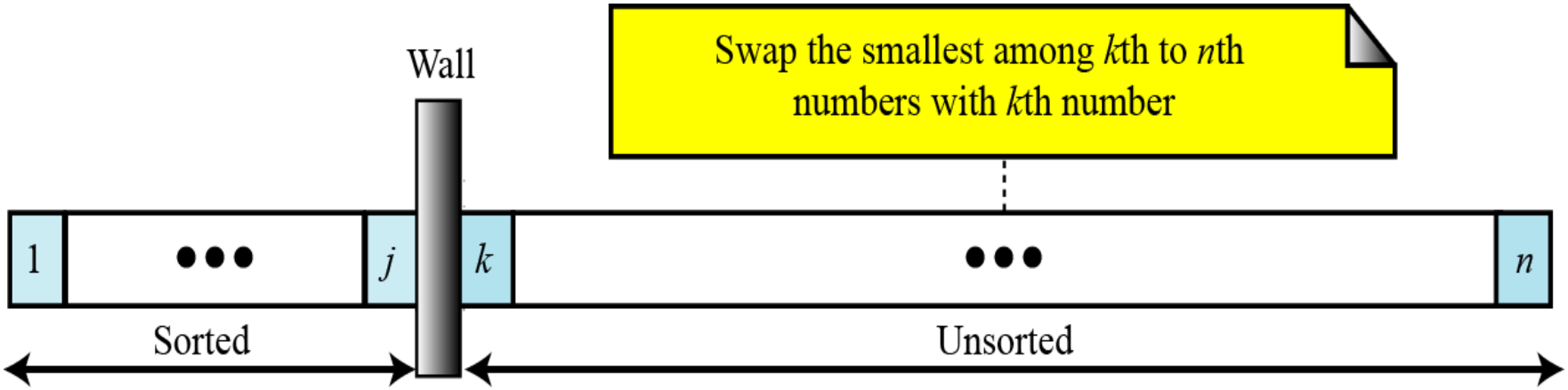


Figure 8.11 **Selection sort**

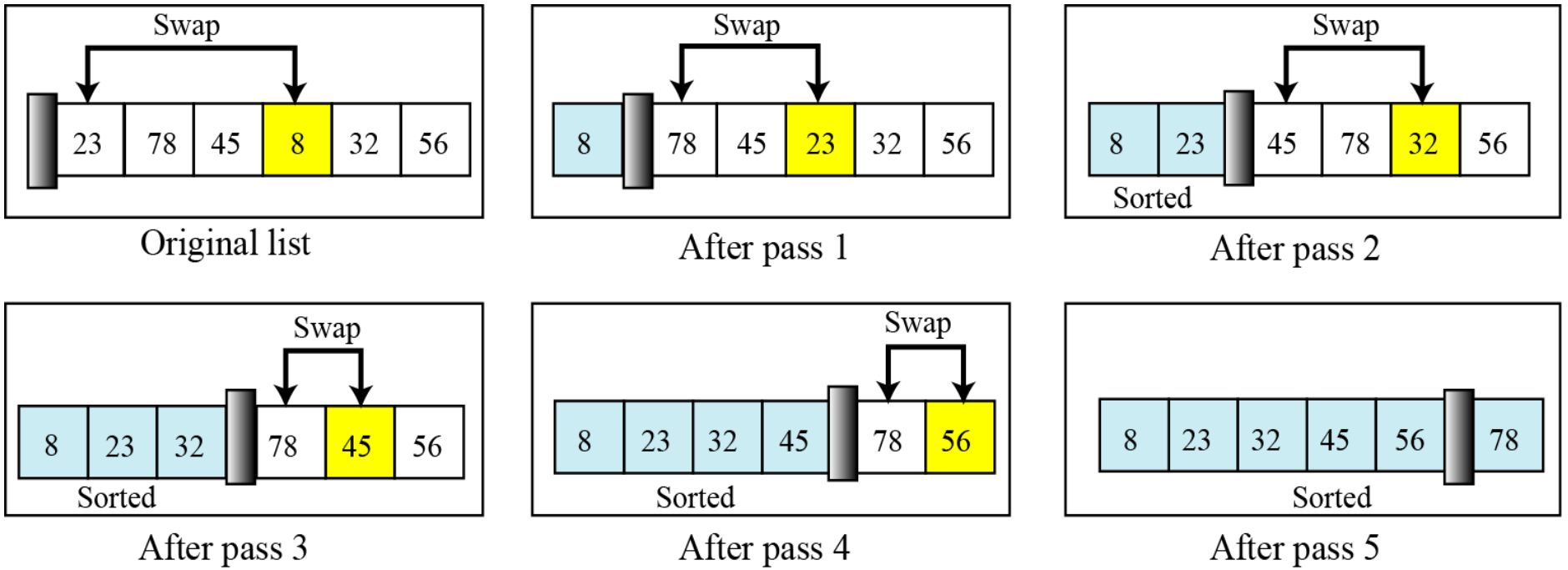


Figure 8.12 **Example of selection sort**

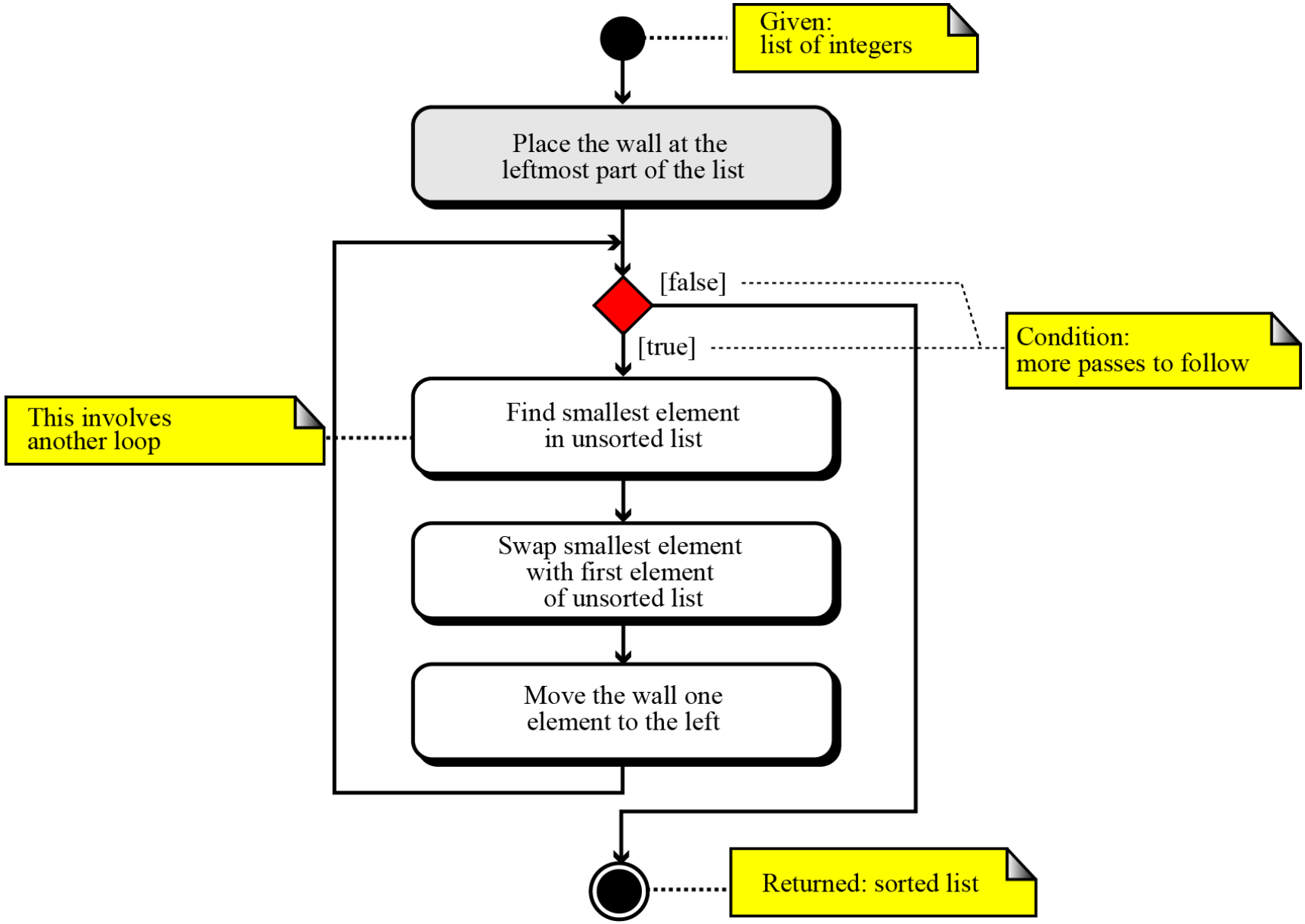


Figure 8.13 Selection sort algorithm

- ***Bubble sorts***

In the bubble sort method, the list to be sorted is also divided into two sublists—sorted and unsorted. The smallest element is bubbled up from the unsorted sublist and moved to the sorted sublist. After the smallest element has been moved to the sorted list, the wall moves one element ahead.

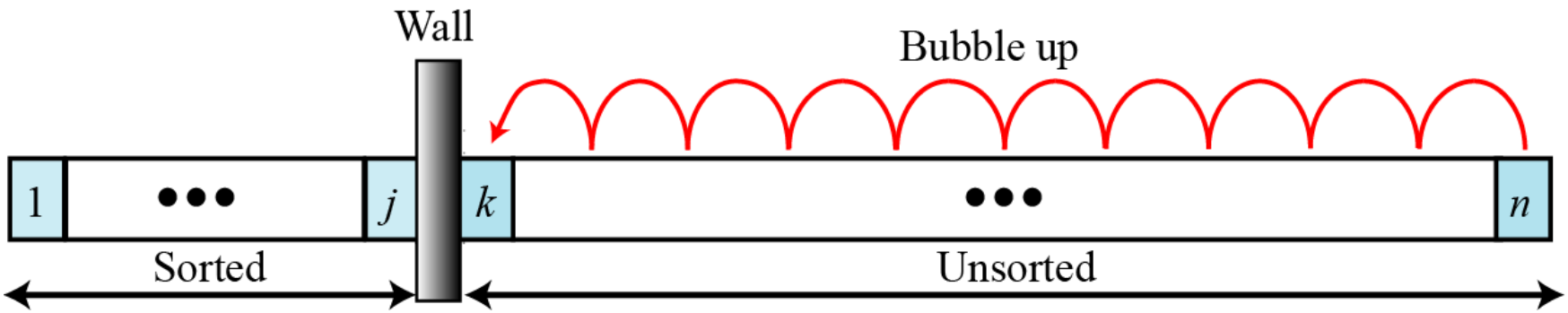


Figure 8.14 **Bubble sort**

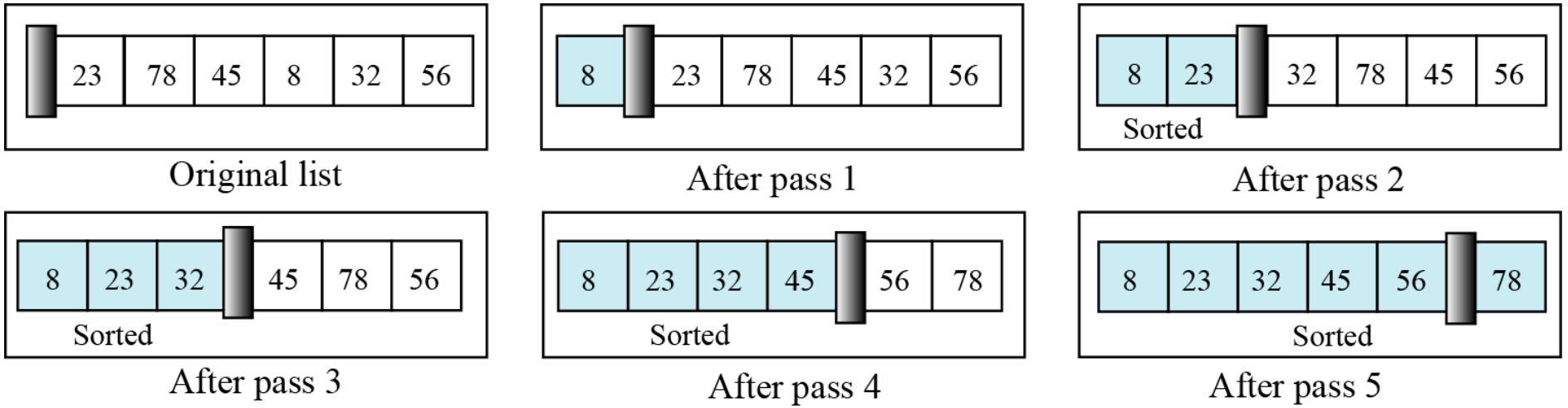


Figure 8.15 **Example of bubble sort**

• Insertion sorts

The insertion sort algorithm is one of the most common sorting techniques, and it is often used by card players. Each card a player picks up is inserted into the proper place in their hand of cards to maintain a particular sequence.

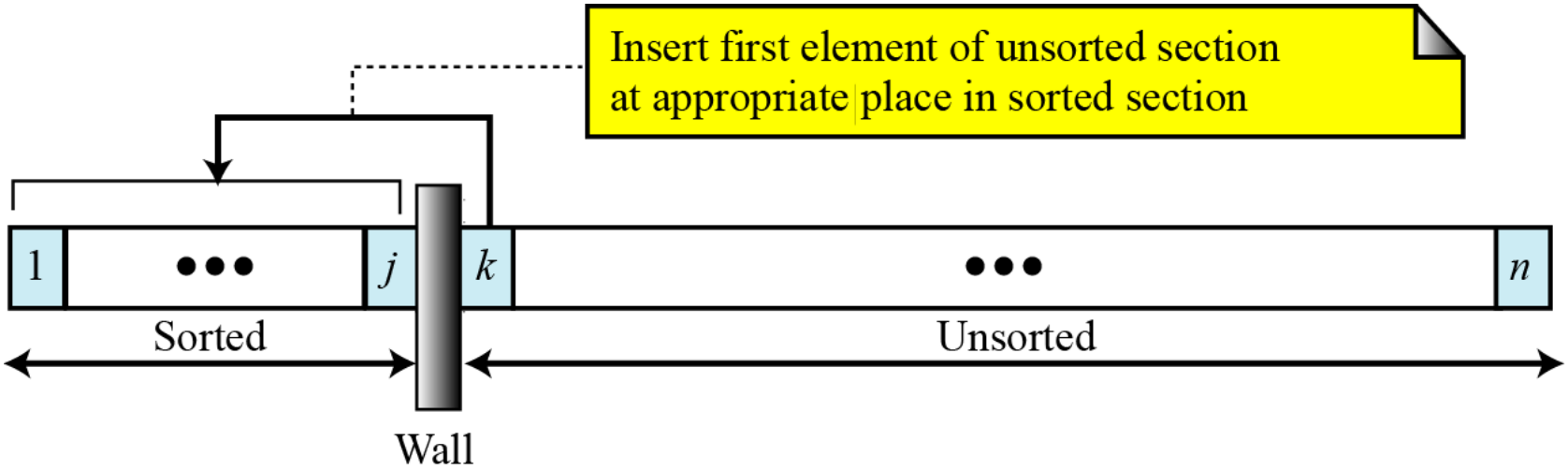


Figure 8.16 **Insertion sort**

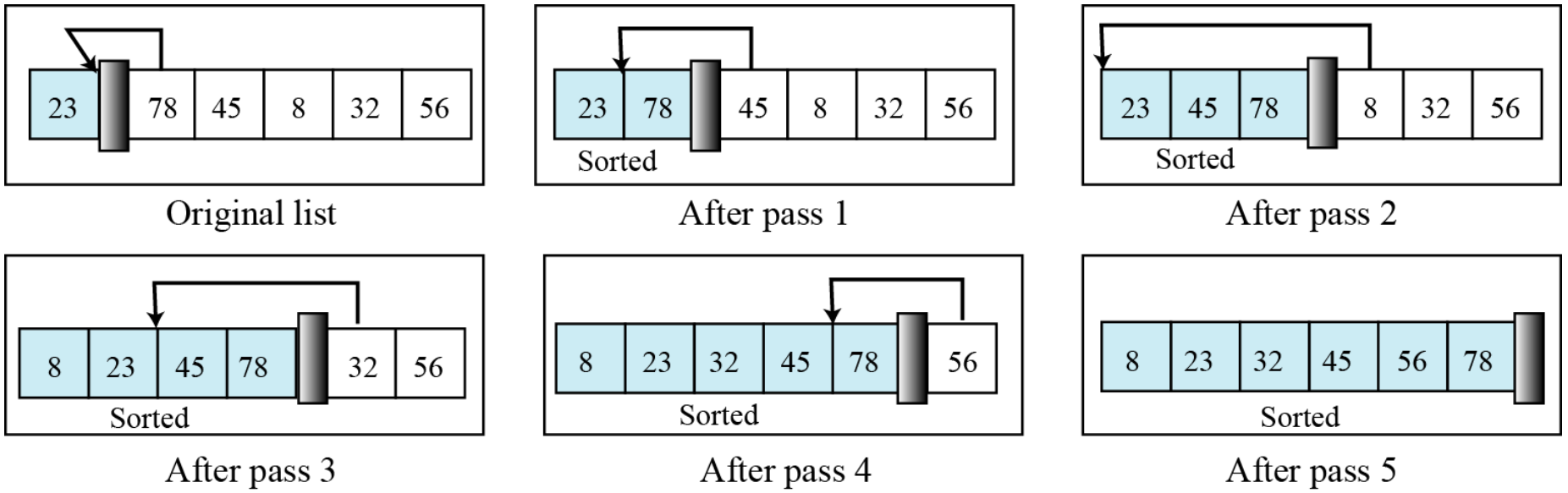


Figure 8.17 **Example of insertion sort**

Searching

Another common algorithm in computer science is searching, which is the process of finding the location of a target among a list of objects.
In the case of a list, searching means that given a value, we want to find the location of the first element in the list that contains that value.

There are two basic searches for lists:
sequential search and binary search.

Sequential search can be used to locate an item in any list, whereas **binary** search requires the list first to be sorted.

• ***Sequential search***

- ✓ Sequential search is used if the list to be searched is not ordered.
- ✓ We use this technique only for small lists, or lists that are not searched often.
- ✓ In other cases, the best approach is to first sort the list and then search it using the binary search discussed later.

In a sequential search, we start searching for the target from the beginning of the list. We continue until we either find the target or reach the end of the list.

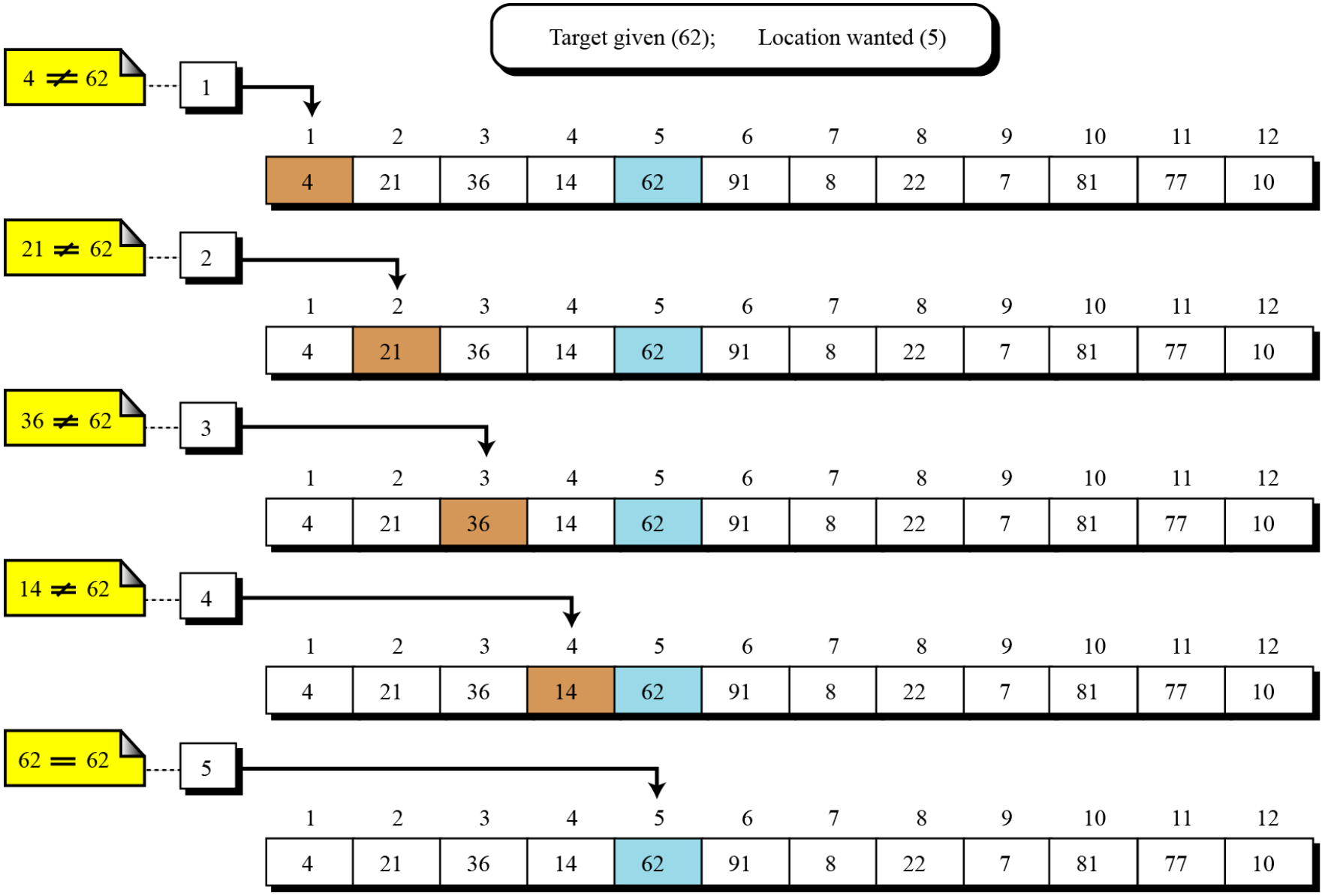


Figure 8.18 An example of a sequential search

- ***Binary search:***

✓ The sequential search algorithm is very slow.

✓ If the list is sorted, however, we can use a more efficient algorithm called binary search. Generally speaking, programmers use a binary search when a list is large.

A binary search starts by testing the data in the element at the middle of the list. This determines whether the target is in the first half or the second half of the list. If it is in the first half, there is no need to further check the second half. If it is in the second half, there is no need to further check the first half. In other words, we eliminate half the list from further consideration.

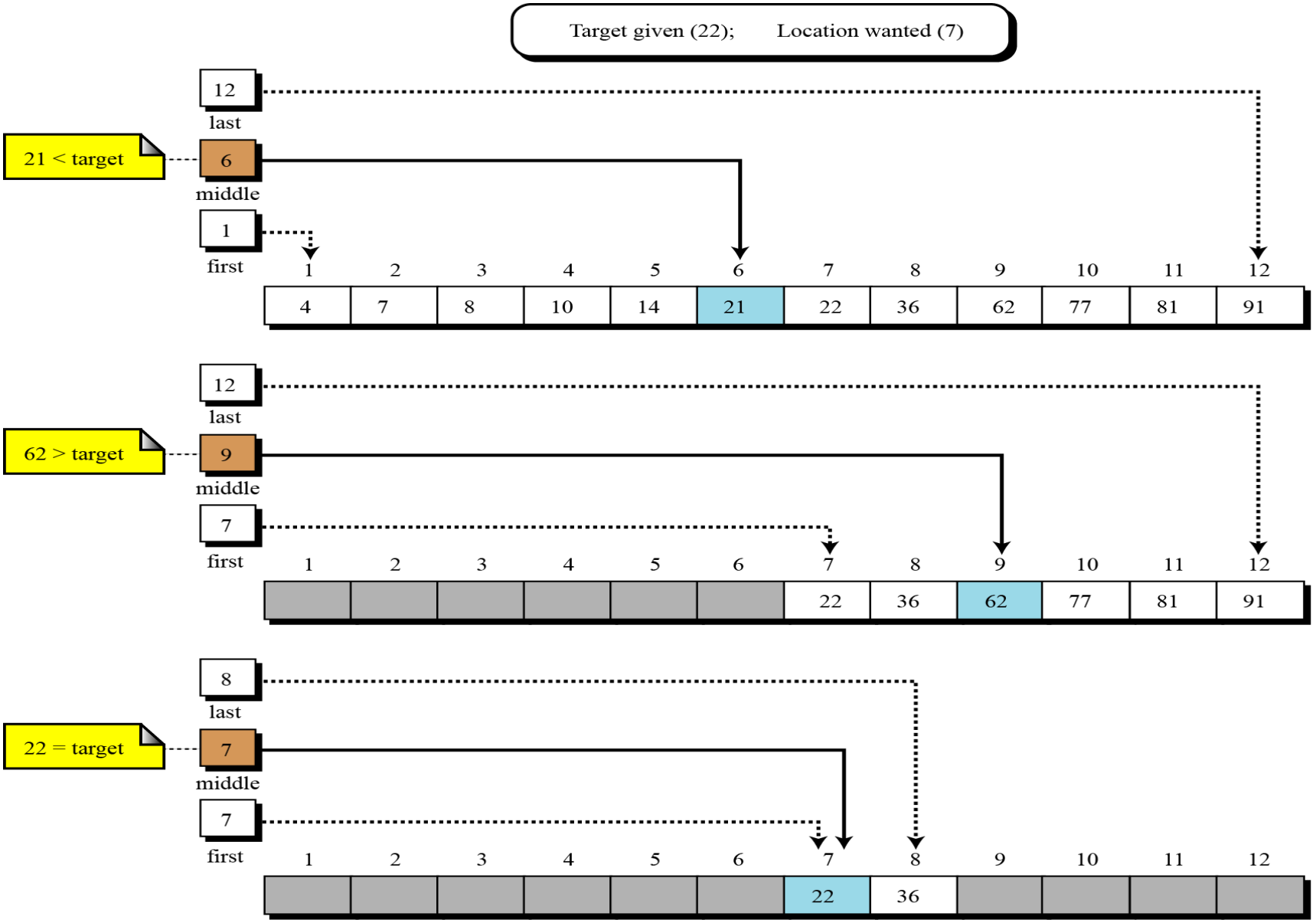


Figure 8.19 Example of a binary search