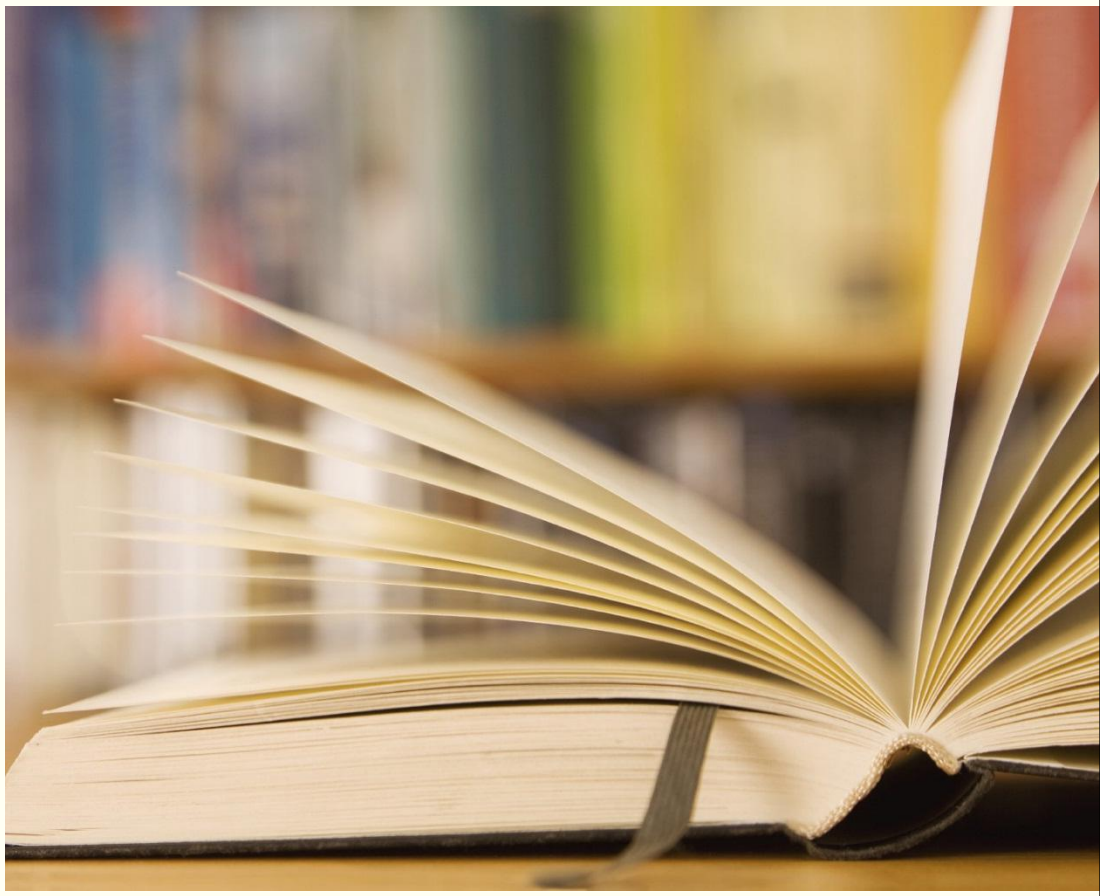
6. ALGORITHMS



Content

- 6.1 Concepts: input, output, processing
- 6.2 Three basic constructs
- 6.3 Algorithm representation
- 6.4 Search Algorithms: linear, binary

Objectives

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

- Define an algorithm and relate it to problem solving.
- Define three constructs—sequence, selection, and repetition—and describe their use in algorithms.
- Describe UML diagrams and how they can be used when representing algorithms.
- Describe pseudocode and how it can be used when representing 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.

1-CONCEPTS: INPUT, OUTPUT, PROCESSING

1. Informal definition

- An informal definition of an algorithm is:

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

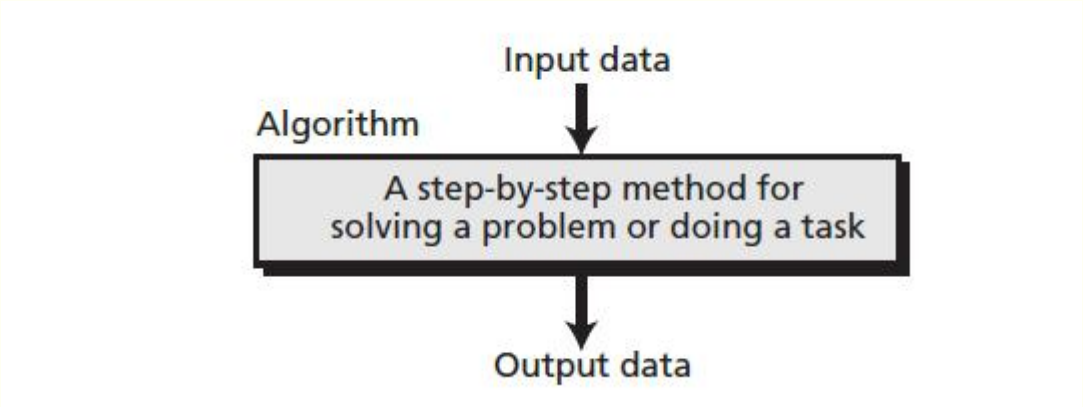


Figure 6.1 Informal definition of an algorithm used in a computer

Example

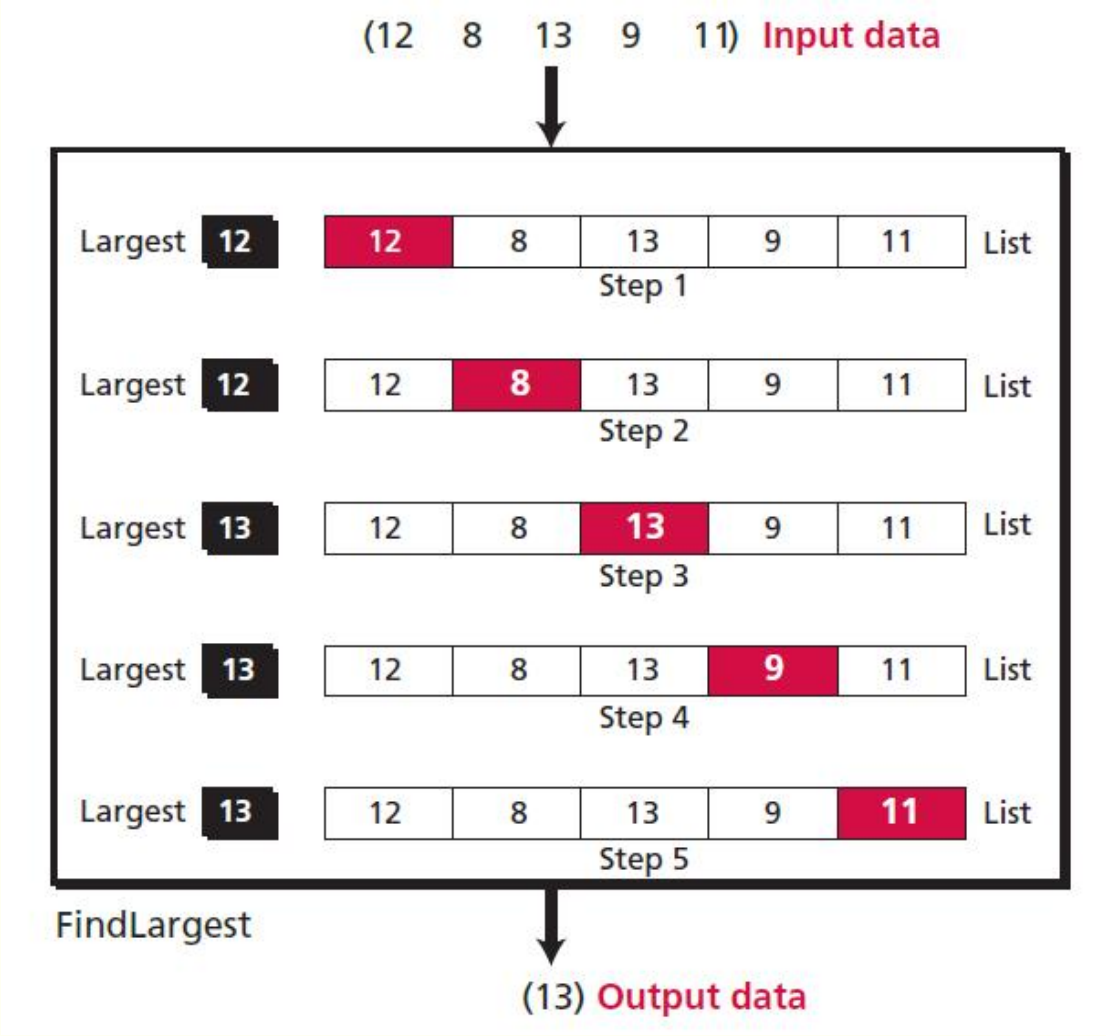
Problem:

- Develop an algorithm for finding the largest integer among a list of positive integers (for example 12, 8, 13, 9, 11). The algorithm should be general and not depend on the number of integers.

Solution :

- ***Step 1*** In this step, the algorithm inspects the first integer (12). The algorithm defines a data item, called Largest, and sets its value to the first integer (12).
- ***Step 2*** The algorithm makes a comparison between the value of Largest (12) and the value of the second integer (8). It finds that Largest is larger than the second integer, which means that Largest is still holding the largest integer.
- ***Step 3*** The largest integer so far is 12, but the new integer (13) is larger than Largest. The value of Largest should be replaced by the third integer (13). The algorithm changes the value of Largest to 13 and moves to the next step.
- ***Step 4*** Nothing is changed in this step because Largest is larger than the fourth integer (9).

Example (flow)



2. Refinement

This algorithm needs refinement to be acceptable to the programming community. 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.

Redefine the algorithm to remove these two inconveniences by changing the wording in steps 2 to 5 to 'If the current integer is greater than Largest, set Largest to the current integer'.

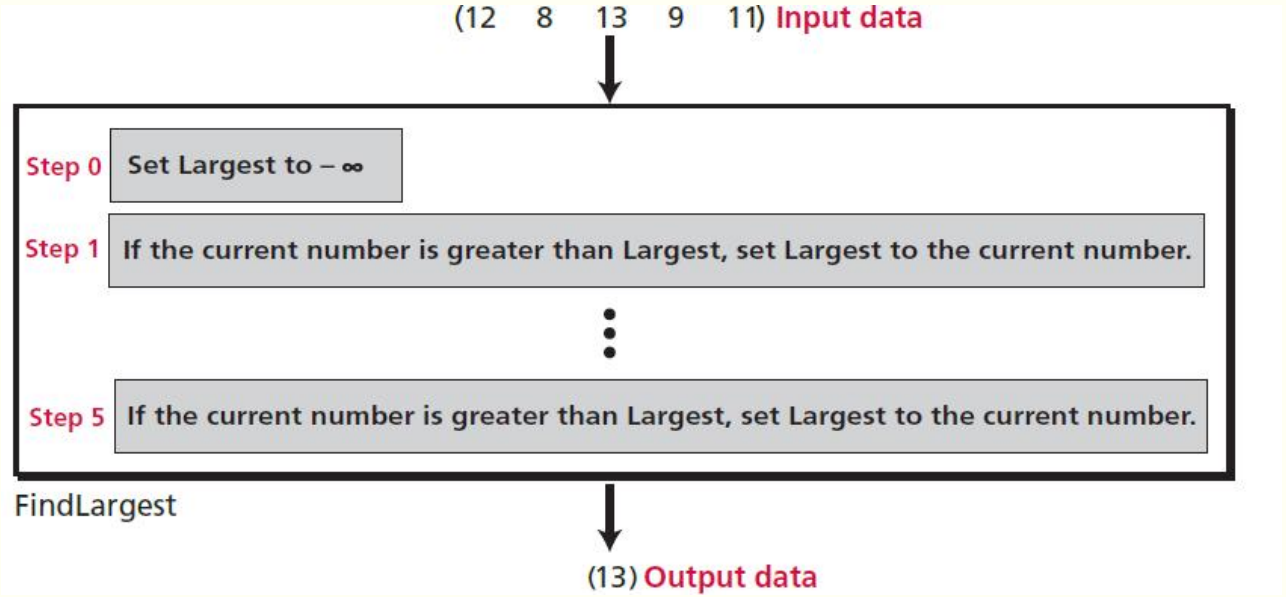


Figure 6.2 *Find largest refined*

3. 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.

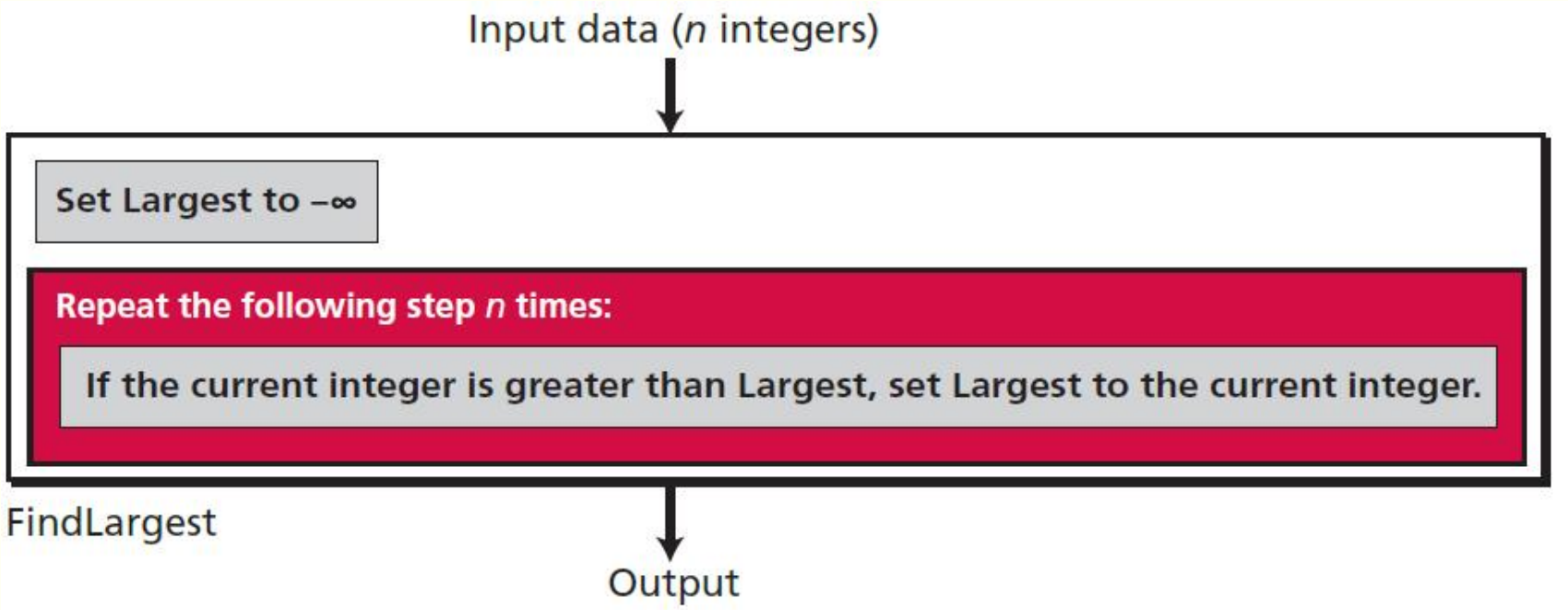


Figure 6.3 *Generalization of Find Largest*

2 - THREE BASIC CONSTRUCTS

Introduction

- 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***, and *repetition* (Figure 8.6).
- 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.

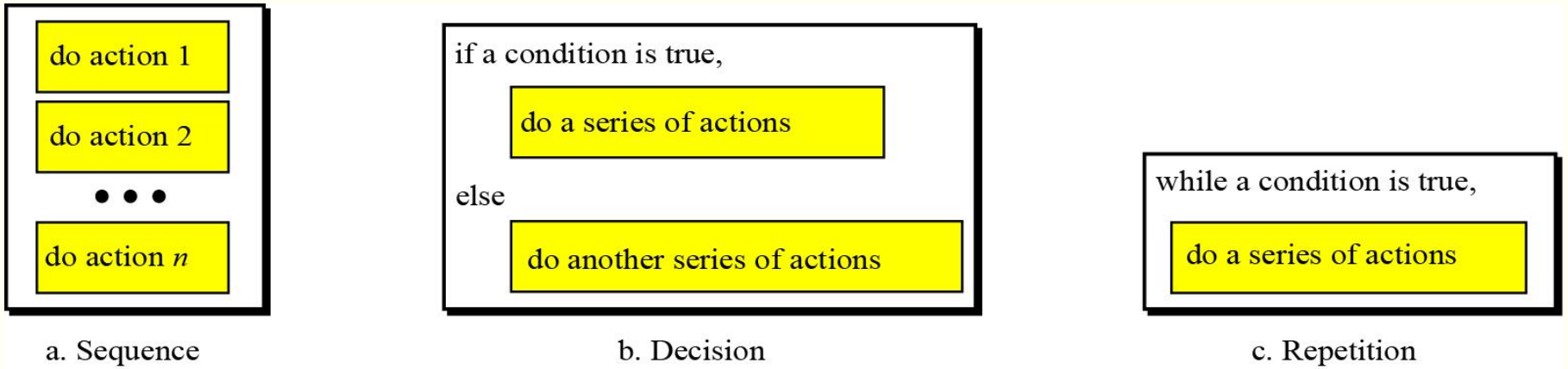


Figure 6.4 Three constructs

Sequence

- The first construct is called the **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.

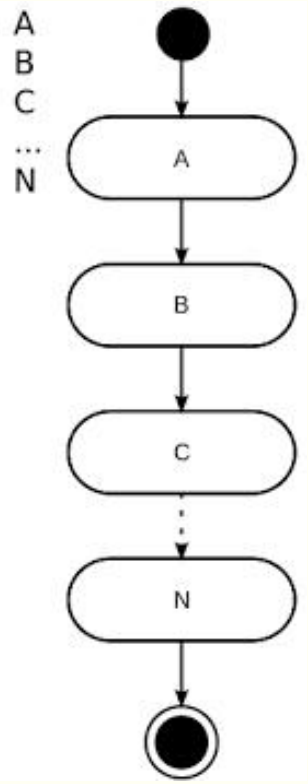


Figure 6.5 Sequence construct

Decision

- 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.

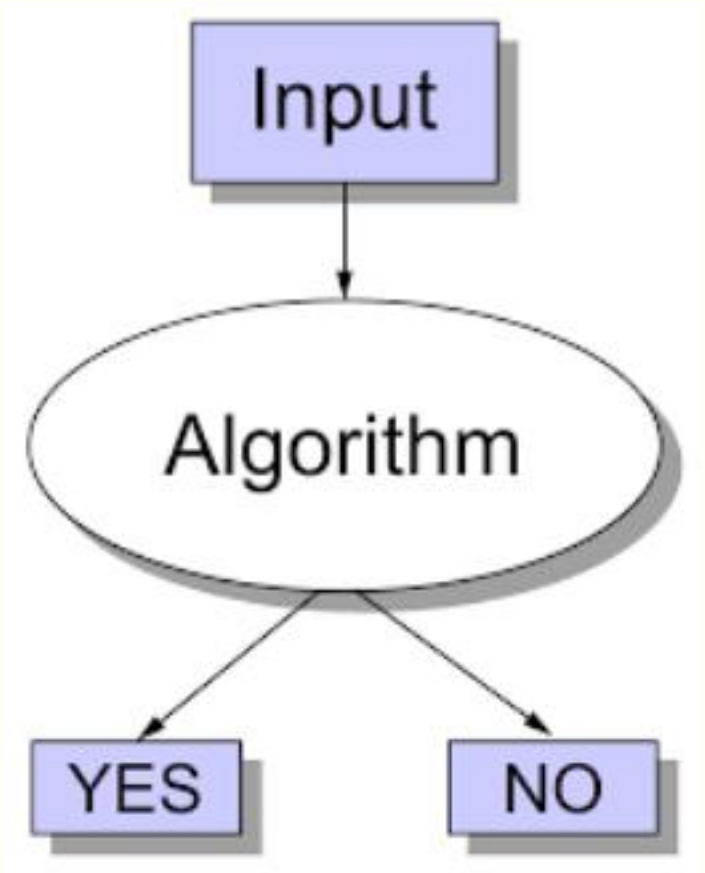


Figure 6.6 Decision construct

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.

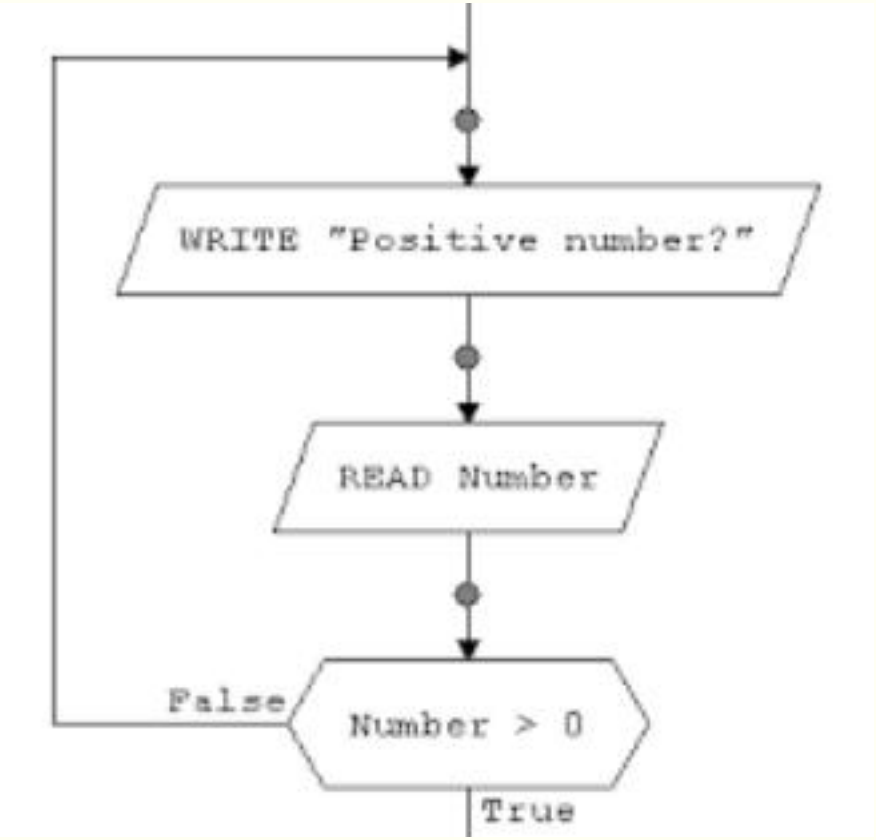


Figure 6.7 Repetition construct

3 - ALGORITHM REPRESENTATION

1. 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.

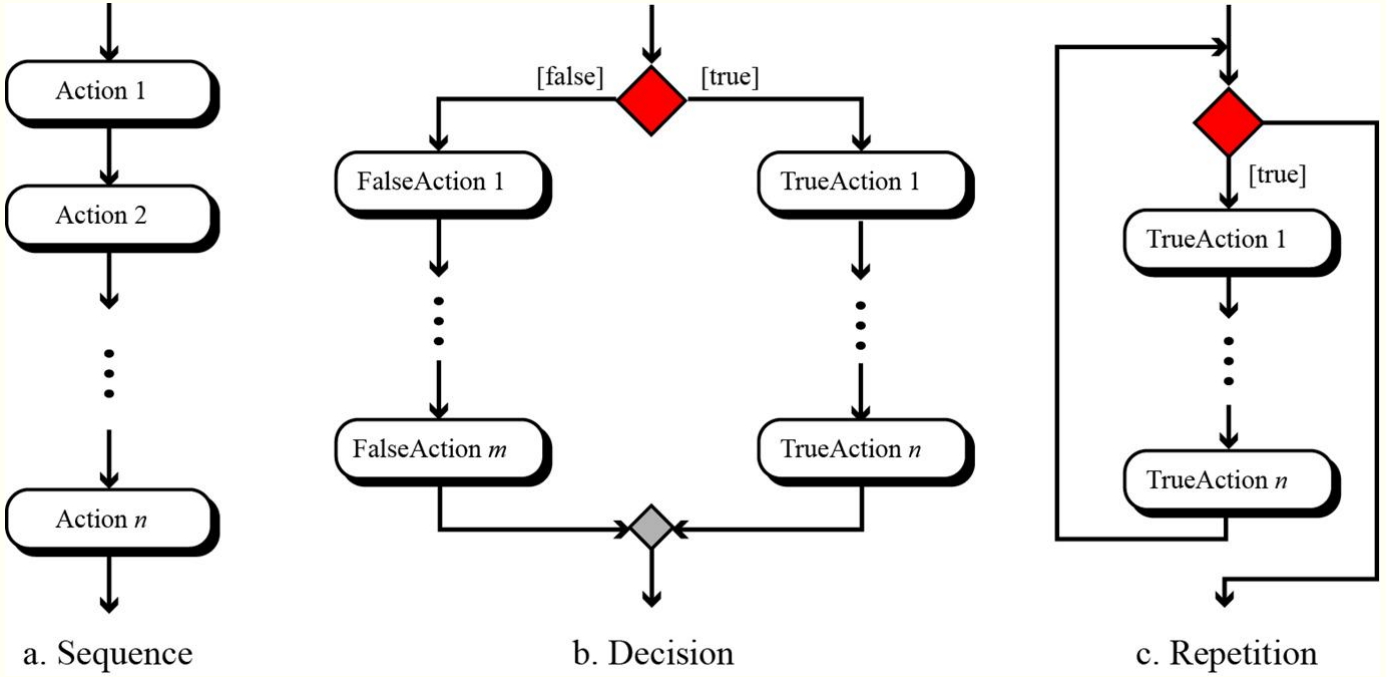


Figure 6.8 *UML for three constructs*

2. 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

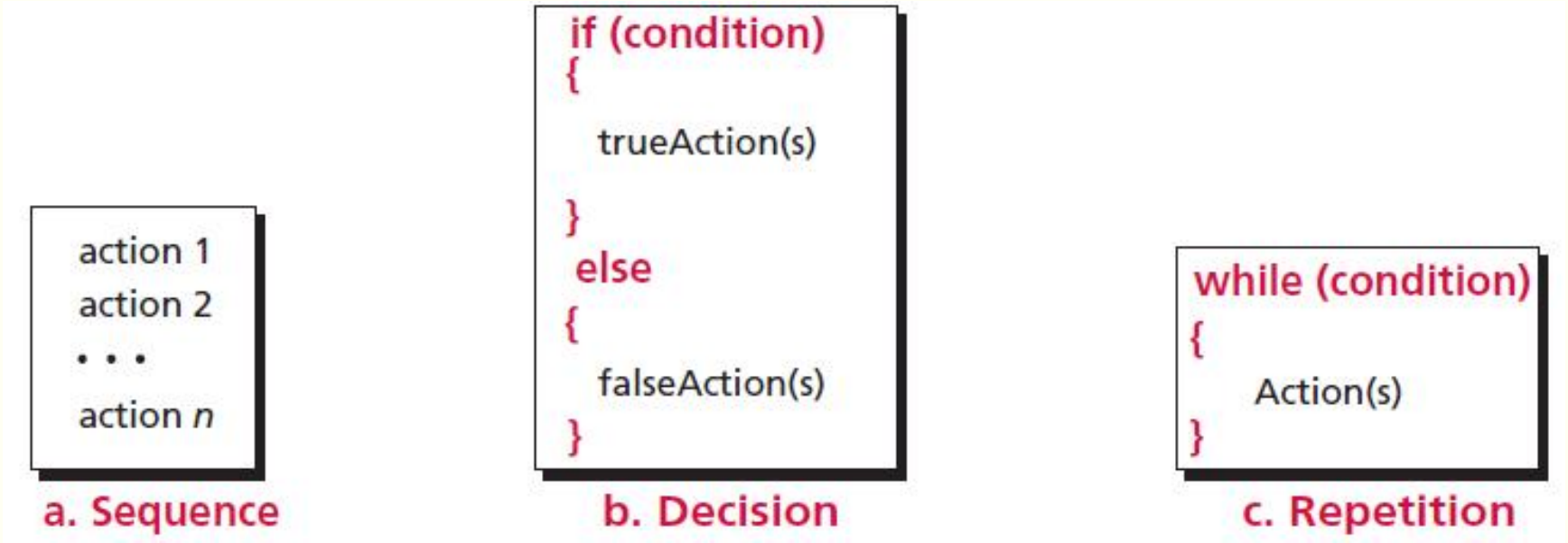


Figure 6.9 *Pseudocode for three constructs*

3. Example

- **Problem** Write an algorithm in pseudocode that finds the sum of two integers.
- ***Solution*** This is a simple problem that can be solved using only the sequence construct. Note also that we name the algorithm, define the input to the algorithm and, at the end, we use a return instruction to return the sum.

```
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
}
```

Figure 6.10 *Pseudocode for Calculating the sum of two integers*

4. BASIC ALGORITHMS

4.1 Summation

- One commonly used algorithm in computer science is **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.

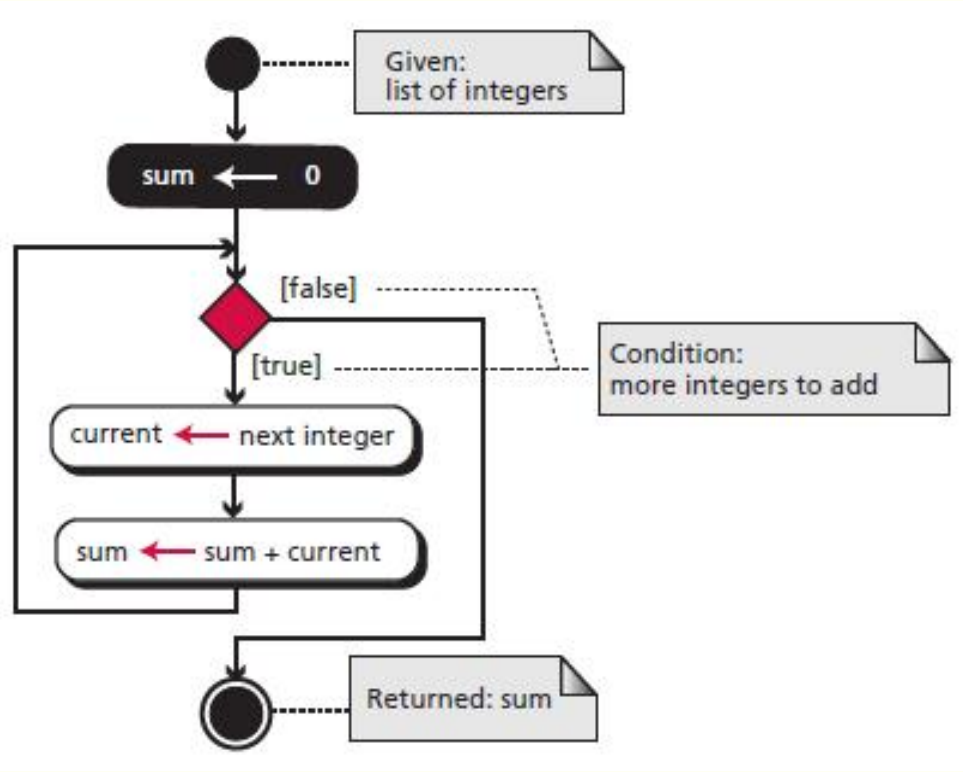


Figure 6.11 UML for Calculating the sum of two integers

2. 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.

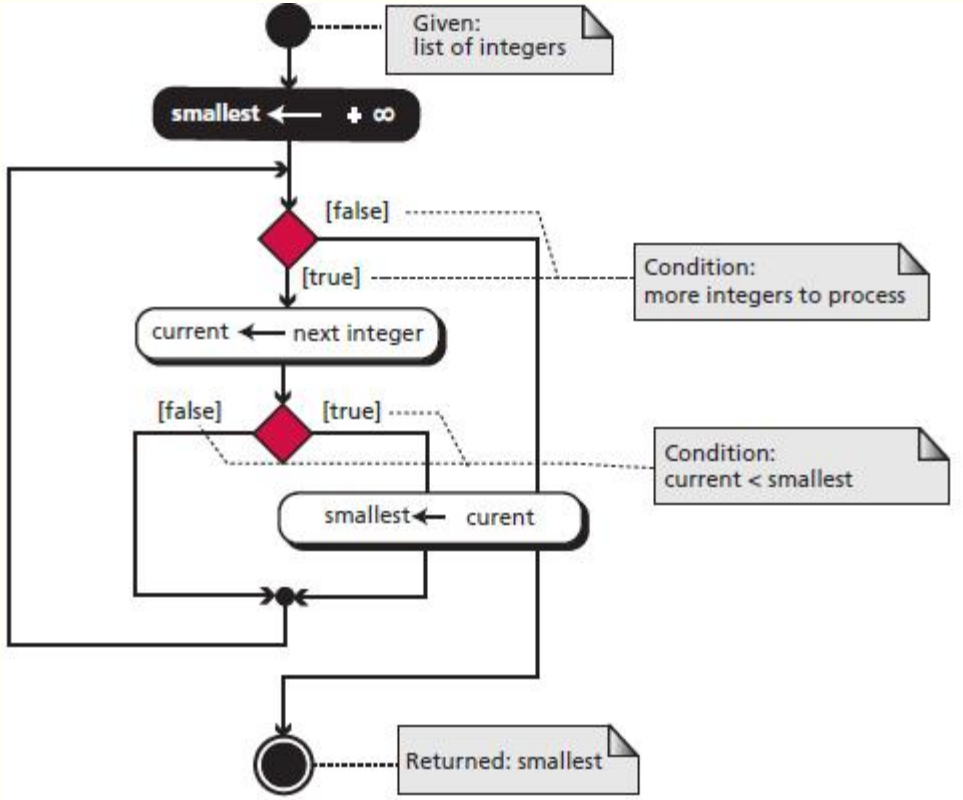


Figure 6.12 *UML for* Finding the smallest data item

3. 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. People are surrounded by data.
- If the data was not ordered, it would take hours and hours to find a single piece of information. Imagine the difficulty of finding someone's telephone number in a telephone book that is not ordered

Common sorting algorithms

- ❑Bubble/Shell sort:
- ❑Insertion sort
- ❑Selection sort
- ❑Quick sort
- ❑Merge sort

Selection Sort

- 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, increasing the number of sorted elements and

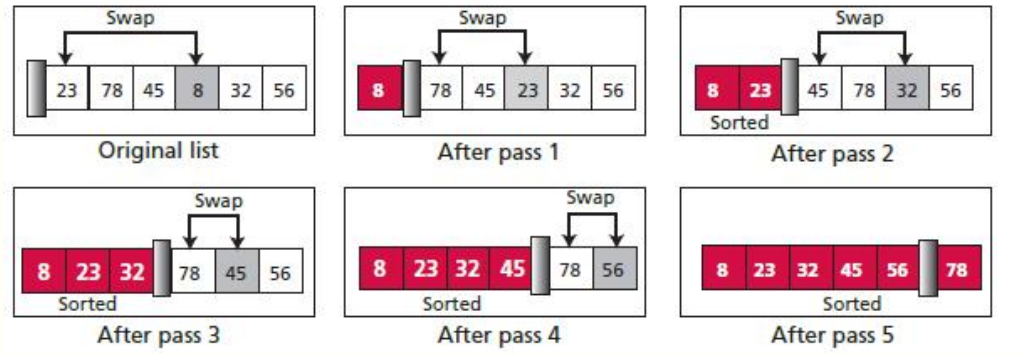


Figure 7.1 Example of selection sort

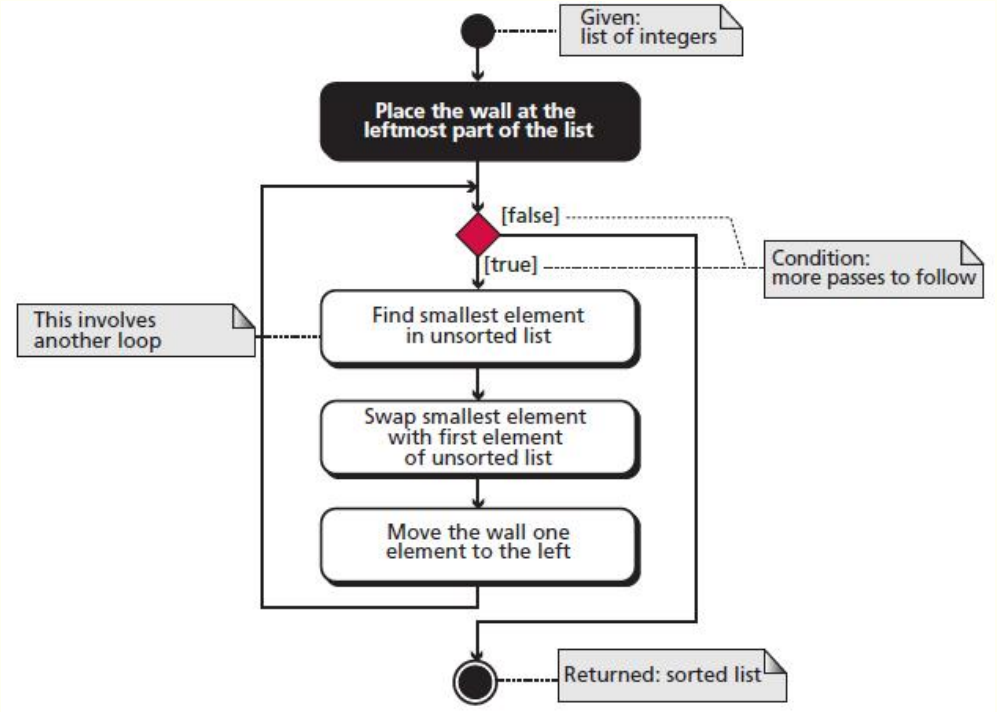


Figure 6.13 Algorithm of selection sort

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, increasing the number of sorted elements and decreasing the number of unsorted ones.
- Each time an el blist, one sort pass is com

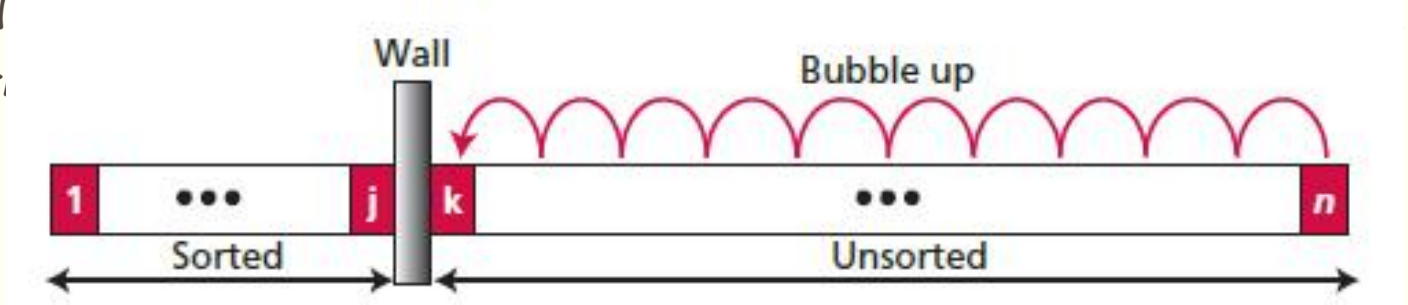


Figure 6.14 Example of selection sort

5- SEARCH ALGORITHMS: LINEAR, BINARY

5.1 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 (linear) 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.

5.2 Linear Search

- A **linear search** or **sequential search** is a method for finding an element within a list . It sequentially checks each element of the list until a match is found or the whole list has been searched.

- A linear search runs in at worst linear time and makes at most n comparisons, where n is the length of the list. If each element is equally likely to be searched, then linear search has an average case of $n+1/2$ comparisons, but the average case can be affected if the search probabilities for each element vary.

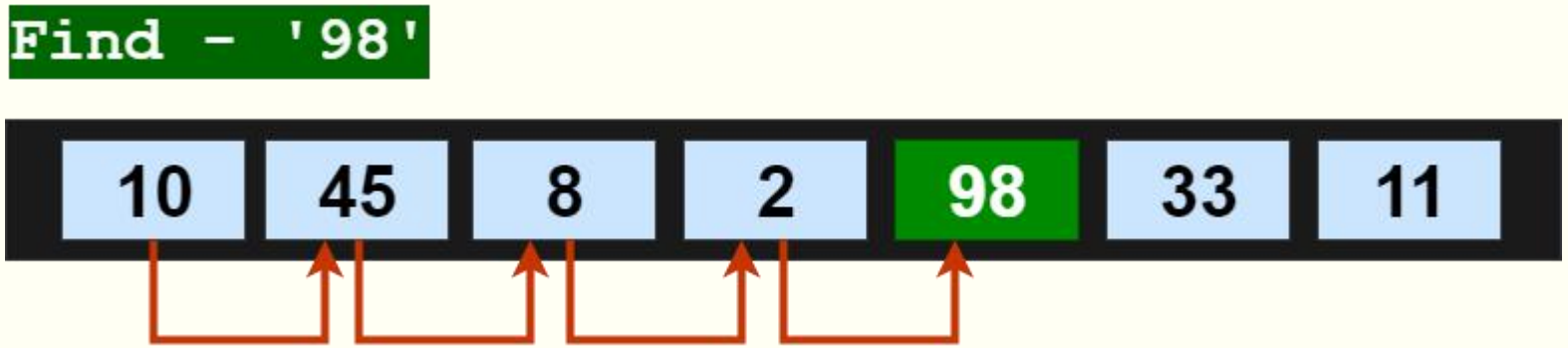


Figure 6.15 Find '98' in list by **linear search**

Linear Search (Pseudocode)

1. Take the input array arr[] from user.
2. Take element(x) you want to search in this array from user.
3. Set flag variable as -1
4. LOOP : arr[start] -> arr[end]
    1. if match found i.e arr[current_postion] == x then
        1. Print "Match Found at position" current_position.
        2. flag = 0
        3. abort
5. After loop check flag variable.
    1. if flag == -1
        1. print "No Match Found"
6. STOP

5.3 Binary Search

- **binary search**, also known as **half-interval search**, **logarithmic search**, or **binary chop**, is a search algorithm that finds the position of a target value within a sorted array.
- Binary search compares the target value to the middle element of the array.
- If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found.
- If the search ends with the remaining half being empty, the target is not in the array.

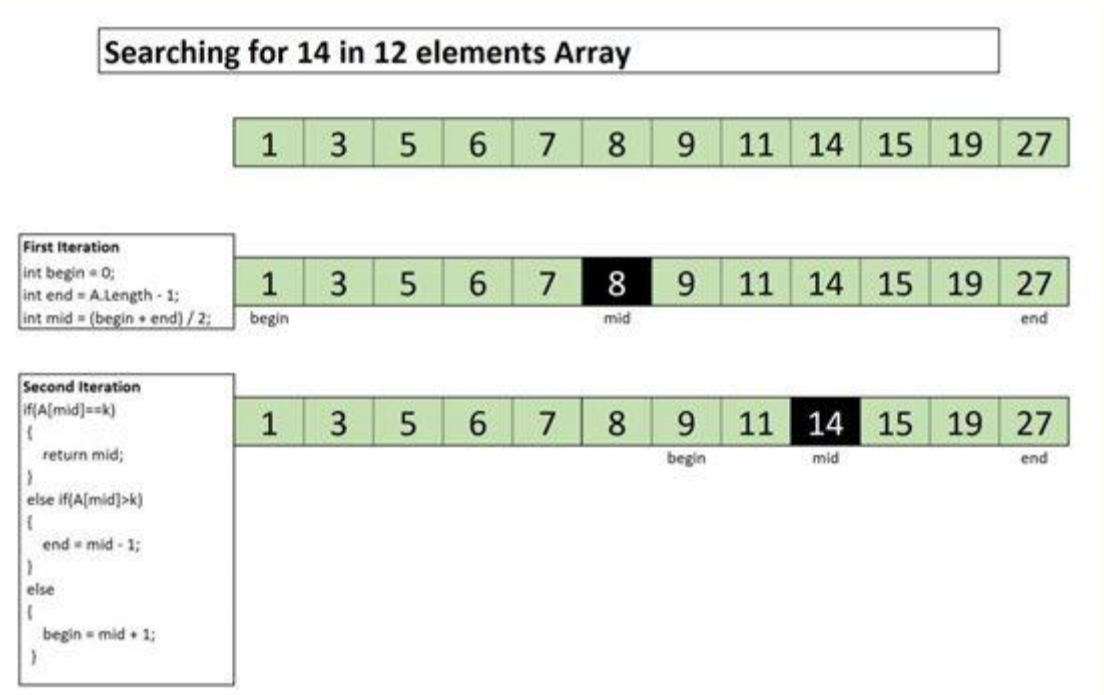


Figure 6.16 Find '14' in list by **Binary search**

Binary Search (Pseudocode)

i = left endpoint of search interval

j = right endpoint of search interval

Start: i = 1, j = n, x = (term to find)

While i<j do the following:

Compute middle index of list, $m = \lfloor (i+j)/2 \rfloor$

If $x > a_m$, search in second half of list $(a_{m+1}, a_{m+2}, \ldots a_j)$

and set i = m+1

Else search in first half of list $(a_i, a_{i+1}, \ldots a_m)$ and set j = m

Figure 6.17 *Pseudocode for* **Binary search**