Chapter 7

Problem Solving and Algorithms

Chapter Goals

- Describe the computer problem-solving process and relate it to *Polya's How to Solve It* list
- Distinguish between a simple type and a composite type
- Describe two composite data-structuring mechanisms
- Recognize a recursive problem and write a recursive algorithm to solve it
- Distinguish between an unsorted array and a sorted array
- Distinguish between a selection sort and an insertion sort

Chapter Goals

- Describe the Quicksort algorithm
- Apply the selection sort, the bubble sort, insertion sort, and Quicksort to an array of items by hand
- Apply the binary search algorithm
- Demonstrate an understanding of the algorithms in this chapter by hand-simulating them with a sequence of items

Problem Solving

Problem solving

The act of finding a solution to a perplexing, distressing, vexing, or unsettled question

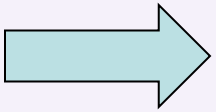
*How do **you** define problem solving?*

Problem Solving

How to Solve It: A New Aspect of Mathematical Method by George Polya

"How to solve it list" written within the context of mathematical problems

But list is quite general



We can use it to solve computer related problems!

Problem Solving

How do you solve problems?

Understand the problem

Devise a plan

Carry out the plan

Look back

Strategies

Ask questions!

- *What do I know about the problem?*
- *What is the information that I have to process in order the find the solution?*
- *What does the solution look like?*
- *What sort of special cases exist?*
- *How will I recognize that I have found the solution?*

Strategies

Ask questions! Never reinvent the wheel!

Similar problems come up again and again in different guises

A good programmer recognizes a task or subtask that has been solved before and plugs in the solution

Can you think of two similar problems?

Strategies

Divide and Conquer!

Break up a large problem into smaller units and solve each smaller problem

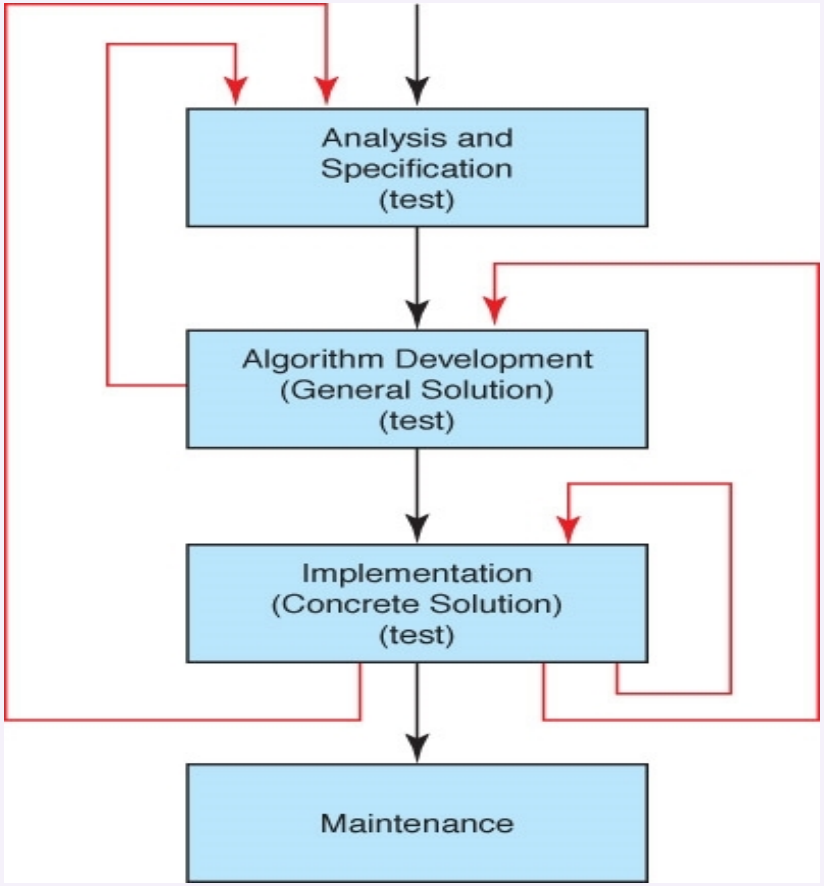
- Applies the concept of abstraction
- The divide-and-conquer approach can be applied over and over again until each subtask is manageable

Computer Problem-Solving

- Analysis and Specification Phase
  - Analyze
  - Specification
- Algorithm Development Phase
  - Develop algorithm
  - Test algorithm
- Implementation Phase
  - Code algorithm
  - Test algorithm
- Maintenance Phase
  - Use
  - Maintain

Can you name a recurring theme?

Phase Interactions



Should we add another arrow?

(What happens if the problem is revised?)

Algorithms

Algorithm

A set of unambiguous instructions for solving a problem or subproblem in a finite amount of time using a finite amount of *data*

Abstract Step

An algorithmic step containing unspecified details

Concrete Step

An algorithm step in which all details are specified

Developing an Algorithm

Two methodologies used to develop computer solutions to a problem

- Top-down design focuses on the **tasks** to be done
- Object-oriented design focuses on the **data** involved in the solution (We will discuss this design in Ch. 9)

Summary of Methodology

Analyze the Problem

Understand the problem!!

Develop a plan of attack

List the Main Tasks (es Main Module)

Restate problem as a list of tasks (modules)

Give each task a name

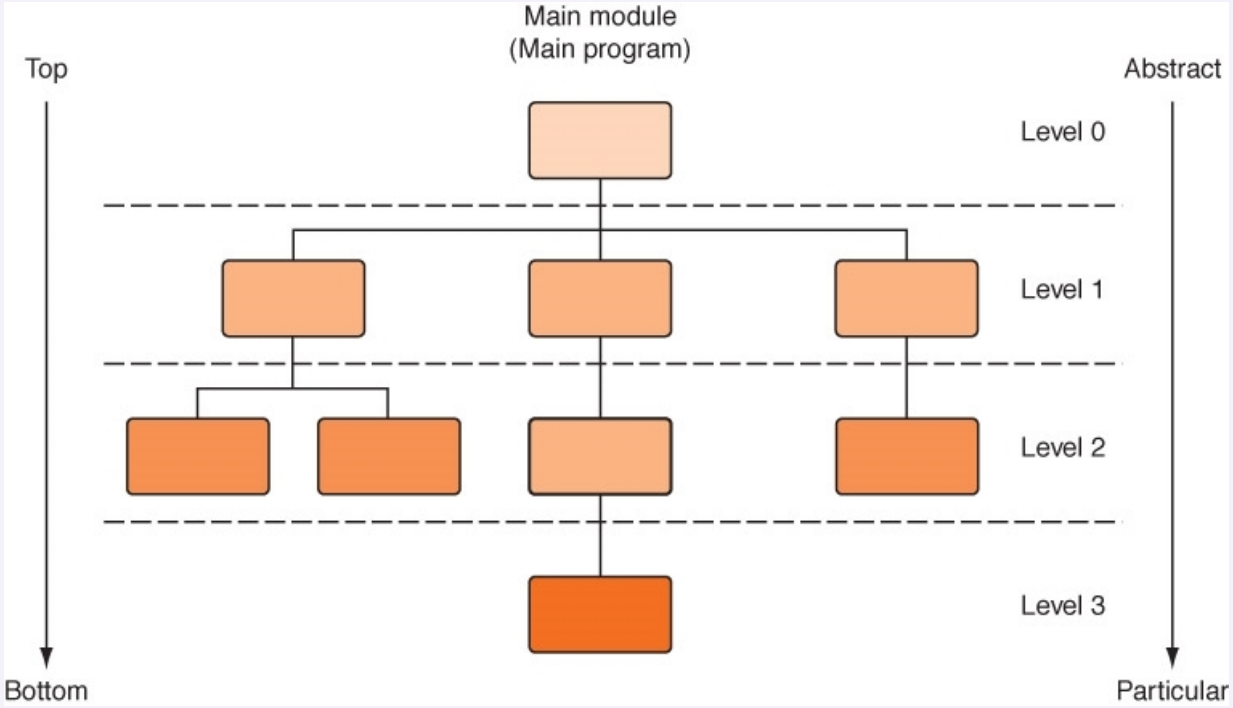
Write the Remaining Modules

Restate each abstract module as a list of tasks

Give each task a name

Resequence and Revise as Necessary

Top-Down Design



Process continues for as many levels as it takes to make every step concrete

Name of (sub)problem at one level es a module at next lower level

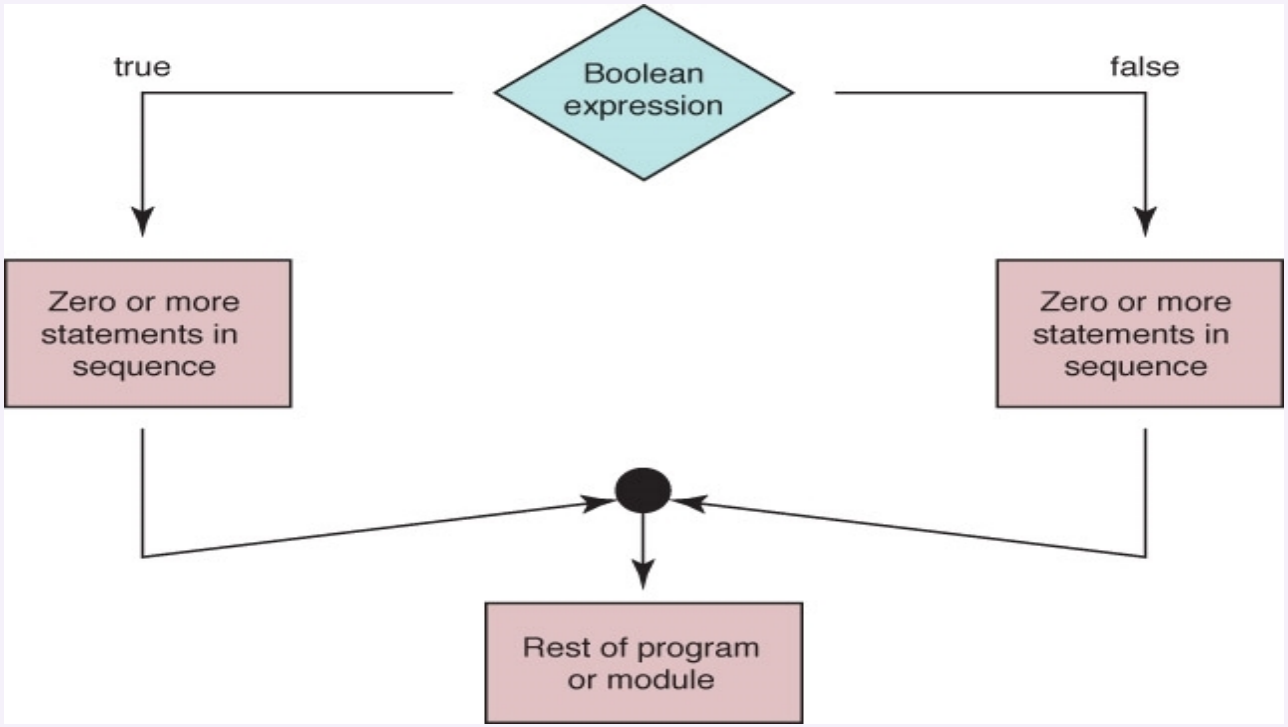
Control Structures

Control structure

An instruction that determines the order in which other instructions in a program are executed

Can you name the ones we defined in the functionality of pseudocode?

Selection Statements



Flow of control of if statement

Algorithm with Selection

Problem: Write the appropriate dress for a given temperature.

Write "Enter temperature"
Read temperature
Determine Dress

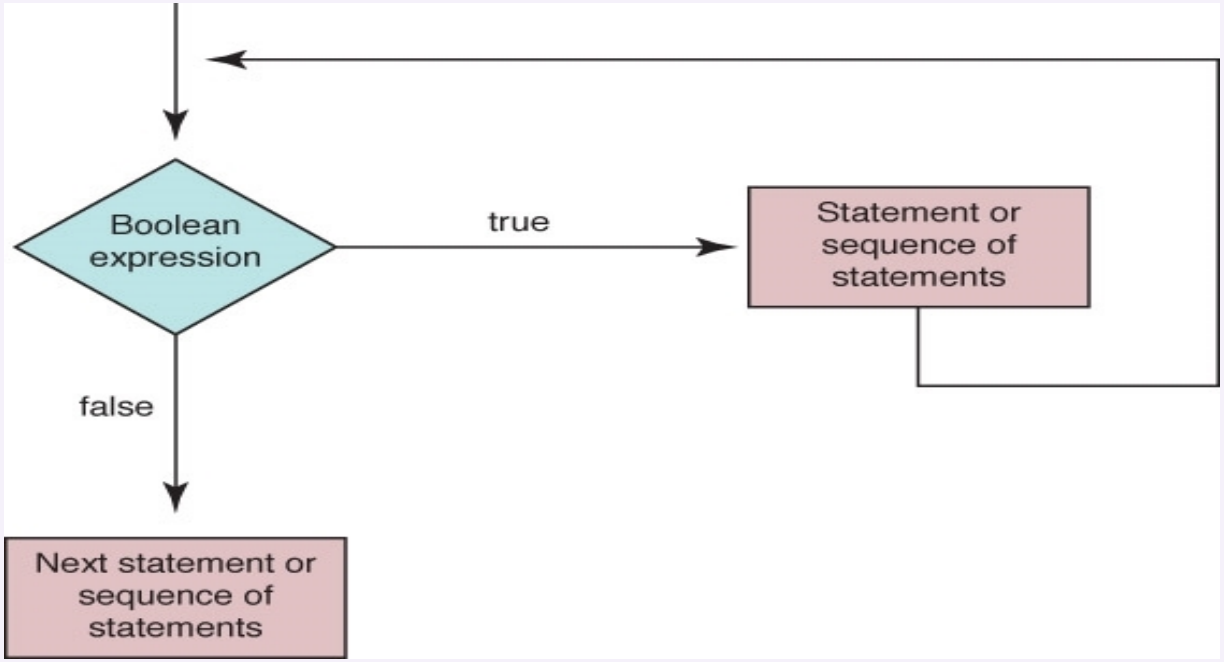
Which statements are concrete?
Which statements are abstract?

Algorithm with Selection

Determine Dress

```
IF (temperature > 90)
        Write “Texas weather: wear shorts”
ELSE IF (temperature > 70)
        Write “Ideal weather: short sleeves are fine”
ELSE IF (temperature > 50)
        Write “A little chilly: wear a light jacket”
ELSE IF (temperature > 32)
        Write “Philadelphia weather: wear a heavy coat”
ELSE
        Write “Stay inside”
```

Looping Statements



Flow of control of while statement

Looping Statements

A count-controlled loop

```
Set sum to 0
Set count to 1
While (count <= limit)
    Read number
    Set sum to sum + number
    Increment count
Write "Sum is " + sum
```

Why is it called a count-controlled loop?

Looping Statements

An event-controlled loop

```
Set sum to 0
Set allPositive to true
WHILE (allPositive)
    Read number
    IF (number > 0)
        Set sum to sum + number
    ELSE
        Set allPositive to false
Write "Sum is " + sum
```

Why is it called an event-controlled loop?
What is the event?

Looping Statements

Calculate Square Root

Read in square
Calculate the square root
Write out square and the square root

Are there any abstract steps?

Looping Statements

Calculate Square Root

```
Set epsilon to 1
WHILE  (epsilon > 0.001)
    Calculate new guess
    Set epsilon to abs(square - guess * guess)
```

Are there any abstract steps?

Looping Statements

Calculate New Guess

Set newGuess to

(guess + (square/guess)) / 2.0

Are there any abstract steps?

Looping Statements

```
Read in square
Set guess to square/4
Set epsilon to 1
WHILE  (epsilon > 0.001)
    Calculate new guess
    Set epsilon to abs(square - guess * guess)
Write out square and the guess
```

Composite Data Types

Records

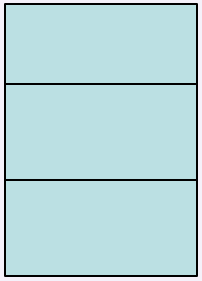
A named heterogeneous collection of items in which individual items are accessed by name. For example, we could bundle name, age and hourly wage items into a record named *Employee*

Arrays

A named homogeneous collection of items in which an individual item is accessed by its position (index) within the collection

Composite Data Types

Employee



name

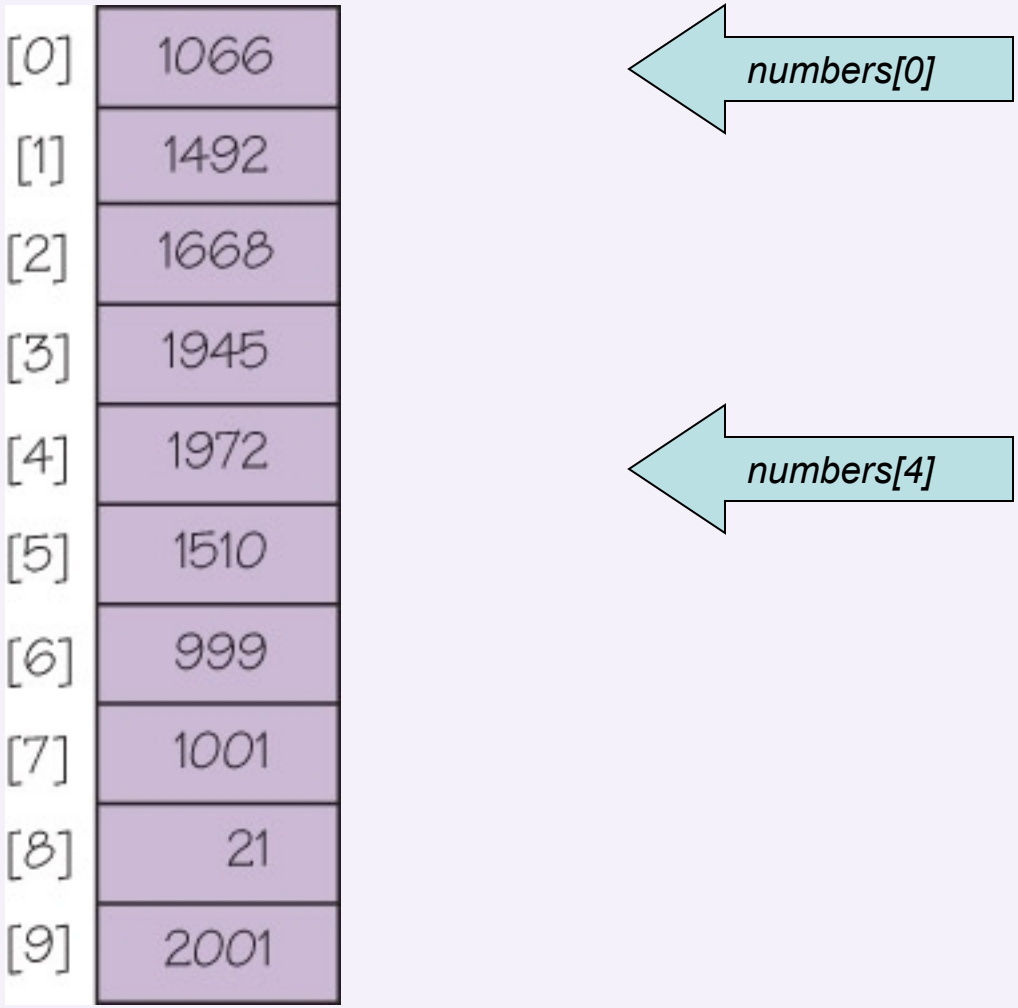
age

hourly/Wage

Following algorithm, stores values into the fields of record:

Employee employee *// Declare and Employee variable*
Set employee.name to “Frank Jones”
Set employee.age to 32
Set employee.hourlyWage to 27.50

Composite Data Types



Arrays

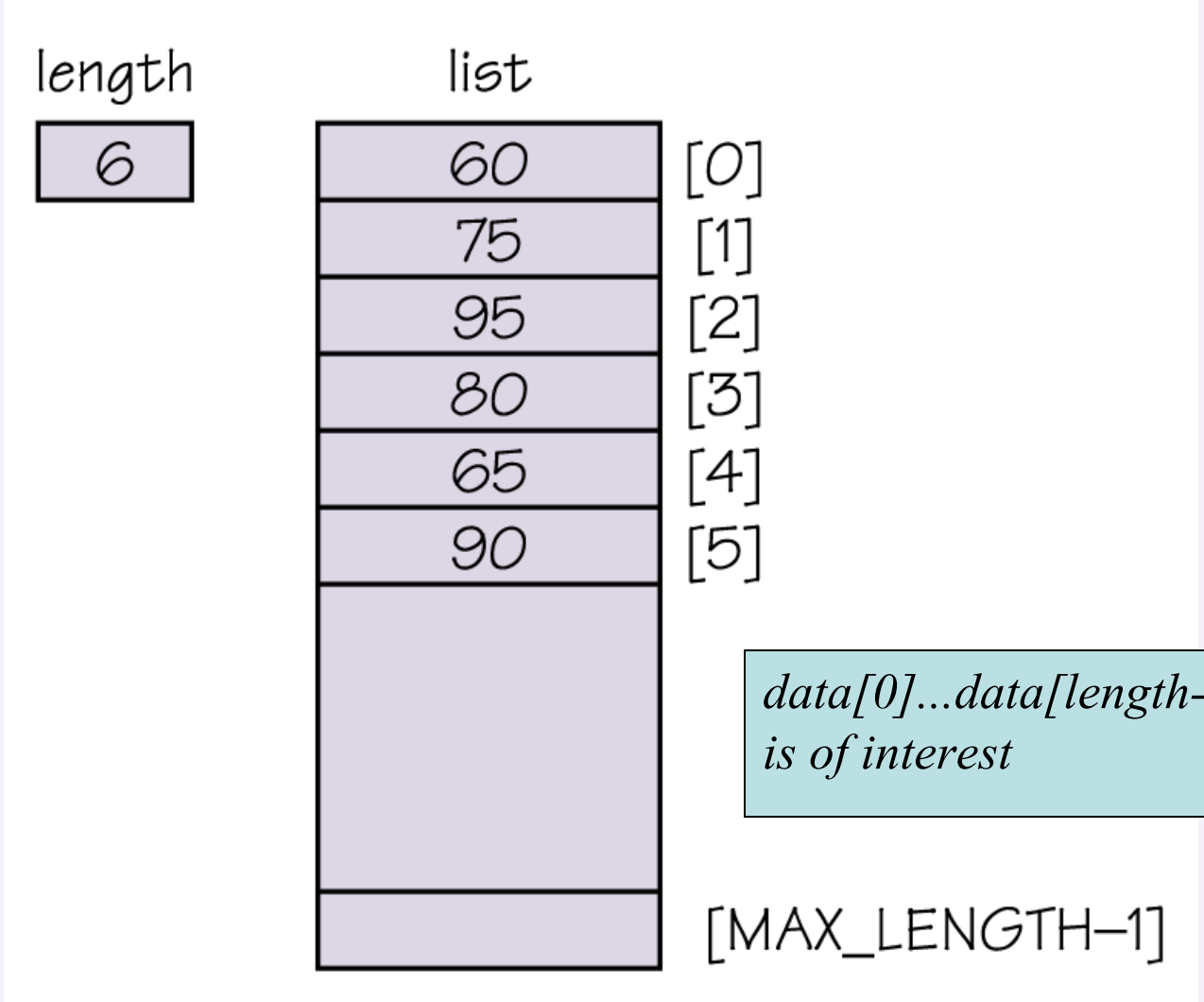
As data is being read into an array, a counter is updated so that we always know how many data items were stored

If the array is called *list*, we are working with

list[0] to list[length-1] *or*

list[0]..list[length-1]

An Unsorted Array



Composite Data Types

Fill array numbers with *limit* values

```
integer data[20]
Write “How many values?”
Read length
Set index to 0
WHILE (index < length)
    Read data[index]
    Set index to index + 1
```

Sequential Search of an Unsorted Array

A sequential search examines each item in turn and compares it to the one we are searching.

If it matches, we have found the item. If not, we look at the next item in the array.

We stop either when we have found the item or when we have looked at all the items and not found a match

Thus, a loop with two ending conditions

Sequential Search Algorithm

```
Set Position to 0
Set found to FALSE
WHILE (position < length AND NOT found )
        IF (numbers [position] equals searchitem)
                Set Found to TRUE
        ELSE
                Set position to position + 1
```

Booleans

Boolean Operators

A Boolean variable is a location in memory that can contain either *true* or *false*

Boolean operator AND returns *TRUE* if both operands are true and *FALSE* otherwise

Boolean operator OR returns *TRUE* if either operand is true and *FALSE* otherwise

Boolean operator NOT returns *TRUE* if its operand is false and *FALSE* if its operand is true

Sorted Arrays

The values stored in an array have unique keys of a type for which the relational operators are defined

Sorting rearranges the elements into either ascending or descending order within the array

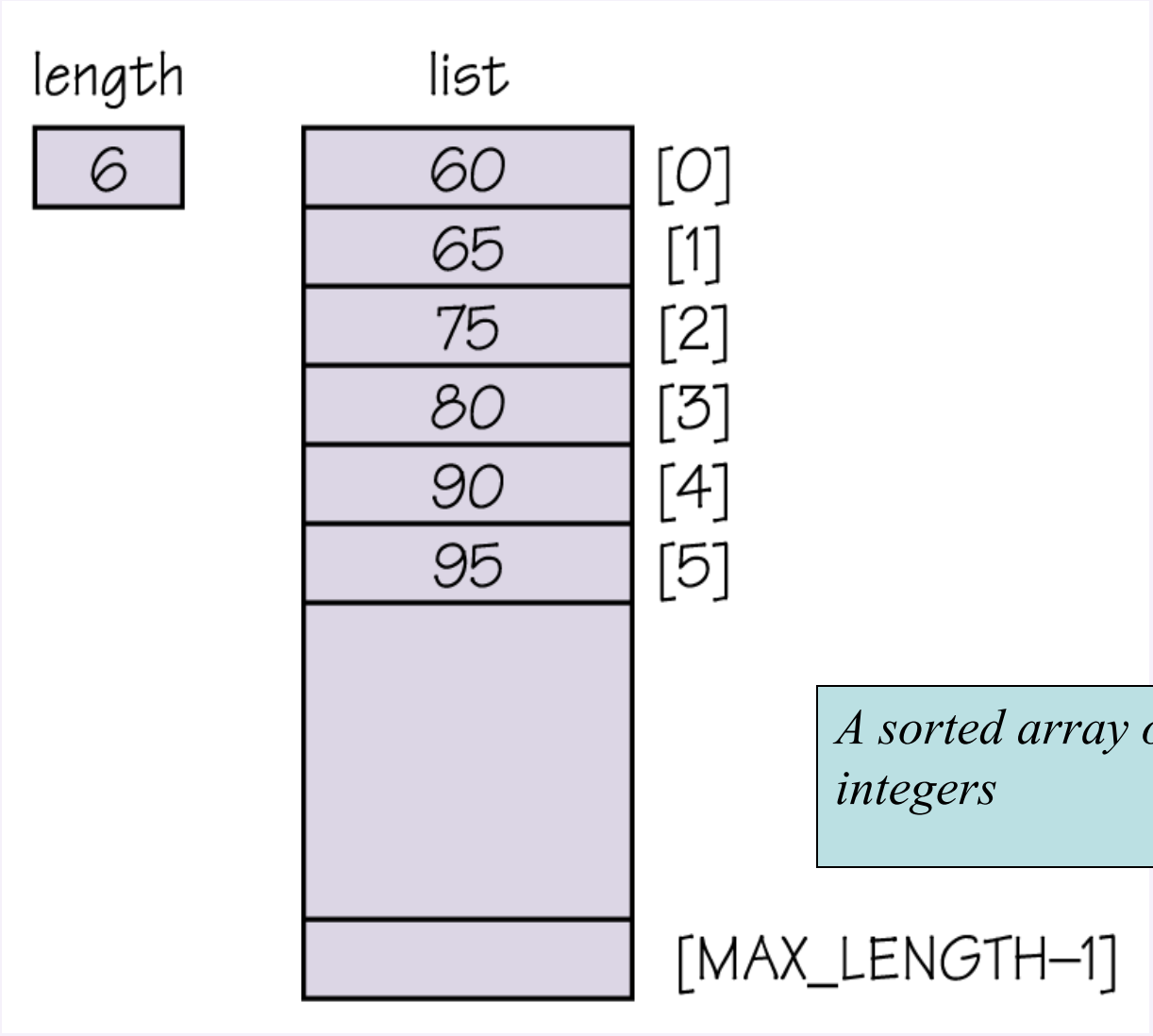
A sorted array is one in which the elements are in order

Sequential Search in a Sorted Array

If items in an array are sorted, we can stop looking when we pass the place where the item would be it were present in the array

Is this better?

A Sorted Array



A Sorted Array

Read in array of values

Write "Enter value for which to search"

Read searchItem

Set found to TRUE if searchItem is there

IF (found)

    Write "Item is found"

ELSE

    Write "Item is not found"

A Sorted Array

```
Set found to TRUE if searchItem is there
Set index to 0
Set found to FALSE
WHILE (index < length AND NOT found)
        IF (data[index] equals searchItem)
                Set found to TRUE
        ELSE IF (data[index] > searchItem)
                Set index to length
        ELSE
                Set index to index + 1
```