CIS1400 – Programming Logic and Technique

Topic 9 → Algorithms

Chapter Topics

8.2 Sequentially Searching an Array → *previous chapter*

9.1 Bubble Sort Algorithm

9.2 Selection Sort Algorithm

9.3 Insertion Sort Algorithm

9.4 Binary Search Algorithm

Algorithm Summary

The xSortLab Applet

8.2 Sequentially Searching an Array

A sequential search algorithm is a simple technique for finding an item in a string or numeric array

- AKA linear search
- Array elements are unordered
- Uses a loop to sequentially step through an array
- Compares each element with the value being searched for
- Stops when the value is found **or** the end of the array is reached

```
Set found = False
Set index = 0
While found == False AND index <= SIZE -1
  If (array[index] == searchValue) Then
        Set found = True
  Else
        Set index = index + 1
  End If
End While
```

8.2 Sequentially Searching an Array

Example: Program 8-6

```
Constant Integer SIZE = 10
Declare Integer scores[SIZE] = 87, 75, 98, 100, 82,
  72, 88, 92, 60, 78
Declare Boolean found
Declare Integer index
Set found = False
Set index = 0
```

array to search

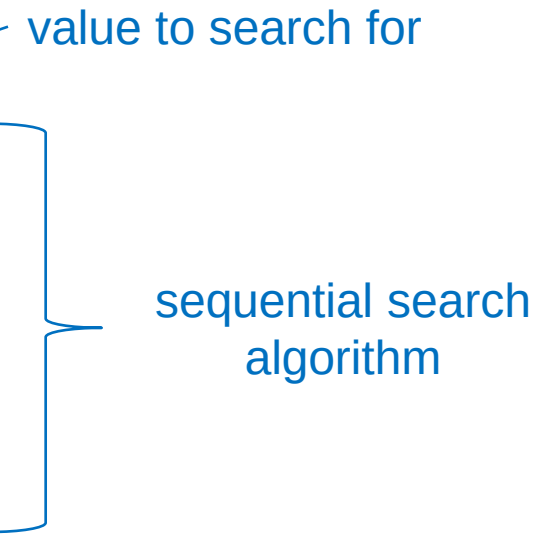
Boolean flag

loop counter

initialize flag and counter

8.2 Sequentially Searching an Array

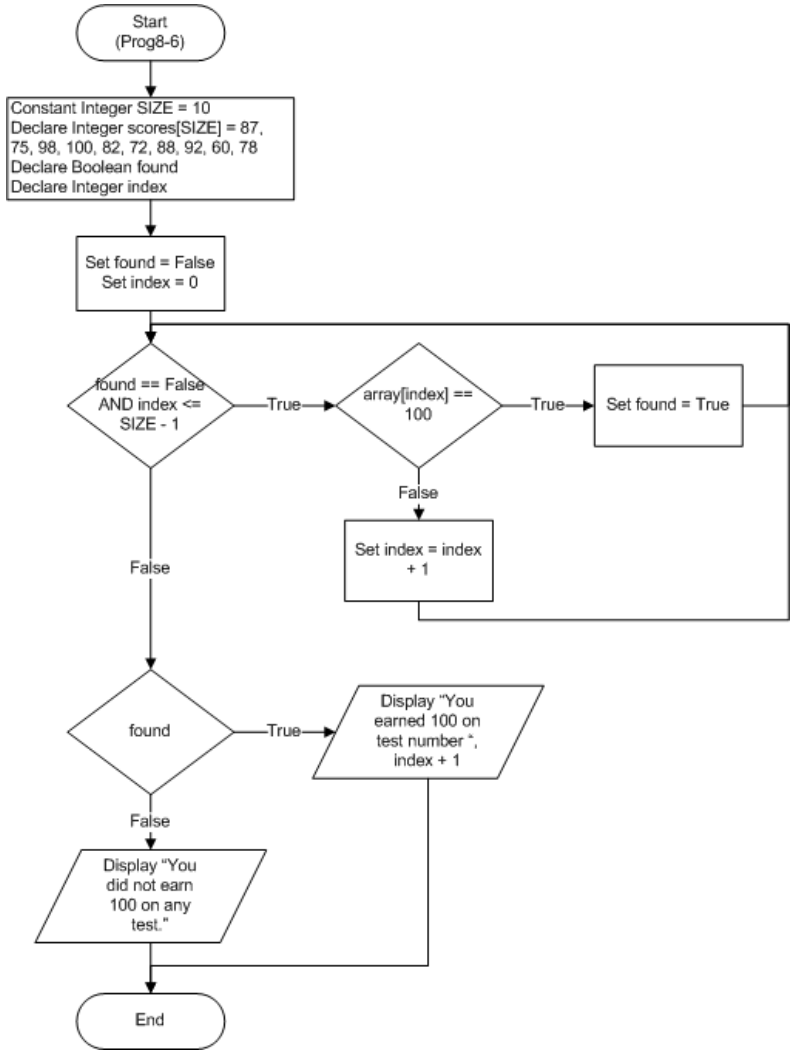
Example: Program 8-6 *(cont'd)*




```
While found == False AND index <= SIZE - 1
  If array[index] == 100 Then
    Set found = True
  Else
    Set index = index + 1
  End If
End While
If found Then
  Display "You earned 100 on test number ", index + 1
Else
  Display "You did not earn 100 on any test."
End If
```

8.2 Sequentially Searching an Array

Example: Program 8-6



VB example

9.1 The Bubble Sort Algorithm

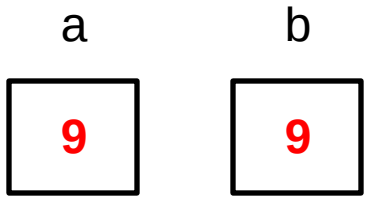
Bubble sort is a simple sorting algorithm for rearranging the contents of an array

- Useful for alphabetical lists and numerical sorting
- Can be done in ascending or descending order
- With the Bubble Sort,
  - array elements are compared
  - if current element is greater than next element
  - elements are swapped
- Larger numbers 'bubble' toward the end of the array
  - when sorting in ascending (increasing) order
- Smaller numbers 'bubble' toward the end of the array
  - when sorting in descending (decreasing) order

Swapping Variable Values

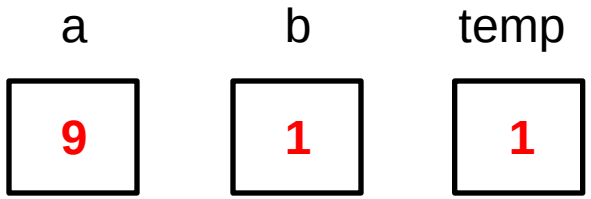
- Exchanging variable values in memory?

```
Declare Integer a = 1
Declare Integer b = 9
Set a = b
Set b = a
```



- Exchanging variable values in memory:

```
Declare Integer a = 1
Declare Integer b = 9
Declare Integer temp
Set temp = a
Set a = b
Set b = temp
```

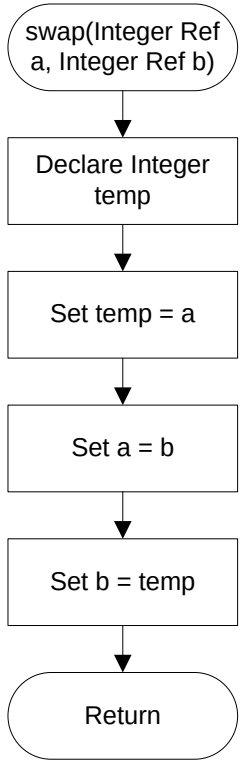


The Swap Module

- In most of the sorts, a swap module can be called
- It is the same in each sorting algorithm and only changes in the parameter list to account for the type of data passed to it

```
//This swap module accepts two
// Integer arguments
Module swap(Integer Ref a, Integer Ref b)
  Declare Integer temp
  //swap the values
  Set temp = a
  Set a = b
  Set b = temp
End Module
```

pass by reference



9.1 The Bubble Sort Algorithm

Program 9-1 **bubbleSort module (not a complete program)**

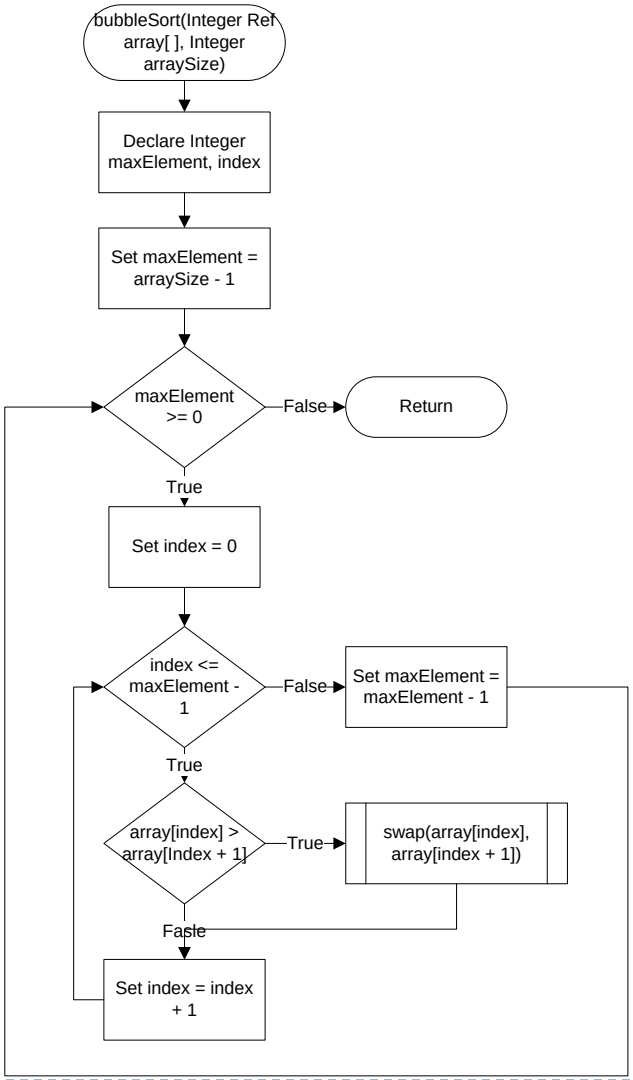
```
 1 Module bubbleSort(Integer Ref array[], Integer arraySize)
 2     // The maxElement variable will contain the subscript
 3     // of the last element in the array to compare.
 4     Declare Integer maxElement
 5
 6     // The index variable will be used as a counter
 7     // in the inner loop.
 8     Declare Integer index
 9
10     // The outer loop positions maxElement at the last
11     // element to compare during each pass through the
12     // array. Initially maxElement is the index of the
13     // last element in the array. During each iteration,
14     // it is decreased by one.
15     For maxElement = arraySize - 1 To 0 Step -1
16
17        // The inner loop steps through the array, comparing
18        // each element with its neighbor. All of the
19        // elements from index 0 through maxElement are
20        // involved in the comparison. If two elements are
21        // out of order, they are swapped.
22        For index = 0 To maxElement - 1
23
24           // Compare an element with its neighbor and swap
25           // if necessary.
26           If array[index] > array[index + 1] Then
27              Call swap(array[index], array[index + 1])
28           End If
29        End For
30     End For
31 End Module
```

compare current with next and swap, if appropriate

9.1 The Bubble Sort Algorithm

Inside Program 9-1

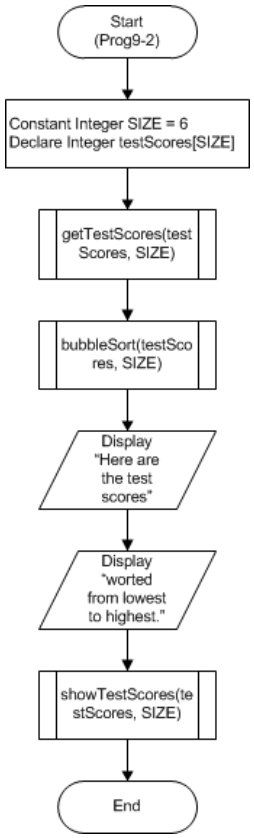
- *maxElement* variable holds the subscript of the last element that is to be compared to its neighbor
- *index* variable is an array subscript
- The outer loop iterates from the end of the array to the beginning
- The inner loop iterates for each of the unsorted array elements
  - Up to the *maxElement* of the outer loop
- The if statement does the comparison between the current and next elements
  - Swap elements if appropriate

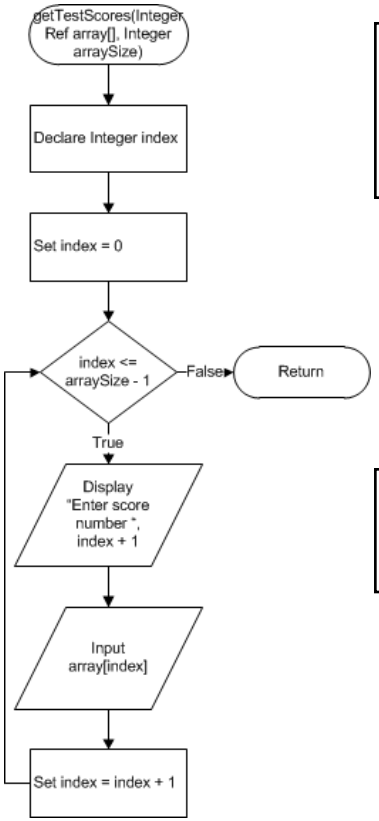


9.1 The Bubble Sort Algorithm

Sorting an array of test scores

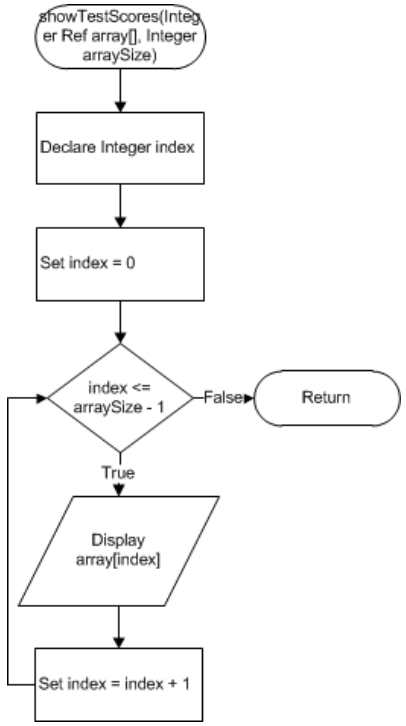
- In the Spotlight: See Program 9-2





bubbleSort(Integer Ref array[ ], Integer arraySize)

swap(Integer Ref a, Integer Ref b)



VB example

9.1 The Bubble Sort Algorithm

Sorting an array of strings to put information in alphabetical order can be done with a Bubble Sort

- See Program 9-3

Algorithm can be modified to sort in descending order

- Compare Program 9-3 and Program 9-4
  - `If array[index] > array[index+1] Then`
    `// Program 9-3 line 60 – **increasing** sort`
  - `If array[index] < array[index+1] Then`
    `// Program 9-4 line 60 – **decreasing** sort`

9.2 The Selection Sort Algorithm

The selection sort works *similar* to the bubble sort, but is more efficient

- Bubble sort moves one element at a time
- Selection sort performs fewer swaps because it moves items immediately to their final position
- With the Selection Sort,
  - For each element in array from 0 to size - 2
    - minimum value and index are initialized to beginning element
    - minimum value is compared with each successive array element
      - if current element is less than minimum value
      - minimum value and index are reset to current
    - minimum value is swapped with beginning element

9.2 The Selection Sort Algorithm

Pseudocode for the selectionSort Module

```
Module selectionSort(Integer Ref array[], Integer arraySize)
  Declare Integer startScan
  Declare Integer minIndex
  Declare Integer minValue
  Declare Integer index
  For startScan = 0 To arraySize - 2
    Set minIndex = startScan
    Set minValue = array[startScan]
    For index = startScan + 1 to arraySize - 1
      If array[index] < minValue Then
        Set minValue = array[index]
        Set minIndex = index
      End If
    End For
    Call swap(array[minIndex], array[startScan])
  End For
End Module
```

find smallest value and index

9.2 The Selection Sort Algorithm

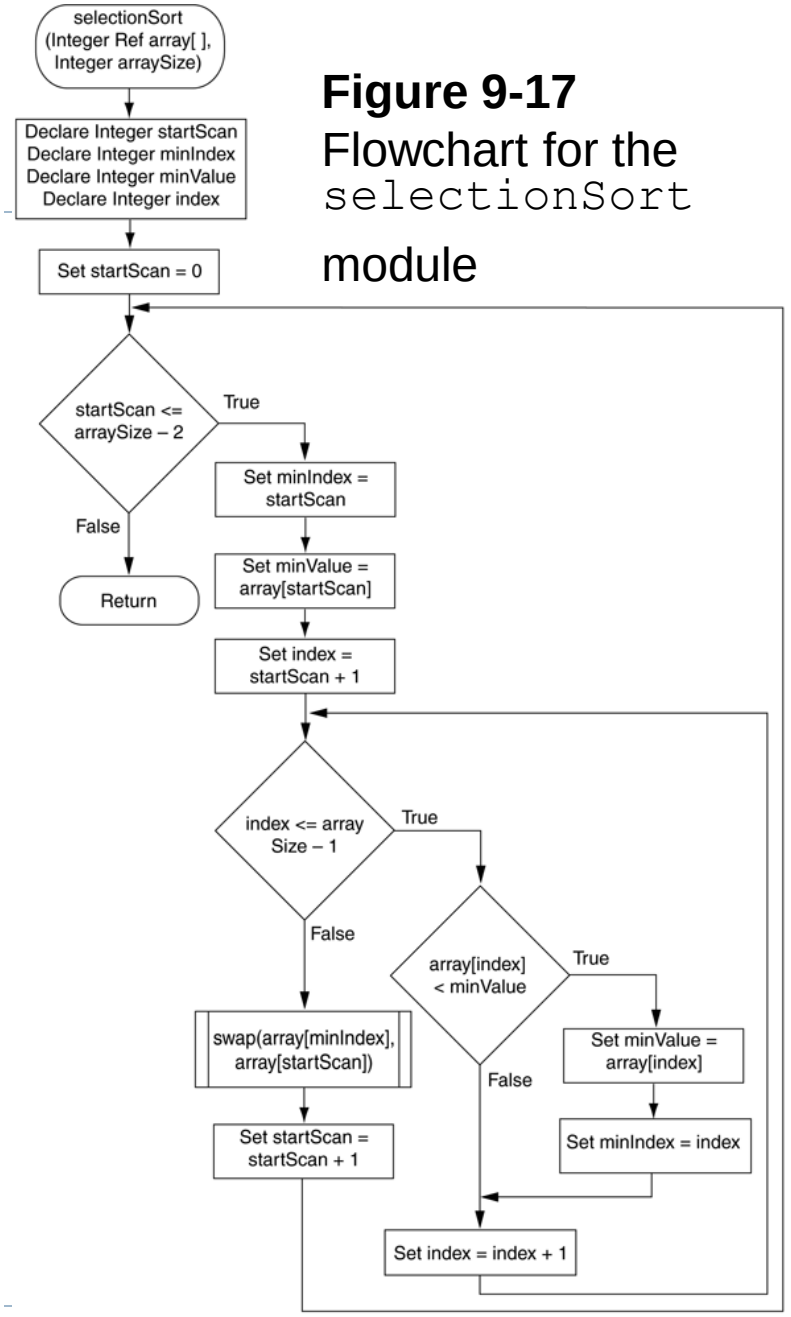


Figure 9-17
Flowchart for the `selectionSort` module

Inside Figure 9-17

- *minIndex* holds the subscript of the element with the smallest value
- *minValue* holds the smallest value found
- The outer loop iterates for each element in the array, except the last
- The inner loop performs the scan to find smallest element

9.2 The Selection Sort Algorithm

Sorting an array of integers

- See Program 9-5

Algorithm can be modified to sort in descending order

- Modify Program 9-5 line 63 from

```
If array[index] < minValue Then
```

- to

```
If array[index] > maxValue Then
```

VB example

9.3 The Insertion Sort Algorithm

The insertion sort works with a small sorted array and then inserts remaining elements into the sorted array

- With the Insertion Sort
  - First two elements are sorted *(sorted array subset)*
  - Each remaining element is placed into the proper position of the sorted array subset
  - Sorted array subset becomes larger with each insertion until
  - Entire array is sorted and there are no unsorted elements
- Also more efficient than the bubble sort

9.3 The Insertion Sort Algorithm

Pseudocode for the insertionSort Module

```
Module insertionSort(Integer Ref array[], Integer arraySize)
  Declare Integer index
  Declare Integer scan
  Declare Integer unsortedValue
  For index = 1 To arraySize - 1
    Set unsortedValue = array[index]
    Set scan = index
    While scan > 0 AND array[scan-1] > unsortedValue
      Set array[scan] = array[scan-1]
      Set scan = scan - 1
    End While
    Set array[scan] = unsortedValue
  End For
End Module
```

find proper location for unsorted value

9.3 The Insertion Sort Algorithm

Inside Figure 9-24

- *scan* is used to scan through the array
- *unsortedValue* holds the first unsorted value
- The outer loop steps the index variable through each subscript of unsorted array subset *(starting at 1)*
- The inner loop moves the first element outside the sorted subset and into its proper position

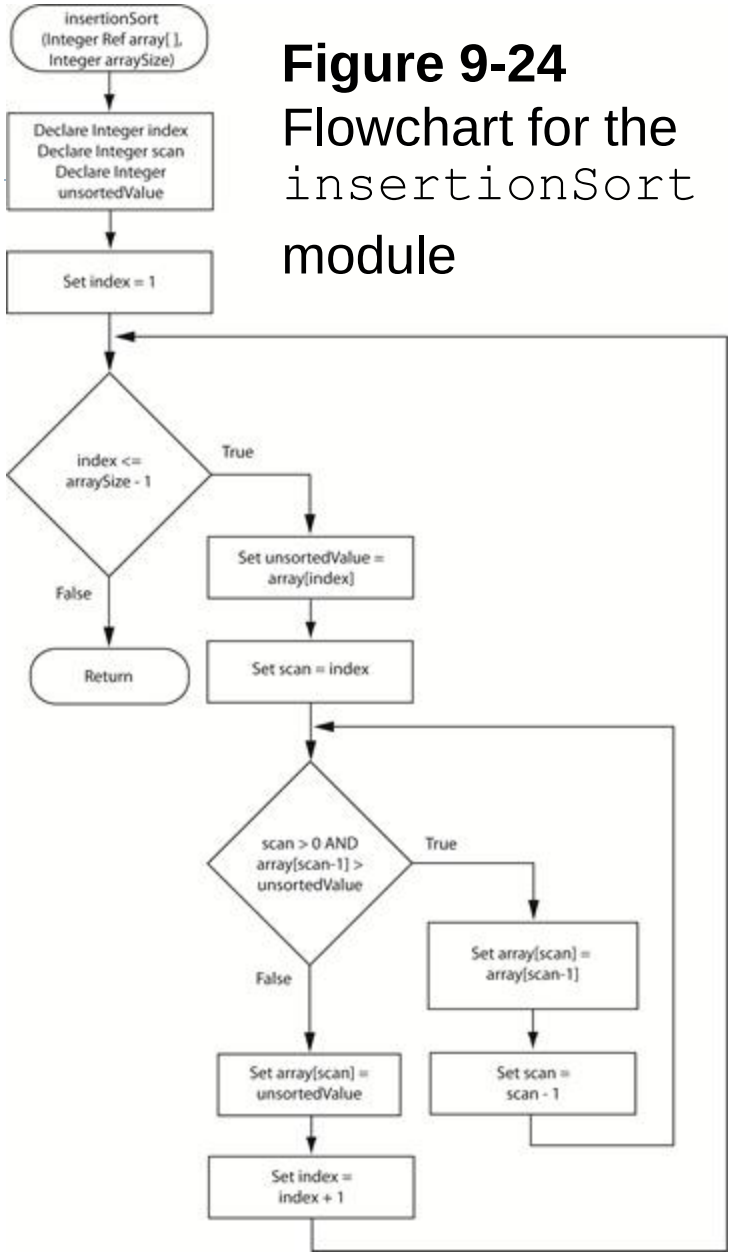


Figure 9-24 Flowchart for the `insertionSort` module

9.3 The Insertion Sort Algorithm

Sorting an array of integers

- See Program 9-6

Algorithm can be modified to sort in descending order

- Modify Program 9-6 line 60 from

```
While scan > 0 AND array[scan - 1] < unsortedValue Then
```

- to

```
While scan > 0 AND array[scan - 1] > unsortedValue Then
```

VB example

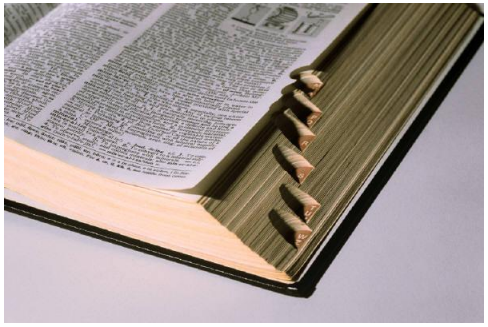
9.4 The Binary Search Algorithm

The binary search algorithm locates an item in an array by repeatedly dividing the array in half

- Array elements must be in sorted order
- Each time it divides the array, it eliminates the half of the array that does not contain the item
- It's more efficient than the sequential search because each time it cuts the array in half and makes a smaller number of comparisons to find a match, but it must first be in sorted order!

9.4 The Binary Search Algorithm

- With the Binary Search,
  - The first comparison is done with the middle element of the array to see if it is greater than or less than the number that is being searched
  - If the middle element is **greater** than the number, then the number must be in the first half of the array
  - If the middle element is **less than** the number, then the number must be in the second half of the array
- This process is continued until the match if found



9.4 The Binary Search Algorithm

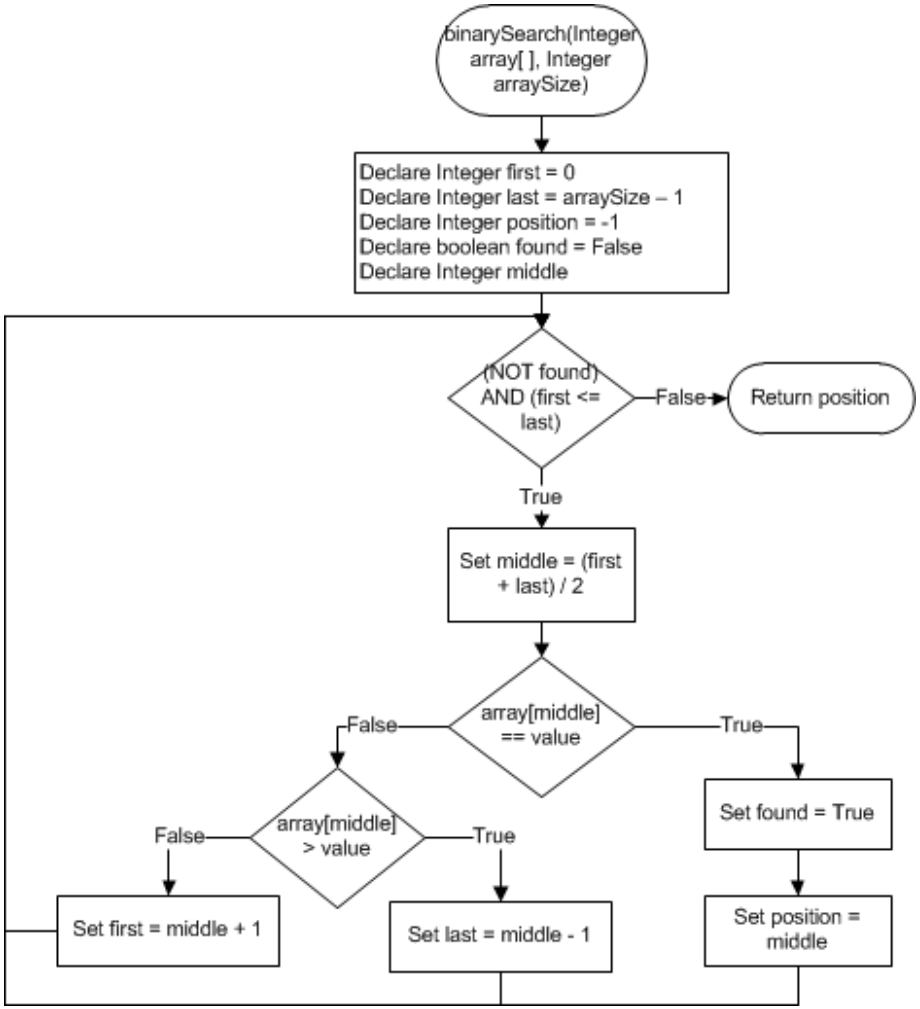
Inside Program 9-7

- *first* and *last* mark boundaries of array being searched
- *middle* is used to store the calculated middle index of array being searched
- *value* stores the value to search
- loop continues while *value* not found and valid array bounds exist, check if
  - found→check *value* with element at *middle*
  - relevant half→adjust *first* or last

Program 9-7 **(not a complete program)**

```
1  // The binarySearch function accepts as arguments an Integer
2  // array, a value to search the array for, and the size
3  // of the array. If the value is found in the array, its
4  // subscript is returned. Otherwise, -1 is returned,
5  // indicating that the value was not found in the array.
6  Function Integer binarySearch(Integer array[], Integer value,
7                                Integer arraySize)
8     // Variable to hold the subscript of the first element.
9     Declare Integer first = 0
10
11    // Variable to hold the subscript of the last element.
12    Declare Integer last = arraySize - 1
13
14    // Position of the search value
15    Declare Integer position = -1
16
17    // Flag
18    Declare Boolean found = False
19
20    // Variable to hold the subscript of the midpoint.
21    Declare Integer middle
22
23    While (NOT found) AND (first <= last)
24       // Calculate the midpoint.
25       Set middle = (first + last) / 2
26
27       // See if the value is found at the midpoint...
28       If array[middle] == value Then
29          Set found = True
30          Set position = middle
31
32       // Else, if the value is in the lower half...
33       Else If array[middle] > value Then
34          Set last = middle - 1
35
36       // Else, if the value is in the upper half...
37       Else
38          Set first = middle + 1
39       End If
40    End While
41
42    // Return the position of the item, or -1
43    // if the item was not found.
44    Return position
45 End Function
```

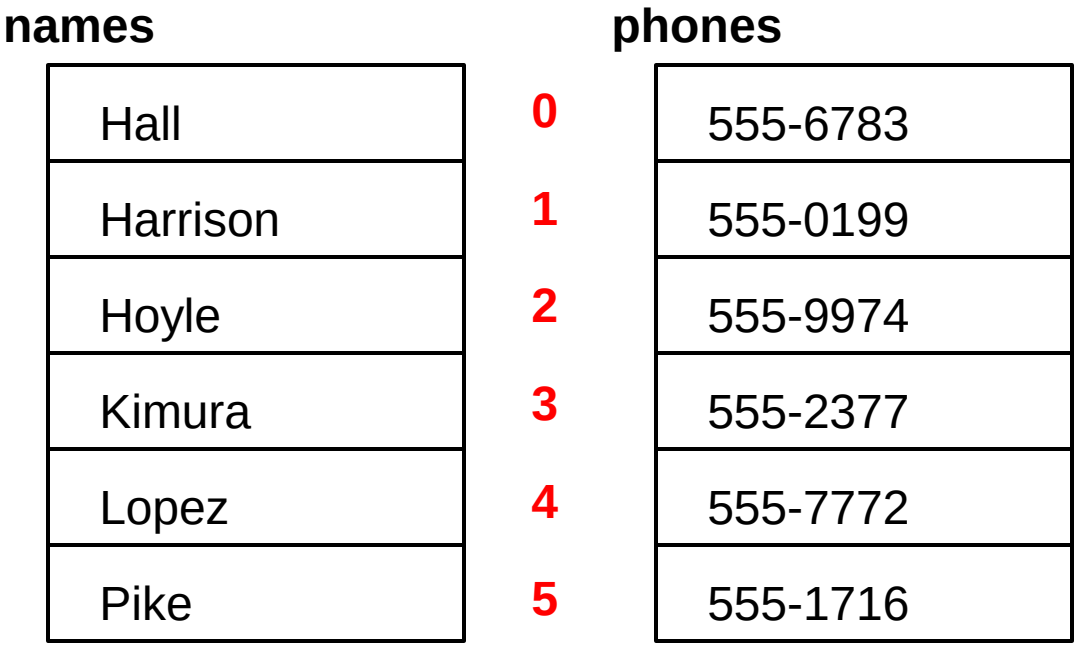
9.4 The Binary Search Algorithm



9.4 The Binary Search Algorithm

Looking up an instructor's phone number using a set of parallel arrays (*instructor names and instructor phone numbers*)

- In the Spotlight: See Program 9-8



VB example

Algorithm Summary

- Searching
  - Sequential/Linear
    - No array ordering required
    - Loop through array comparing each element with search value, stop when
      - match found
      - end of array reached
  - Binary
    - Array must be in sorted order
    - Continue dividing array in half, determine relevant half, stop when
      - match found
      - array location passed

Algorithm Summary

- Sorting
  - Bubble
    - Loop through array comparing current element with next element and swap as appropriate
      - largest/smallest element 'bubbled' to end
    - Repeat with unsorted subset of array
  - Selection
    - Loop through array to locate largest/smallest element and place in first location
    - Repeat with unsorted subset of array
  - Insertion
    - Create sorted subset of array and insert element from unsorted array subset into appropriate location of sorted array subset