Chapter 8
Searching and Sorting Arrays

Searching and Sorting

- We often need to **search** an array for a specific item.
- We often need to **sort** an array (from highest to lowest, into alphabetical order, and so on).
- There are many algorithms for searching and sorting.
- You could probably think of ways to write your own original programs to search or sort but you should also know how to evaluate algorithms that are available for a task and be able to select which best serves your purpose in a given situation.

8.1 The Serial Search Technique

Steps:

- **Load**, or populate, the arrays.
- Identify the **key** (the item to locate).
- **Search** the array that contains the key to find a match.
- **Display** the key and corresponding values in the non-key arrays with the same subscript as the one with the key value, if a match is found.
- **Display** an "`unsuccessful search`" message if no matches are found.

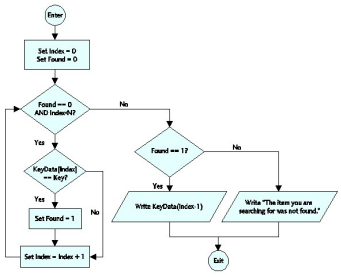
Use a Flag

- Use a flag, or switch, to decide which message to print.
- A **flag** is a `Boolean` variable that is set to `0` or `1` (`true` or `false`), depending on the results of an operation.
- The value of the flag can be checked later in the program with an **`If`** statement to determine which action to take.

General Pseudocode for the Serial Search

```
//Set the subscript (Index) of the current key to 0
Set Index = 0
//Set a flag (Found) to 0
Set Found = 0
While (Found == 0) AND (Index < N)
            If KeyData[Index] == Key Then
                        Set Found = 1
            End If
            Set Index = Index + 1
End While
If Found == 1 Then
            Write KeyData[Index - 1]
Else
            Write "The item you are searching for was not found."
End If
```


Flowchart for the Serial Search



Example: Searching and Parallel Arrays

- This program segment displays the test score for a particular student when the student's identification number is input by the user.
- It searches an array, **IDNumbers**, which consists of identification numbers for the ID number input (**IDKey**).
- Then it does the following:
  - Displays the corresponding student name and test score contained in two parallel arrays named **Names** and **Scores**, if the number **IDKey** is found in the **IDNumbers** array or…
  - Displays an appropriate message if the **IDKey** is not found in the **IDNumbers** array

Pseudocode is on next slide.

Assume that the arrays **IDNumbers**, **Names**, and **Scores** have already been declared and loaded with the necessary data.

Assume that the number of elements in each of these parallel arrays is **N**, the variables **IDKey**, and **Index**, have been declared as `Integer` variables, and **Found** has been declared as a `Boolean` variable.

Example: Searching and Parallel Arrays

```
1     Write "Enter a student ID number: "
2     Input IDKey
3     Set Index = 0
4     Set Found = 0
5     While (Found == 0) AND (Index < N)
6           If IDNumbers[Index] == IDKey Then
7                 Set Found = 1
8           End If
9           Set Index = Index + 1
10    End While
11    If Found == 0 Then
12          Write "Student ID not found"
13    Else
14          Write "ID Number: " + IDKey
15          Write "Student name: " + Names[Index - 1]
16          Write "Test score: " + Scores[Index - 1]
17    End If
```


8.2 The Bubble Sort Technique

- To sort data means to arrange it in a particular numerical or alphabetical order, ascending or descending.
- To sort small amounts of data, use the **bubble sort technique**.
- Make several passes through the data.
- Compare adjacent pairs of data.
- If the pairs are not in order, swap them.
- Continue until all data is processed.

Swapping Values

- To swap two values, you must store one of them in a temporary storage location while you make the swap.
- To interchange array elements `Array[K]` and `Array[K + 1]`, the pseudocode for the swap looks like this:

```
Set Temp = Array[K]
Set Array[K] = Array[K + 1]
Set Array[K + 1] = Temp
```

Walk through the swap routine with a few values to see how it works.

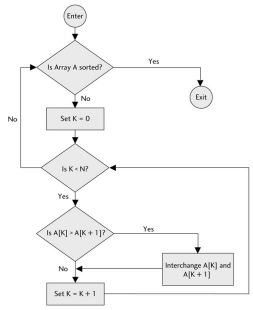
If `Array[K] = 6` and `Array[K + 1] = 18` at first, after the swap:

- `Array[K]` will contain the value `18`
- `Array[K + 1]` will contain the value `6`
- What will be the value in `Temp`?

Swap Example: The shirt can be either yellow or green and the logo can be either black or red. (assumption: only the 4 colors specified are entered but may be in incorrect order)

```
1    Write "Pick two colors: either yellow or green and either red or black."
2    Write "Enter your first color: "
3    Input Shirt
4    Write "Enter your second color: "
5    Input Logo
6    If (Shirt != "yellow") OR (Shirt != "green") Then
7          Set Temp = Shirt
8          Set Shirt = Logo
9          Set Logo = Temp
10   End If
11   Write "You have selected a " + Shirt + " shirt with a "
          + Logo + " logo."
```


Flowchart for the Bubble Sort



General Pseudocode for the Bubble Sort

Given an array, **A**, with **N** elements:

```
While (array A is not sorted)
     For (K = 0; K < N - 1; K++)
          If A[K] > A[K + 1] Then
               Interchange A[K] and A[K + 1]
          End If
     End For
End While
```

The maximum number passes needed to sort **N** numbers is **N - 1** but may be less. A flag is used to exit the array when all elements are sorted.


```
1       Declare TestScores[100] As Float
2       Declare Count, K As Integer
3       Declare Flag As Boolean
4       Declare OneScore, Temp As Float
5       Write "Enter a test score; enter -9999 when done: "
6       Input OneScore
7       Set Count = 0
8       While OneScore != -9999
9                 Set TestScores[Count] = OneScore
10                Set Count = Count + 1
11                Write "Enter a test score; enter -9999 when done: "
12                Input OneScore
13      End While(OneScore)
14      Set Flag = 0
15      While Flag == 0
16                Set Flag = 1; Set K = 0
17                While K <= (Count - 2)
18                          If TestScores[K] < TestScores[K + 1] Then
19                                    Set Temp = TestScores[K]
20                                    Set TestScores[K] = TestScores[K + 1]
21                                    Set TestScores[K + 1] = Temp
22                                    Set Flag = 0
23                          End If
24                          Set K = K + 1
25                End While(K)
26      End While(Flag)
27      Write "Sorted list..."
28      Set K = 0
29      While K <= (Count - 1)
30                Write TestScores[K]
31                Set K = K + 1
32      End While(K)
```

Example: Input numbers in any order and sort them

8.3 The Binary Search

- The **binary search** method is a good way to search a large amount of data for a particular item.
- That item is called the **search key.**
- The binary search works better for large amounts of data than the serial search.
- The binary search is:
  - more efficient than the serial search technique
  - requires that the **table keys**, the array of data to be searched, is in numerical or alphabetical order

General Process for the Binary Search

- First compare the **search key** (the **target**) to the **table key** midway through the given array.
  - Because the array data is ordered, it is possible to see in which *half* of the array the search key lies.
- Then compare the search key to the table key in the middle of this half.
  - This determines in which *quarter* of the array the search key is located.
  - Then look at the middle entry in this quarter.
  - Continue this process until search key is found.

General Pseudocode for Binary Search

```
1    Declare Low, High, N, Index As Integer
2    Declare Found As Boolean
3    Set Low = 0
4    Set High = N
5    Set Index = Int(N/2)
6    Set Found = 0
7    While (Found == 0) AND (Low <= High)
8           If Key == Array[Index] Then
9                  Set Found = 1
10          End If
11          If Key > Array[Index] Then
12                 Set Low = Index + 1
13                 Set Index = Int((High + Low)/2)
14          End If
15          If Key < Array[Index] Then
16                 Set High = Index - 1
17                 Set Index = Int((High + Low)/2)
18          End If
19   End While
```


Combining Parallel Arrays and a Binary Search

Professor Crabtree saves all her student records in parallel arrays.

The array **Names** holds the students' names, listed alphabetically by last name.

Each time she gives an exam or grades a homework assignment, she adds a new parallel array: **Exam1**, **Exam2**, **HW1**, **HW2**, and so on.

Now Dr. Crabtree wants to be able to locate the record for one specific student.

The pseudocode on the next slide assumes the following parallel arrays have been declared, filled with data, and that the following variables have been declared:

- **Names**[100] is an array of Strings with each element holding a student's last name
- **First**[100] is an array of Strings with each element holding a student's first name
- **Exam1**[100], **HW1**[100], and **HW2**[100] are parallel arrays of Floats.
- **Low**, **High**, **N**, and **Index** are Integer variables.
- **Found** is a Boolean variable.
- **Student** is a String variable.

Professor Crabtree's pseudocode

```
1       Set N = 99, Set Low = 0, Set High = N
2       Set Found = 0
3       Set Index = Floor(N/2)
4       Write "Enter a student's name: "
5       Input Student
6       While (Found == 0) AND (Low <= High)
7               If Student < Names[Index] Then
8                       Set High = Index - 1
9                       Set Index = Floor((High + Low)/2)
10              Else
11                      If Student > Names[Index] Then
12                              Set Low = Index + 1
13                              Set Index = Floor((High + Low)/2)
14                      Else
15                              Set Found = 1
16                      End If
17              End If
18      End While
19      If Found == 0 Then
20              Write "Student record not found."
21      Else
22              Write "Student record for: "
23              Write First[Index] + " " + Names[Index]
24              Write "Exam 1: " + Exam1[Index]
25              Write "Homework 1: " + HW1[Index]
26              Write "Homework 2: " + HW2[Index]
27      End If
```


8.4 The Selection Sort

- The **selection sort** procedure is a more efficient way to sort data stored in an array than the bubble sort.
- Basic idea:
  - On 1st pass, locate the smallest array element and swap it with the first array element.
  - On 2nd pass, locate the second smallest element and swap it with the second element of the array.
  - On the 3rd pass, locate the next smallest element and swap it with the third element of the array and so forth...
  - If the array contains **N** elements, it will be completely sorted after, at most, **N – 1** passes.

General Pseudocode for the Selection Sort

```
1       Declare Array[K] As Float
2       Declare Littlest As Float
3       Declare K, N, Index, Temp As Integer
4       For (K = 0; K < N; K++)
5               Set Littlest = Array[K]
6               Set Index = K
7               For (J = K + 1; J <= N; J++)
8                       If Array[J] < Littlest Then
9                               Set Littlest = Array[J]
10                              Set Index = J
11                      End If
12              End For(J)
13              If K != Index Then
14                      Set Temp = Array[K]
15                      Set Array[K] = Array[Index]
16                      Set Array[Index] = Temp
17              End If
18      End For(K)
```

Applying the Selection Sort Technique

In this example assume we have an array that holds the ages of 200 students in Professor Crabtree's classes. This program will sort the array in ascending order.

The pseudocode is on the following slide.

To save space we will assume:

- **Ages**`[200]` is an array of `Integers` with each element holding the age of a student (in years).
- **Youngest**, **J**, **K**, **M**, **N**, **Temp** and, **Index** are `Integer` variables.

Example:

Use the selection sort to sort a large data set.

```
1       Set N = 199, Set M = 0, Set Temp = 0, Set K = 0
2       While K < N
3               Set Youngest = Ages[K]
4               Set Index = K
5               Set J = K + 1
6               While J <= N
7                       If Ages[J] < Youngest Then
8                               Set Youngest = Ages[J]
9                               Set Index = J
10                      End If
11                      Set J = J + 1
12              End While(J)
13              If K != Index Then
14                      Set Temp = Ages[K]
15                      Set Ages[K] = Ages[Index]
16                      Set Ages[Index] = Temp
17              End If
18              Set K = K + 1
19      End While(K)
20      Write "Ages sorted: "
21      While M < N + 1
22              Write Ages[M]
23              Set M = M + 1
24      End While(M)
```