Finding an item in a list – Is Connie in this class?

A simple method (algorithm) for finding an item in an array (list) is to start at the beginning of the array and check each item in turn. This is a linear search. The items in the list can be in any order.

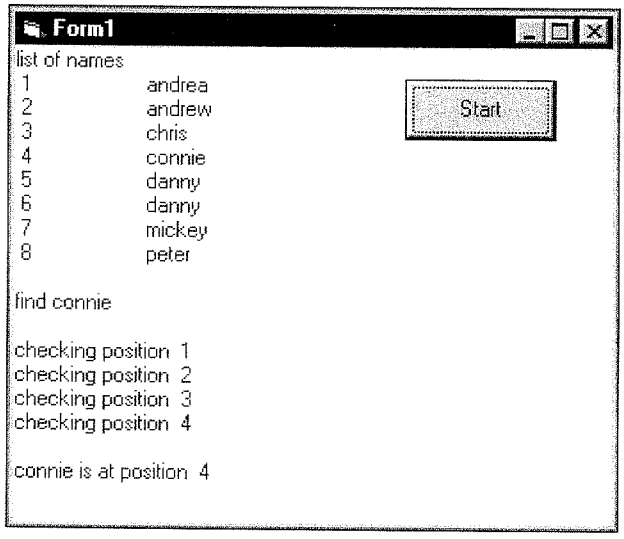
SAMPLE PROGRAM 1
Searching for Connie

The program below lists all the members of a class and then allows you to search for a particular student. The program:

- Sets up an array of eight students.
- Prints the students.
- Allows you to enter the name of the student required.
- Searches the list one item (element) at a time from the first.
- Stops when the person is found.

Procedure

- Create a new project, add a Start button, enter the code and try the program.




```
Option Explicit
Private Sub cmdStart_Click()
Dim name As String
Dim person(20) As String
Dim max As Integer
Dim item As Integer
Dim index As Integer
max = 8
Cls
Print "linear search"
person(1) = "andrea"
person(2) = "andrew"
person(3) = "chris"
person(4) = "connie"
person(5) = "danny"
person(6) = "danny"
person(7) = "mickey"
person(8) = "peter"
'****************************************
Print "list of names"
For Item = 1 To max
   Print Item, person(Item)
Next Item

'enter the person to be found
name = InputBox("enter a name to find ")
Print
Print "find "; name
Print

'search array of names
Index = 0
Do
   Index = Index + 1
   Print "checking position "; Index
   If name = person(Index) Then
      Print
      Print person(Index); " is at position "; Index
   End If
Loop Until name = person(Index) Or Index = max

End Sub
```

Exercises 1

1. Try to find Connie, then other students —the first and the last — to check the algorithm.
2. What happens if you attempt to find a person who is not on the list?
3. Change the program so the situation is covered where the person is not on the list. Include a new boolean variable, 'found', which is initially set to 'false'. When a person is found, 'found' is set to 'true'. After the search check if 'found' is 'true' or 'false', ie., found or not found.
4. Simulate a linear search by putting six students in a line. Check each student in turn to find "Sue".
5. Try to find Danny. What would you want the program to tell you?

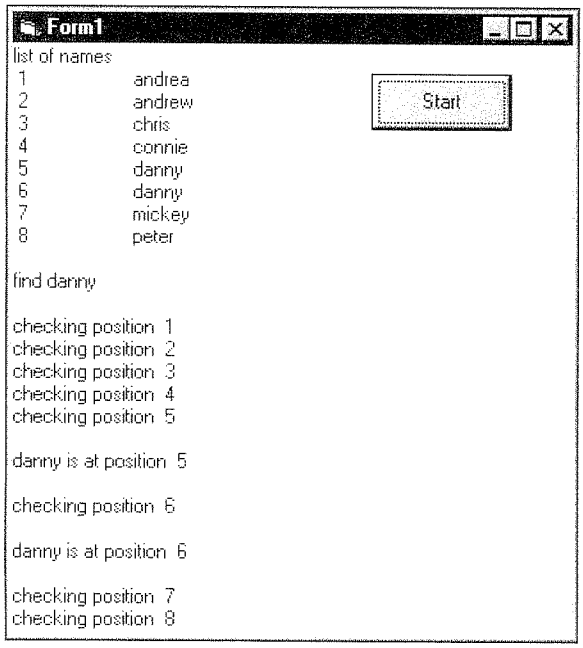
SAMPLE PROGRAM 2

Duplicate names!

Often we have an array of names which contains more than one Connie, or Danny in the case above. The program below is similar to the example above, but it searches the whole list each time.

Modify the previous program as shown below.

```
Option Explicit
Private Sub cmdStart_Click()
Dim name As String
Dim person(20) As String
Dim max As Integer
Dim item As Integer
Dim index As Integer
Max = 8
Cls
Print "linear search"
person(1) = "andrea"
person(2) = "andrew"
person(3) = "chris"
person(4) = "connie"
person(5) = "danny"
person(6) = "danny"
person(7) = "mickey"
person(8) = "peter"
```




```
Print "list of names"
For Item = 1 To 8
   Print Item, person(Item)
Next Item

'*************************************
'enter the person to be found
Index = 0
name = InputBox("enter a name to find ")
Print
Print "find "; name
Print

'search array of names
For Index = 1 To 8
   Print "checking position "; Index
   If name = person(Index) Then
      Print
      Print person(Index); " is at position "; Index
      Print
   End If
Next Index
End Sub
```

Exercises 2

1. Try and modify the second program to allow for the case where the person is not found in the list. (See Question 3 on previous page.)
2. Create an array of 20 random numbers in the range 1 to 100 and use the linear search to find a specific number. Use an input box to enter the number to be found.
3. Modify the program used, to generate 40 numbers in the range 1 to 10. Again use the linear search but now add to the program to count how many times your selected number occurs.
4. Create a list of 1000 random numbers without repeats in the range 1 to 1000. Using the timer function as used in the previous workshop, time how long it takes to find 999. Try 10000 numbers. Find 9999.
5. Give six cards, each with a random number, to a group of six students. Have another student pretend they are the computer, performing the algorithm with the six students.

Finding an item in a list more quickly — Is there a Connie in our school?

The linear search is satisfactory when the array of items is not large. If there were only 20 students in a class then the linear search may be convenient. If we have to search through the whole school of 800 students then the linear search may not be adequate, as we could have to search all the way through 800 students.

SAMPLE PROGRAM 1

Binary is better ... the HiLo game

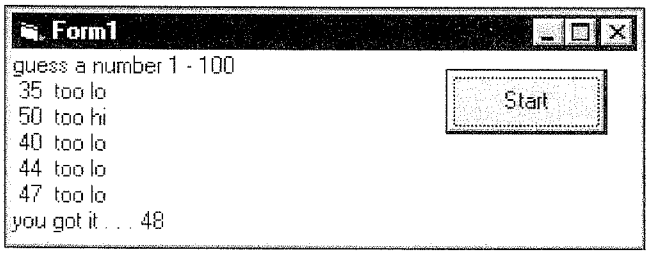
The binary search is much more efficient than the linear search, as it is able to find an item very quickly. The binary search is based on the strategy that is useful in the following guessing game. Here you have to guess a number between 1 and 100, where the number is selected by the computer.

The program:

- Generates a random number between 1 and 100.
- You guess the number.
- The computer tells you if you are too high or too low.
- You keep guessing until the number is found.

Procedure

- Create a new project, add a Start button, enter the code and try the program.




```
Option Explicit
Private Sub cmdStart_Click()
Dim guess As Integer
Dim randno As Integer
Cls
Print "guess a number 1 - 100"
Randomize
'gives a different sequence each time
randno = Int(100 * Rnd(1) + 1)
Do
   guess = InputBox("enter a randno 1 - 100")
   If guess = randno Then
      Print "you got it . . ."; randno
   End If
   If guess > randno Then
      Print guess; " too hi"
   End If
   If guess < randno Then
      Print guess; " too lo"
   End If
Loop Until guess = randno
End Sub
```

Exercises

1. Try the program. What strategy can you adopt to find the number as quickly as possible?
2. What is the minimum number of guesses you need to find the number?
3. Modify the program so you only allow the minimum number of guesses. Now the computer might have a chance of winning a game!
4. Modify the program so the game becomes Hot-Cold, so the message is not Too Hi or Too Lo but Very Hot, Hot, Cold, Very Cold, depending on how far you are from the number. What strategy would you adopt here to find the number? You, as the programmer, can decide what Hot, Cold means.

SAMPLE PROGRAM 2

The binary search

The binary search is based on the strategy used in the HiLo game in the previous section. In the binary search the items in the list must be arranged in order; for example, we need a class to be in alphabetical order. The binary search

- starts at the middle of the list
- checks if the guess is correct (you have found the item)
- checks if the guess is Too Hi (that is, after the item)
- checks if the guess is Too Lo (that is, before the item)
- discards half the remaining list at each guess when narrowing the search. For example, if when guessing a number between 1 and 100, you are told 50 is too Hi then the numbers 50 and above are rejected from the search.

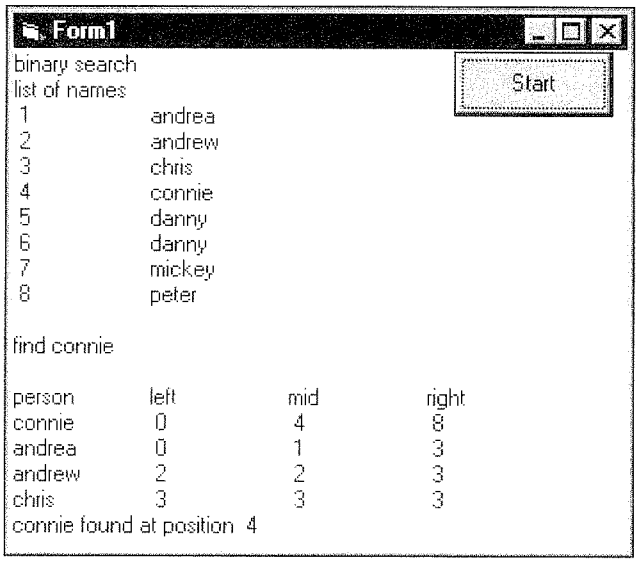
The program uses:

- left — to indicate the left index of the remaining list (initially 1)
- right — to indicate the right index of the remaining list (initially 100)
- mid — as the current position where you are searching (initially 50).

The binary search (cont.)

Procedure

- Create a new project, add a Start button, enter the code and try the program.




```
Option Explicit
Private Sub cmdStart_Click()
Dim name As String
Dim person(20) As String
Dim left As Integer
Dim right As Integer
Dim mid As Integer
Dim position As Integer
Dim max As Integer
Dim item As Integer
max = 8
Cls
Print "binary search"
person(1) = "andrea"
person(2) = "andrew"
person(3) = "chris"
person(4) = "connie"
person(5) = "danny"
person(6) = "danny"
person(7) = "mickey"
person(8) = "peter"

Print "list of names"
For item = 1 To 8
   Print item, person(item)
Next item

'************************************
'enter the person to be found
name = InputBox("enter a name to find ")
Print
Print "find "; name
Print

Print "person", "left", "mid", "right"

'search array of names
left = 0
right = max
position = 0
While left <= right
   mid = Int((left + right) / 2)
   Print person(mid), left, mid, right
   If name = person(mid) Then position = mid
   If name <= person(mid) Then right = mid - 1
   If name > person(mid) Then left = mid + 1
Wend

'************************************
'print result of search
If position > 0 Then
   Print name; " found at position "; position
Else
   Print name; " not found"
End If

End Sub
```

Exercises

1. What is the function of 'left' and 'right' in the binary search program?
2. When there are duplicate names, the above program locates the first of the names. How does the program do this?
3. Create a program which generates 20 random numbers in increasing order, between 1 and 99. Add to the program so it uses a binary search to find a number you select. Use an inputbox to enter the number. Print a message if the number is not found, indicating the nearest number before and after the selected number.
4. Create a list of 10 000 random numbers (without repeats) in the range 1 to 10 000. Using the VB timer function as used in the previous workshop, time how long it takes to find 9999? (If it exists.)

Sorting numbers into ascending order

Sorting is an important part of processing data. There are a number of important sorting algorithms that have been developed. These include the 'bubble' sort, 'selection' sort, 'insertion' sort, 'heap' sort, 'quick' sort and 'shell' sort.

In general, sorting programs 'compare' terms in an array, followed by a subsequent swapping of terms, if required. The comparison is based on

IF term1 > term2 THEN ...

Either then, or at a later stage, terms are 'swapped'. The swapping involves,

store = term1
term1 = term2
term2 = store

Some computing languages have a 'swap' command to do the swapping of terms directly.

Bubble sort is generally the first sorting algorithm introduced to students.

SAMPLE PROGRAM 1

Bubble sorting five random numbers

The program below:

- Generates five random two-digit numbers and assigns them to an array of integers.
- Bubble sorts the numbers into order.
- Prints the array at each stage of the number sort, so the operation of the sort can be seen.
- Prints the sorted numbers.

The bubblesort works by:

- Comparing adjacent terms (next to each other), starting from the first two terms (1 and 2 of the array).
- Swapping the first term for the second position if the second term is smaller.
- Comparing terms 2 and 3 and swapping the terms if required.
- Repeating the process with terms 3 and 4 and so on, until the largest term is at the end of the list.

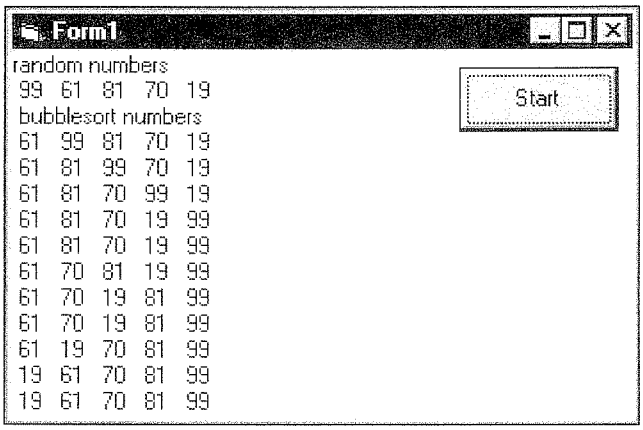
The overall result of the process is to move the largest term to the last position in the array, like 'bubbling' the largest bubble to the surface first. The process is repeated with the second largest term being 'bubbled' to the second last position, and so on.

Look at the output below and you will be able to see the sorting process.

Procedure

- Create a new project and add a Start button.
- Enter the code below and try the program.
- Check the output to see how the program works.

```
Option Explicit
Private Sub cmdStart_Click()
Dim first As Integer
Dim last As Integer
Dim max As Integer
Dim current As Integer
Dim item(20) As Integer
Dim temp As Integer
```




```
Cls
Randomize
Print "random numbers"
max = 5
For current = 1 To max
   item(current) = Int(Rnd(1) * 90 + 10)
   Print item(current); " ";
Next current
Print

Print "bubblesort numbers"
'initialise
first = 1
last = max
While last > first
   current = first
   While current < last
      If item(current) > item(current + 1) Then
         'swap current item with next item
         temp = item(current)
         item(current) = item(current + 1)
         item(current + 1) = temp
      End If
      current = current + 1
      'print the sort at each step
      For temp = 1 To max
         Print item(temp); " ";
      Next temp: Print
```

The bubble sort (cont.)

```
    Wend
    last = last - 1
Wend

For current = 1 To max
    Print item(current); " ";
Next current

End Sub
```

As well as numbers, words and codes can be sorted in a similar way. The computer bases the order on the ASCII codes, where 0<1<2 ... <A<B<C<D ... < a<b<c ... <z.

Exercises

1. Alter the program to sort the numbers into descending order. (Modify the IF statement.)
2. Modify the program to sort 10000 numbers, put the numbers in an array, and use the timer function to time the program. Remove the code, which prints the sorting process at each step, as this only slows the program down.
3. Modify the program to generate 100, two-character, lower-case letter codes. That is, use random ASCII codes (from 97–122), to generate codes like *bx, wf, ys*. (*See Workshop 9 in Book 1 on ASCII Codes*). Sort these codes into Book 1 alphabetical order.
4. Compare the time to sort two-digit numbers and two-digit codes.
5. Simulate the operation of the sort by assigning six random numbers to six students in a line. Have another student move along the line performing the 'sort' manually. The simulation could also be performed by individual students, sorting six randomly numbered cards.
6. FOR loops can be used instead of the WHILE loops, as the number of repeats is known for the sorting process. Replace the original sorting code with the new code below. Try the program.

```
Print "bubblesort numbers"
'initialise
first = 1

For last = max To first Step -1
    For current = 1 To last - 1
        If item(current) > item(current + 1) Then
            'swap current item with next item
            temp = item(current)
            item(current) = item(current + 1)
            item(current + 1) = temp
        End If
    Next current
        'print the sort at each step
        For temp = 1 To max
            Print item(temp); " ";
        Next temp: Print
Next last
```

Handy Hints: Program documentation

Make your program easier to understand and fix!

Documentation can be extrinsic or intrinsic. Extrinsic documentation is external to the program, for example development documents, installation guide, user manual, tutorial, reference manual. Intrinsic documentation is built into your computer code and includes:

- Comments at the start of sections of code to explain the function of the following code.
- The use of meaningful variable names so you know what the code is processing. Variable names (identifiers) are usually a compromise between long, fully understandable names, which hide the structure of a program or short names without meaning. You might use 'no_items' for 'number_of_items'
- Indenting of code so structures like REPEAT . . . UNTIL or IF . . . THEN . . . ENDIF stand out. The structures have a beginning and an end.
- Spaces in, or lines across, the code to separate sections of code doing different processes.
- Use of subprograms, modules or subroutines to break the program into logical parts.

Selection sort

The selection sort is similar to the bubble sort, as it moves the largest term to the last place in the list, then the second largest to second last place in the list, and so on. The selection sort is a useful sort where small arrays need to be sorted.

SAMPLE PROGRAM

Selection sorting six random numbers

The sample program below is similar to the one in the previous workshop. The program:

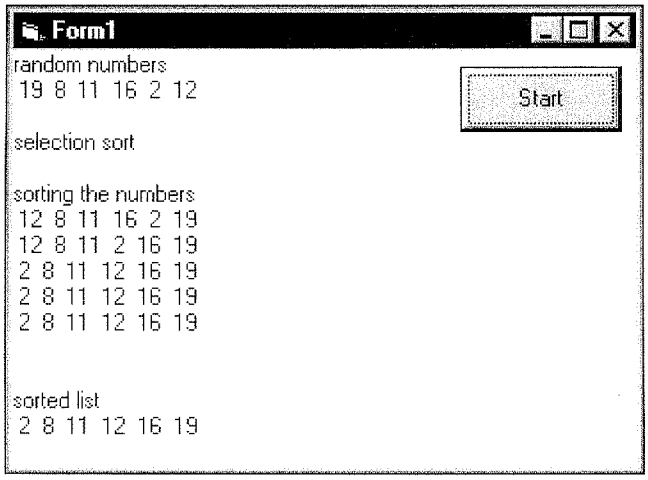
- Generates six random numbers in the range 1 to 20 and assigns them to an array.
- Prints the random numbers.
- Sorts the numbers, printing out the results of the sort at each stage.
- Prints the sorted numbers.

The program works by:

- Beginning at the start of the array and assigning the first term as 'largest'.
- Moving along the list, comparing 'largest' with each term in turn.
- Replacing 'largest' with the current term, if the current term is larger and recording the position.
- Swapping the largest term to the end of the list.
- Repeating the process by finding and swapping the second largest term to the second last position, and so on.

Procedure

- Create a new project; add a Start button.
- Enter the code and try the program.




```
Option Explicit
Private Sub cmdStart_Click()
Dim count As Integer
Dim item(10) As Integer
Dim last As Integer
Dim current As Integer
Dim position As Integer
Dim largest As Integer
Dim first As Integer
Dim max, temp As Integer
Cls
max = 6

'make list of random numbers
Print "random numbers"
For count = 1 To max
   item(count) = Int(20 * Rnd(1) + 1)
   Print item(count); " ";
Next count
Print
Print

first = 1
last = max

'*************************************
Print "selection sort": Print
Print "sorting the numbers"
Do While last > first
   current = first
   largest = item(current)
   position = current
   Do While current < last
      current = current + 1
      If item(current) > largest Then
         largest = item(current)
         position = current
      End If
   Loop
   temp = item(position)
   item(position) = item(last)
   item(last) = temp
   last = last - 1

   For count = 1 To max
      Print item(count); " ";
   Next count
   Print
Loop

'*************************************
Print: Print
Print "sorted list"

For count = 1 To max
   Print item(count); "";
```

The selection sort (cont.)

```
    Next count
    Print

    End Sub
```

Exercises

Repeat the exercises (listed below) from the previous workshop, but this time use the selection sort.

1. Modify the program to sort 10 000 numbers, and use the timer function to time the program. (*See Workshop 8, Question 2*). Remove the code, which prints the sorting process at each step, as this only slows the program down.
2. Modify the program to generate 100 two-character, lower-case letter codes. That is, use random ASCII codes (from 97–122), to generate codes like *bx*, *wf*, *ys*. Sort these codes into alphabetical order.
3. Compare the time to sort two-digit numbers and two-digit codes.
4. Simulate the operation of the sort by assigning six random numbers to six students in a line. Have another student move along the line performing the 'sort' manually. The simulation could also be performed by individual students, sorting six randomly numbered cards.
5. FOR loops can be used instead of the WHILE loops, as the number of repeats is known for the sorting process. Replace the original sorting code with the new code below. Try the program.

```
Print "selection sort": Print
Print "sorting the numbers"
For last = max To first Step -1
      current = first
      largest = item(current)
      position = current
      'find next largest term
      For current = 2 To last
         If item(current) > largest Then
            largest = item(current)
            position = current
         End If
      Next current
   'swap largest to "last"
      temp = item(position)
      item(position) = item(last)
      item(last) = temp

      For count = 1 To max
            Print item(count); "";
      Next count
      Print
Next last
```

Handy Hints: Program Testing

Check your program so it doesn't 'crash'!

To test a program properly you should anticipate all possible inputs and all program pathways.

- Use test data to check all inputs and pathways. Include input data, which is in the expected range, at boundaries between expected ranges, outside the expected range, the wrong type of input (for example a string instead of a number).
- Check program output against expected program output.
- Check loops. Check the type of loop you have used — a counted loop (FOR), a pre-test (guarded) loop (WHILE), which may not even be executed, or a post-test (unguarded) loop, which will be executed at least once.
- Check selections. Make sure you cover all possible alternatives when using IF and CASEWHERE statements.
- Check structures. Some parts of the program are sequential with one process following another. Other parts of programs are nested with structures inside structures, for example loops inside loops OR, IFs inside IFs. Be very careful and desk-check these structures carefully. Do they process as you intended?

Insertion sort

The insertion sort is like shuffling cards into order. The cards from the original random pack are taken one at a time and placed into a 'sorted' group of cards at the back of the pack.

SAMPLE PROGRAM 1

Sorting six random numbers

The insertion sort:

- Compares the last two numbers and if necessary swaps the largest number to the last position.
- Takes the third last term, **(nextpos)**, and stores it **(nextval)**.
- 'Shuffles' the following numbers forward, that is **item(current - 1) = item(current)**
- While **nextval < item(current + 1**, that is, until the correct position of the third-last term is found.
- Places the number in the correct position, **item(current) = nextval** ... and so on for the fourth-last term ...

The program:

- Generates six two-digit random numbers, assigning the numbers to an array.
- Prints the random numbers.
- Sorts the numbers into order, printing each stage of the sort.
- Prints the sorted numbers.

Procedure

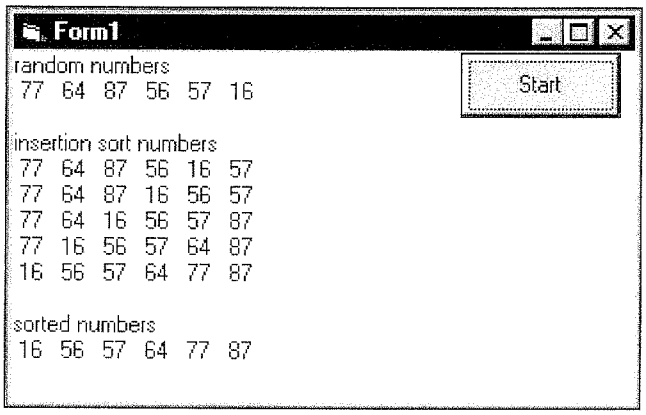
- Create a new project and add a Start button.
- Enter the code and try the program.
- Check the program output to see how the sort works.

```
Option Explicit
Private Sub cmdStart_Click()
Dim first As Integer
Dim last As Integer
Dim max As Integer
Dim current As Integer
Dim item(20) As Integer
Dim temp As Integer
Dim largest As Integer
Dim nextpos As Integer
Dim nextval As Integer

Cls
Randomize
Print "random numbers"
max = 6
For current = 1 To max
   item(current) = Int(Rnd(1) * 90 + 10)
   Print item(current); " ";
Next current
Print: Print

'*****************************************
Print "insertion sort numbers"
'initialise
first = 1
last = max
nextpos = last - 1

While nextpos >= first
   nextval = item(nextpos)
   current = nextpos
```



```
    While (current < last) And (nextval > item(current + 1))
       current = current + 1
       item(current - 1) = item(current)
    Wend
    item(current) = nextval
    nextpos = nextpos - 1

       'print the sort at each step
       For temp = 1 To max
          Print item(temp); " ";
       Next temp: Print
Wend

Print: Print "sorted numbers"
For current = 1 To max
   Print item(current); " ";
Next current
Print

End Sub
```

Exercises

1. Modify the program to sort 10 000 numbers, and use the timer function to time the program. (*See Workshop 8, Question 2*). Remove the code, which prints the sorting process at each step, as this only slows the program down. Compare the speed of the three sorts. Which sorting algorithm is the best? Why?
2. Modify the program to generate 100 two-character, lower-case letter codes. That is, use random ASCII codes (from 97–122), to generate codes like *bx*, *wf*, *ys*. Sort these codes into alphabetical order.
3. Compare the time to sort two-digit numbers and two-digit codes.
4. Simulate the operation of the sort by assigning six random numbers to six students in a line. Have another student move along the line performing the 'sort' manually. The simulation could also be performed by individual students, sorting six randomly numbered cards.
5. A FOR loop can be used instead of a WHILE loops, as the number of repeats is known for the sorting process. Replace the original sorting code with the new code below. Try the program.

```
Print "insertion sort numbers"
'initialise
first = 1
last = max

For nextpos = last - 1 To first Step -1
   nextval = item(nextpos)
   current = nextpos

   'shuffle items forward
   While (current < last) And (nextval > item(current + 1))
      current = current + 1
      item(current - 1) = item(current)
   Wend
   item(current) = nextval

      'print the sort at each step
      For temp = 1 To max
         Print item(temp); " ";
      Next temp: Print
Next nextpos
```