Basic Programming Algorithms

1-D Arrays

- 1-D fixed-size array

```
Dim arr(5) As Double ‘0-base, 6 elements
Dim arr(0 to 5) As Double ‘0-base, 6 elements
Dim arr(1 to 5) As Double ‘1-base, 5 elements
Option Base 1
```

- 1-D dynamic array

```
Dim arr() As Double
…
Redim arr(5) ‘allocate 6 elements to the array
…
Redim Preserve arr(10) ‘increase the array size to 11 while
    preserving the values
Redim Preserve arr(1 to 10) ‘this will cause an out of range error
```

2-D Arrays

- 2-D fixed-size array
  Dim arr(5, 10) As Double ‘6 rows, 11 columns
- 2-D dynamic array
  Dim arr( ) As Double
  …
  Redim arr(5, 10)
  …
  Redim Preserve arr(5, 20) ‘when using Preserve keyword, you can only change the size of the highest dimension

2-D Arrays

Dim A (3, 7)
‘the array has 4 rows, 8 cols, a total of 32 elements

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1,0	1,1	1,2	1,3				
2,0	2,1	2,2	2,3				
3,0	3,1	3,2	3,3				

0,0	1,0	2,0	3,0	0,1	1,1	2,1	3,1	0,2	1,2	2,2	…

Working with 2-D Arrays

```
Dim A(3, 7) As Integer '2-D array with 32 elements
Dim B(31) As Integer '1-D array with 32 elements
Dim iIndex As Long
Dim irow As Integer, icol As Integer

For icol = 0 to 7
   For irow = 0 to 3
        iIndex = icol * 4 + irow
        A(irow, icol) = iIndex
        B(iIndex) = iIndex
   Next
Next
```

Array Size

- UBound
- LBound

```
Dim A()
Redim A(5)
Redim A(UBound(A) + 1) 'increase the size of the array by 1

Dim A (1 To 100, 0 To 3, -3 To 4)

'Statement               Return Value
UBound(A, 1)             100
UBound(A, 2)             3
UBound(A, 3)             4
```

Calculate Mean Value

```
Dim i As Integer
Dim n As Integer
Dim inarr() As Double
Dim arr_sum As Double, arr_avg As Double
‘set the value of n
…
Redim inarr(1 to n)
‘Redim Preserve inarr(1 to n)
‘Initialize inarr
…
arr_sum = 0
For i = 1 to n
    arr_sum = arr_sum + inarr(i)
Next

arr_avg = arr_sum / n
```

Swap a pair of numbers

```
Dim a As Integer, b As Integer
Dim tempval As Integer

tempval = a
a = b
b = tempval
```

Subroutines & Functions

Public Sub Swap(a As Integer, b As Integer)

- Scope
- Sub or Function
- Name of sub (function)
- Argument list (called by value versus by reference)
- Return data type

Subroutine and Function Example 1

```
Option Explicit

Sub test()
   Dim response As String
   Dim dArea As Double

   response = InputBox("Enter the radius of a cirle")
   If Len(response) = 0 Then Exit Sub 'User press cancel

   MsgBox "Area of the cirlce is " & Area_of_Circle(CDbl(response))
End Sub

Public Function Area_of_Circle(r As Double) As Double
   Dim pi
   pi = 4 * Atn(1) 'pi equals 4 times the arctangent of 1
   Area_of_Circle = pi * r * r
End Function
```

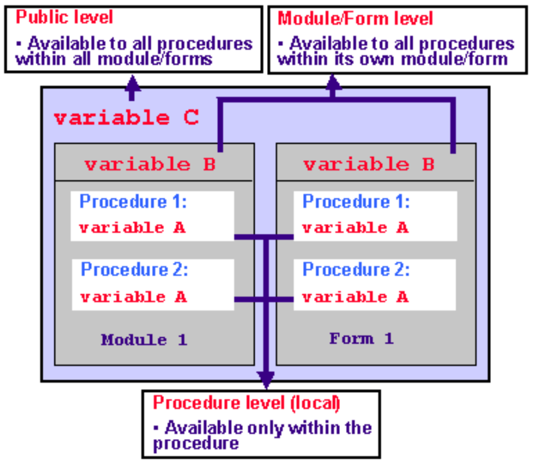
Subroutine and Function Example 2

```
Sub test()
   Dim a As String, b As String
   a = "First"
   b = "Second"
   MsgBox "Before swap: a is " & a & ", b is " & b

   Swap_Values a, b      'or Call Swap_Value(a, b)

   MsgBox "After swap: a is " & a & ", b is " & b
End Sub

Public Sub Swap_Values(ByRef Item1 As Variant, ByRef Item2 As Variant)
   Dim tempval As Variant

   tempval = Item1
   Item1 = Item2
   Item2 = tempval
End Sub
```

Variable Scope

- Scope
  - Public
  - Private

```
Dim B As Integer

Private Sub Sub1()
   Dim A As Integer
End Sub
```



Debug

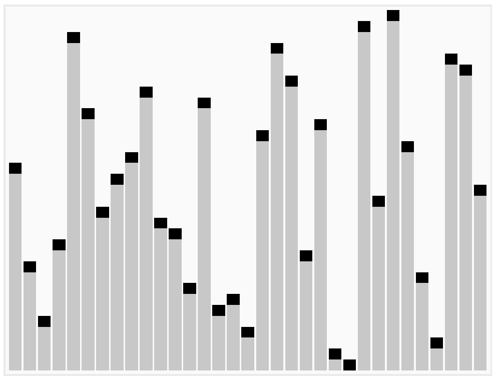
- VBA IDE Debug Tool
  - Breakpoint
  - Step
  - Variable values browsing
- Debug.print and Debug.pause
- Error handler

Sorting Algorithms

- Rearrange a list of elements in certain order.
- Sorting order:
  - Numerical vs. lexicographical order
  - Ascending vs. descending order

Quick Sort

- *Quicksort* is a divide and conquer algorithm which relies on a *partition* operation: to partition an array, we choose an element, called a *pivot*, move all smaller elements before the pivot, and move all greater elements after it. We then recursively sort the lesser and greater sublists.



Pseud-code

```
function quicksort(array)
    var list less, pivotList, greater

    if length(array) ≤ 1
        return array

    select a pivot value pivot from array
    for each x in array
        if x < pivot then add x to less
        if x = pivot then add x to pivotList
        if x > pivot then add x to greater

    return concatenate(quicksort(less), pivotList, quicksort(greater))
```

Stack & Queue

- Stack
  - First in, last out
- Queue
  - First in, first out

- Stack: A fixed amount of memory used by program to preserve local variables and arguments during procedure calls.
- Stack Overflow: Stack memory is full!

QuickSort Pesudo-code

```
function partition(array, left, right, pivotIndex)
    pivotValue := array[pivotIndex]
    swap( array, pivotIndex, right) // Move pivot to end
    storeIndex := left
    for i from left to right-1
        if array[i] <= pivotValue
            swap( array, storeIndex, i)
            storeIndex := storeIndex + 1
    swap( array, right, storeIndex) // Move pivot to its final place
    return storeIndex

function quicksort(array, left, right)
    if right > left
        select a pivot index (e.g. pivotIndex := left)
        pivotNewIndex := partition(array, left, right, pivotIndex)
        quicksort(array, left, pivotNewIndex-1)
        quicksort(array, pivotNewIndex+1, right)
```