











Syllabus Content:

4.1.2 Algorithms (Bubble sort & insertion sort)

-  write a binary search algorithm to solve a particular problem
-  show understanding of the conditions necessary for the use of a binary search
-  show understanding of how the performance of a binary search varies according to the number of data items
-  write an algorithm to implement an insertion sort
-  write an algorithm to implement a bubble sort
-  show understanding that performance of a sort routine may depend on the initial order of the data and the number of data items
-  write algorithms to find an item in each of the following: linked list, binary tree, hash table
-  write algorithms to insert an item into each of the following: stack, queue, linked list, binary tree, hash table
-  write algorithms to delete an item from each of the following: stack, queue, linked list
-  show understanding that different algorithms which perform the same task can be compared by using criteria such as time taken to complete the task and memory used

Binary Search:

A binary search is most efficient if **List** is already **sorted**. The value of the middle item in the list is first tested to check if it matches the required item, and half of the list that **does not** contain the item is then discarded. Then in next step, value is again checked from the middle in remaining half of list and if not found again half of list is discarded. This is repeated until the item is found or nothing is left in List to check.

To find the letter W using a binary search there could be just three comparisons.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
												-							-			-			
												W							W			W			
												1							2			3			

Binary search takes far few comparisons compared to **Linear search** which checks each and every item one by one.

Below is Identifier Table and Pseudocode for **Binary Search Algorithm**:

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
item	Item to be found
found	Flag to show when item has been found

```

DECLARE myList      : ARRAYS[0:8] OF INTEGER
DECLARE upperBound  : INTEGER
DECLARE lowerBound  : INTEGER
DECLARE index       : INTEGER
DECLARE found       : BOOLEAN
DECLARE item        : INTEGER

upperBound ← 8
lowerBound ← 0
top ← upperBound

OUTPUT ("Please Input Item to be found")
INPUT item
found = False
REPEAT
    index = INT( (upperbound + lowerbound)/2 )

    IF item = myList[index]
        THEN found = TRUE
    ELSEIF item > myList[index]
        THEN lowerBound = index + 1
    ELSEIF item < myList[index]
        THEN lowerBound = index - 1
    END IF
UNTIL (found = TRUE) OR (lowerbound = upperbound)

IF found = TRUE
    THEN
        OUTPUT ("ITEM found")
    ELSE
        OUTPUT ("ITEM NOT found")
    END IF

```

Bubble Sort

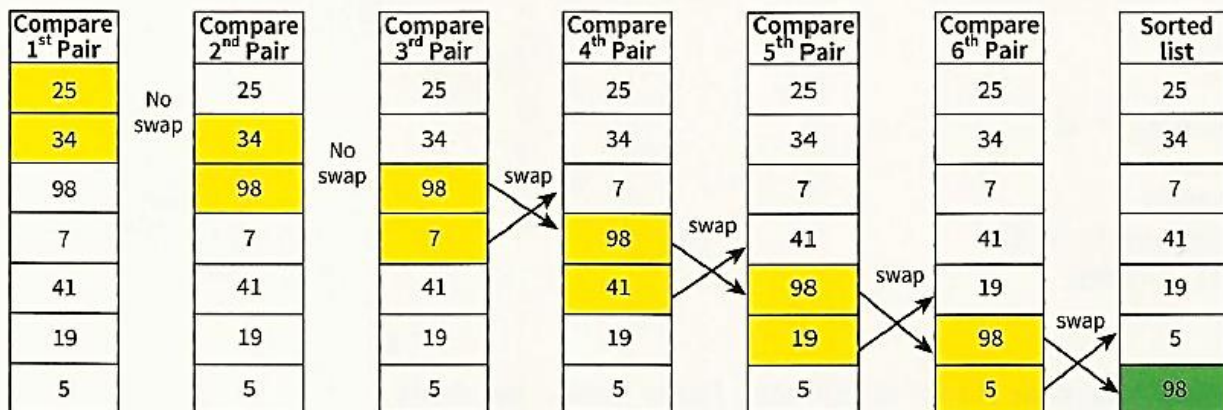


Figure 11.12 Swapping values working down the array

When we have completed the first pass through the entire array, the largest value is in the correct position at the end of the array. The other values may or may not be in the correct order.

We need to work through the array again and again. After each pass through the array the next largest value will be in its correct position, as shown in Figure below.

Original list	After pass 1	After pass 2	After pass 3	After pass 4	After pass 5	After pass 6
25	25	25	7	7	7	5
34	34	7	25	19	5	7
98	7	34	19	5	19	19
7	41	19	5	25	25	25
41	19	5	34	34	34	34
19	5	41	41	41	41	41
5	98	98	98	98	98	98

Figure 11.13 States of the array after each pass

In effect we perform a loop within a loop, a nested loop. This method is known as a **bubblesort**. The name comes from the fact that smaller values slowly rise to the top, like bubbles in a liquid.

KEY TERMS

Bubble sort: a sort method where adjacent pairs of values are compared and swapped

Bubble Sort Algorithm:

```

DECLARE myList      : ARRAYS[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index      : INTEGER
DECLARE swap       : BOOLEAN
DECLARE temp       : INTEGER
DECLARE top        : INTEGER
upperBound ← 8
lowerBound ← 0
top ← upperBound
REPEAT
  FOR index = lowerBound TO top - 1
    Swap ← FALSE
    IF myList [index] > myList [index + 1]
      THEN temp ← myList[index]
           myList[index] ← myList[index + 1]
           myList[index + 1] ← temp
           swaps ← TRUE
    END IF
  NEXT
  top ← top - 1
UNTIL (NOT swap) OR (top = 0)
    
```



Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
swap	Flag to show when swaps have been made
top	Index of last element to compare
temp	Temporary storage location during swap

Bubble Sort Algorithm using VB Console Mode:

In this tutorial, i will teach you how to create a program for bubble sorting using vb.net console. We all know that bubble sort is a sorting algorithm that is repeatedly searching through lists that need to be sorted, comparing each pair of items and swapping them if they are in the wrong order.

```

Module Module1
    Sub Main()
        Dim myList() As Integer = New Integer() {70, 46, 43, 27, 57, 41, 45, 21, 14}
        Dim index, top, temp As Integer
        Dim Swap As Boolean
        top = myList.Length - 1

        Do
            Swap = False
            'LOOP can work fine without STEP also
            For index = 0 To top - 1 Step 1 'STEP is a keyword to increment in loop
                If myList(index) > myList(index + 1) Then
                    temp = myList(index)
                    myList(index) = myList(index + 1)
                    myList(index + 1) = temp
                    Swap = True
                End If
            Next
            top = top - 1
        Loop Until (Not Swap) Or (top = 0)
        'Output The Sorted Array
        For index = 0 To myList.Length - 1
            Console.WriteLine(myList(index) & " ")
        Next
        Console.ReadKey()
    End Sub
End Module

```

Or we can get values from user for sorting and make a bubblesort Program

Now, let's start this tutorial!

1. Let's start with creating a Console Application for this tutorial by following the following steps in Microsoft Visual Studio: **Go to File**, click **New Project**, and choose **Console Application**.
2. Name your project as **bubbleSort** as your module.

Note: This is a console application so we cannot have visual controls for this tutorial.

3. Add the following code in your module.

1. Module bubbleSort

2. Sub Main()

4. Make a function named **sorting**. This will automatically sort the inputted elements that we will create in Sub Main.

```

1. Sub sorting(ByVal x() As Integer, ByVal upperbound As Integer)
2. Dim index, lowerbound, temp As Integer
3. For index = 0 To upperbound - 1
4.     For lowerbound = index + 1 To upperbound - 1
5.         If x(index) > x(lowerbound) Then
6.             temp = x(index)
7.             x(index) = x(lowerbound)
8.             x(lowerbound) = temp
9.         End If
10.    Next
11. Next
12. End Sub

```

5. For entering number of elements, put this code below.

```

1. Console.WriteLine("Bubble Sorting")
2. Console.WriteLine()
3. Dim num, i As Integer
4. Console.Write("Enter Number of Elements: ")
5. num = CInt(Console.ReadLine)
6. Dim arr(num) As Integer
7. Console.WriteLine()
8. For i = 0 To num - 1
9.     Console.Write("Enter Element(" & (i + 1) & "): ")
10.    arr(i) = CInt(Console.ReadLine)
11. Next

```

6. For printing the inputted elements above, put this code below.

```

1. Console.WriteLine()
2. Console.WriteLine("Inputted Elements")
3. Console.WriteLine()
4. For i = 0 To num - 1
5.     Console.WriteLine("Element in (" & i & "): " & arr(i))
6. Next

```

7. Lastly, we will code for the sorting of elements (bubble sort), put this code below.

```

1. Console.WriteLine()
2. sorting(arr, num)
3. Console.WriteLine("Sorted Elements")
4. Console.WriteLine()
5. For i = 0 To num - 1
6.     Console.WriteLine("Element in (" & i & "): " & arr(i))
7. Next
8. Console.ReadLine()

```

Total Code Together: (Code can be copied and tried in VB)

```

Module Module1
Sub sorting(ByVal x() As Integer, ByVal upperbound As Integer) 'X() is declared array
Dim index, lowerbound, temp As Integer

    For index = 0 To upperbound - 1
        For lowerbound = index + 1 To upperbound - 1
            If x(index) > x(lowerbound) Then
                temp = x(index)
                x(index) = x(lowerbound)
                x(lowerbound) = temp
            End If
        Next
    Next
End Sub

Sub Main()
    Console.WriteLine("Bubble Sorting")
    Console.WriteLine()
    Dim num, count As Integer
    Console.Write("Enter Number of Elements: ")
    num = CInt(Console.ReadLine)
    Dim array(num) As Integer 'Array Made to insert values to be sorted
    Console.WriteLine()

    For count = 0 To num - 1 'LOOP to Insert Values in Array
        Console.Write("Enter Element(" & (count + 1) & "): ")
        array(count) = CInt(Console.ReadLine)
    Next

    Console.WriteLine()
    Console.WriteLine("Inputted Elements")
    Console.WriteLine()

    For count = 0 To num - 1 'LOOP to Show Inserted Values
        Console.WriteLine("Element in (" & count & "): " & array(count))
    Next

    Console.WriteLine()

    sorting(array, num) 'SORTED Procedure applied on Array and Upperbound of Array
    Console.WriteLine("Sorted Elements")
    Console.WriteLine()

    For i = 0 To num - 1 ' To Display Sorted Elements
        Console.WriteLine("Element in (" & i & "): " & array(i))
    Next
    Console.ReadLine()

End Sub
End Module
3

```

Output:

```

file:///C:/Users/Nile/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Console...
Bubble Sorting
Enter Number of Elements: 4
Enter Element(1): 5
Enter Element(2): 9
Enter Element(3): 7
Enter Element(4): 3

Inputted Elements
Element in (0): 5
Element in (1): 9
Element in (2): 7
Element in (3): 3

Sorted Elements
Element in (0): 3
Element in (1): 5
Element in (2): 7
Element in (3): 9
    
```

Insertion sort:

Imagine you have a number of cards with a different value printed on each card. How would you sort these cards into order of increasing value?

You can consider the pile of cards as consisting of a sorted part and an unsorted part. Place the unsorted cards in a pile on the table. Hold the sorted cards as a pack in your hand. To start with only the first (top) card is sorted. The card on the top of the pile on the table is the next card to be inserted. The last (bottom) card in your hand is your current card.

		Index of element being checked											
myList		1	2	3	4	5	6	7	8				
[0]	27	19	19	19	19	16	16	16	16	16	16	16	16
[1]	19	27	27	27	27	19	19	19	19	16	16	16	16
[2]	36	36	36	36	36	27	27	27	21	21	19	19	19
[3]	42	42	42	42	42	36	36	36	27	27	21	21	21
[4]	16	16	16	16	16	42	42	42	36	36	27	27	27
[5]	89	89	89	89	89	89	89	89	42	42	36	36	36
[6]	21	21	21	21	21	21	21	21	89	89	42	42	42
[7]	16	16	16	16	16	16	16	16	16	16	89	89	55
[8]	55	55	55	55	55	55	55	55	55	55	55	55	89

Figure shows the sorted cards in your hand as blue and the pile of unsorted cards as white. The next card to be inserted is shown in red. Each column shows the state of the pile as the cards are sorted.

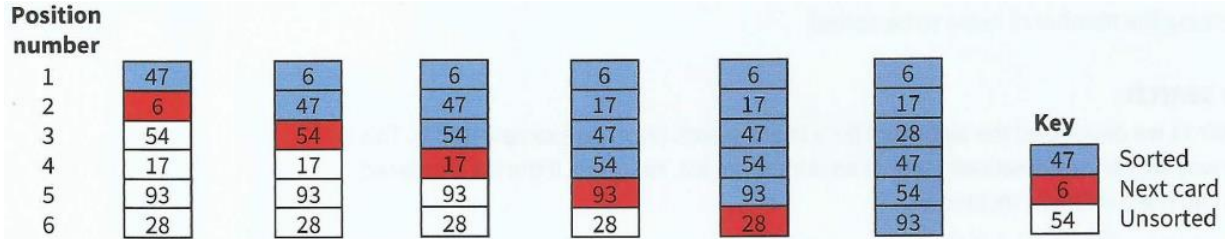


Figure 23.01 Sorting cards

Insertion Sort Algorithm

Identifier Table for Insertion sort:

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
key	Element being placed
place	Position in array of element being moved

```

DECLARE myList      : ARRAYS[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index      : INTEGER
DECLARE key        : INTEGER
DECLARE place      : INTEGER
DECLARE temp       : INTEGER
upperBound ← 8
lowerBound ← 0
top ← upperBound
REPEAT
  FOR index = lowerBound + 1 TO upperbound
    Key ← myList [index]
    Place ← index - 1
    IF myList [index] > key
      THEN
        WHILE place >= lowerbound AND myList[Place] > key
          temp ← myList[place + 1]
          myList[place + 1] ← myList[place]
          myList[place ] ← temp
          place ← place - 1
        ENDIF
      ENDIF
    NEXT index
  
```


Insertion sort in VB Console Mode:

```

Module Module1
    Sub insertionSort(ByVal dataset() As Integer, ByVal n As Integer)
        Dim i, j As Integer
        For i = 1 To n - 1 Step 1
            Dim pick_item As Integer = dataset(i)
            Dim inserted As Integer = 0
            j = i - 1
            While (j >= 0 And inserted <> 1)
                If (pick_item < dataset(j)) Then
                    dataset(j + 1) = dataset(j)
                    j -= 1
                    dataset(j + 1) = pick_item
                Else : inserted = 1
                End If
            End While
        Next
    End Sub
    Sub Main()
        Dim arr() As Integer = New Integer() {100, 12, 320, 34, 45, 90}
        'sort the array using insertion sort
        insertionSort(arr, arr.Length)
        Dim i As Integer
        For i = 0 To arr.Length - 1
            Console.WriteLine(arr(i))
        Next
        Console.ReadLine() 'wait for keypress
    End Sub
End Module

```

Output:

```

Sub Main()
    Dim arr() As Integer = New Integer() {100, 12, 320, 34, 45, 90}
    'sort the array using insertion sort
    insertionSort(arr, arr.Length)

    Dim i As Integer
    For i = 0 To arr.Length - 1
        Console.WriteLine(arr(i))
    Next

    Console.ReadLine() 'wait for keypress
End Sub

```

```

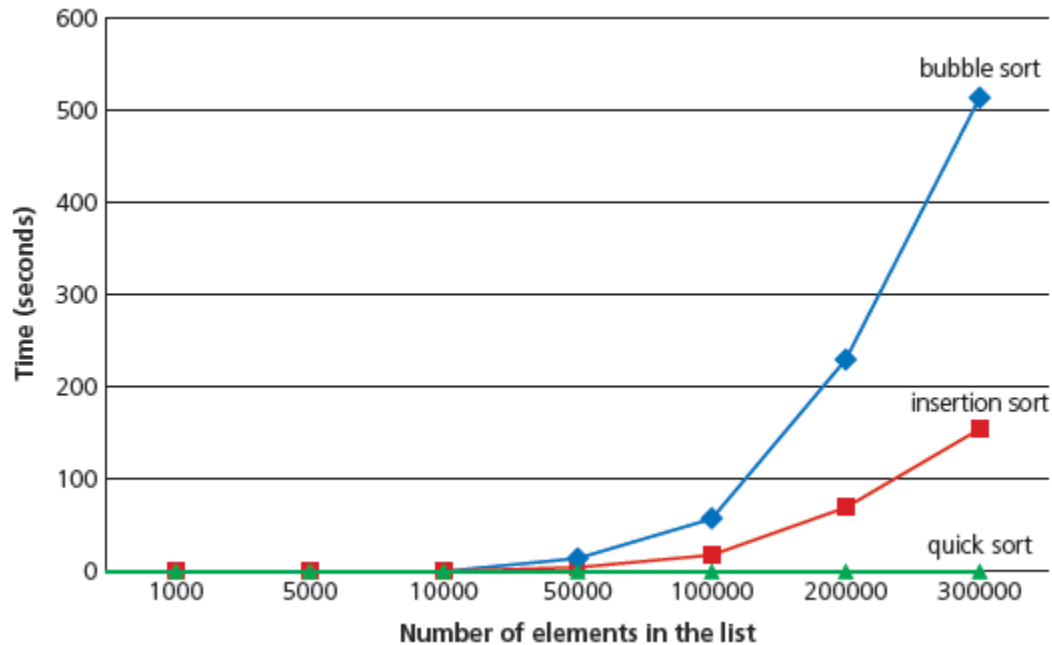
Sub insertionSort(ByVal dataset() As Integer)
    Dim i, j As Integer
    For i = 1 To n - 1 Step 1
        Dim pick_item As Integer = dataset(i)
        Dim inserted As Integer = 0
        j = i - 1

```

```

file:///C:/Users/Nile/documents/visual studio 2010/Projects/Bubble Sort/Bubble Sort/bin/
12
34
45
90
100
320

```



Performance of Sorting Algorithms

As the number of elements in List increases, the time taken to sort the list increases. It has been observed that when number of items in list increases, the performance of **bubble sort** deteriorates faster than **insertion sort**

References:

Computer Science AS & A Level Coursebook by Sylvia Langfield & Dave Duddell

Computer Science Teacher's Resource

Computer Science AS & A level by HODDER EDUCATION

<https://www.dotnetperls.com/dictionary-vbnet>

http://www.worldbestlearningcenter.com/index_files/vb.net-example-insertion-sort.htm