

Syllabus Content:

19.1 Algorithms part:1 (Linear and Binary search, Bubble sort & insertion sort)



Show understanding of linear and binary searching methods

Notes and guidance

- Write an algorithm to implement a linear search
- Write an algorithm to implement a binary search
- The conditions necessary for the use of a binary search
- How the performance of a binary search varies according to the number of data items



Show understanding of insertion sort and bubble sort methods

Notes and guidance

- Write an algorithm to implement an insertion sort
- Write an algorithm to implement a bubble sort
- Performance of a sorting routine may depend on the initial order of the data and the number of data items

Linear Search:

Linear search is a method of searching a list in which each element of an array is checked in order, from the lower bound to the upper bound, until the item is found, or the upper bound is reached.

The pseudocode linear search algorithm and identifier table to find if an item is in the 1D array(myList) is given below.

```

DECLARE count, num As Integer
DECLARE found As Boolean = False
//Creating array to search item (Free notes @ www.majidtahir.com)
DECLARE Mylist() As Integer = {4, 2, 8, 17, 9, 3, 7, 12, 34, 21}

OUTPUT ("Please enter any integer to be checked in List")
INPUT num
For count = 0 To 9
  If item = Mylist(count) Then
    found = True
  End If
Next
If found = True Then
  OUTPUT ("Item Found = ", num)
Else
  OUTPUT ("Item Found is unsuccessful")
End If

```

Sample VB program of linear search:

```

Dim count, num As Integer
Dim found As Boolean = False
Dim Mylist() As Integer = {4, 2, 8, 17, 9, 3, 7, 12, 34, 21} 'Create array to search
item

    Console.WriteLine("please enter an integer to be found")
    num = Console.ReadLine()
    For index = 0 To 9
        If item = Mylist(count) Then
            found = True
        End If
    Next
    If found = True Then
        Console.WriteLine("Item Found = " & item)
    Else
        Console.WriteLine("Item Found is Unsucessful")
    End If

```

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 count       : INTEGER
DECLARE found       : BOOLEAN
DECLARE item        : INTEGER
    upperBound ← 8
    lowerBound ← 0
OUTPUT ("Please Input Item to be found")
INPUT item
found = False
REPEAT
    count = INT( (upperbound + lowerbound)/2 )
    IF item = myList[index]
        THEN found = TRUE
    ELSEIF item > myList[index]
        THEN lowerBound = index + 1
    ELSEIF item < myList[index]
        THEN upperBound = index - 1
    END IF
UNTIL (found = TRUE) OR (lowerbound = upperbound)
IF found = TRUE
    THEN
        OUTPUT ("ITEM found")
    ELSE
        OUTPUT ("ITEM NOT found")
END IF

```

Sample VB Program of Binary Search:

```

Dim mylist() As Integer = {11, 22, 33, 44, 55, 66, 77, 88, 99, 110}
Dim upperbound, lowerbound, count As Integer
Dim item As Integer
Dim found As Boolean = False
upperbound = 10
lowerbound = 0
Console.WriteLine("please input num to be found")
item = Console.ReadLine()
Do
    count = Int((upperbound + lowerbound) / 2)
    If item = mylist(index) Then
        found = True
    ElseIf item > mylist(index) Then
        lowerbound = index + 1
    Else upperbound = index - 1
    End If
Loop Until (found = True) Or upperbound = lowerbound

If found = True Then
    Console.WriteLine("Item found: " & item)
Else
    Console.WriteLine("Item not found")
End If

```

Bubble Sort

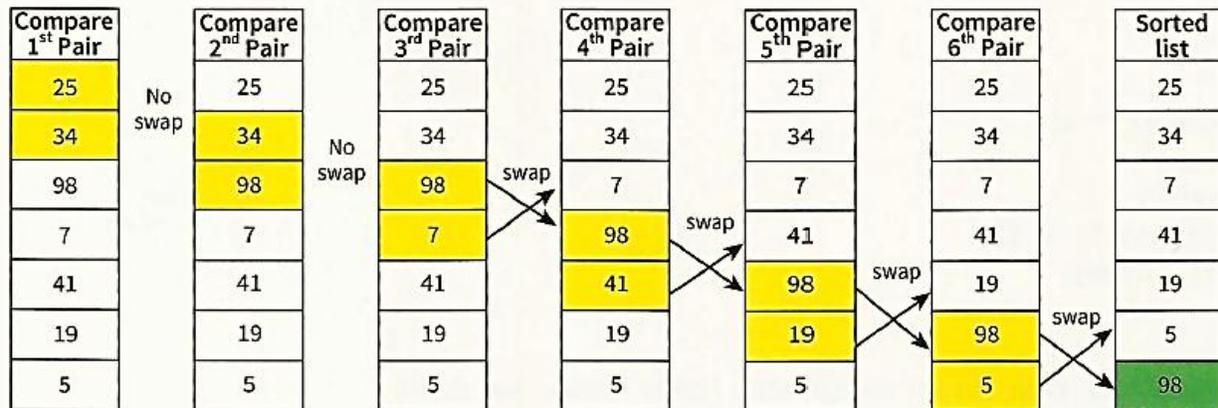


Figure 11.12 Swapping values working down the array

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
        swap ← 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!

- 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**.
- Name your project as **bubbleSort** as your module.
Note: This is a console application so we cannot have visual controls for this tutorial.
- Add the following code in your module.

```

1. Module bubbleSort
2. Sub Main()
3.     4. Make a function named sorting. This will automatically sort the inputted elements that
       we will create in Sub Main.
4.     Sub sorting(ByVal x() As Integer, ByVal upperbound As Integer)
5.         Dim index, lowerbound, temp As Integer
6.         For index = 0 To upperbound - 1
7.             For lowerbound = index + 1 To upperbound - 1
8.                 If x(index) > x(lowerbound) Then
9.                     temp = x(index)
10.                    x(index) = x(lowerbound)
11.                    x(lowerbound) = temp
12.                End If
13.            Next
14.        Next
15.    End Sub

```

- 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

```

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 |
| [1] | 19 | 27 | 27 | 27 | 27 | 19 | 19 | 19 | 19 | 16 | 16 | 16 |
| [2] | 36 | 36 | 36 | 36 | 36 | 27 | 27 | 27 | 21 | 21 | 19 | 19 |
| [3] | 42 | 42 | 42 | 42 | 42 | 36 | 36 | 36 | 27 | 27 | 21 | 21 |
| [4] | 16 | 16 | 16 | 16 | 16 | 42 | 42 | 42 | 36 | 36 | 27 | 27 |
| [5] | 89 | 89 | 89 | 89 | 89 | 89 | 89 | 42 | 42 | 36 | 36 | 36 |
| [6] | 21 | 21 | 21 | 21 | 21 | 21 | 21 | 89 | 89 | 42 | 42 | 42 |
| [7] | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 89 | 89 | 55 |
| [8] | 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.

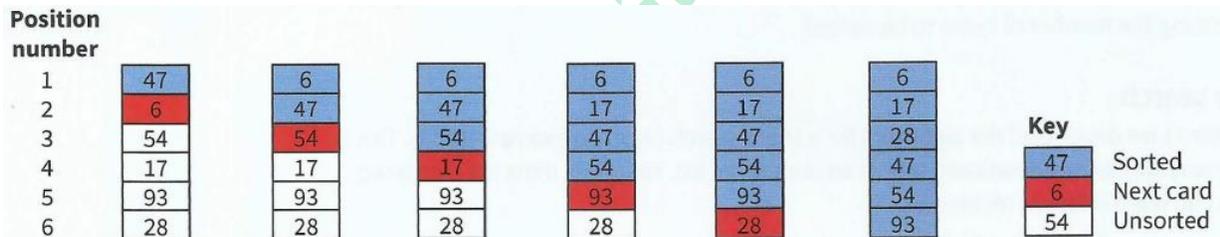


Figure 23.01 Sorting cards

Insertion Sort Algorithm

Identifier Table for Insertion sort:

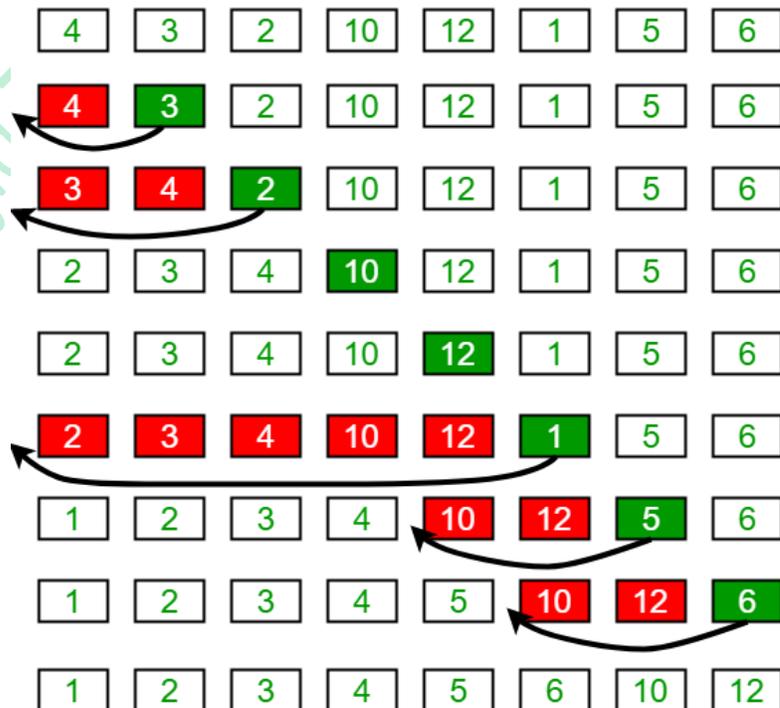
| Identifier | Description |
|------------|--|
| myList | Array to be searched |
| upperBound | Upper bound of the array |
| lowerBound | Lower bound of the array |
| index | Pointer to current array element |
| key | Element being placed |
| place | Position in array of element being moved |

```

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 [Place] > 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
      NEXT index

```

Insertion Sort Execution Example



Insertion sort in VB Console Mode:

```

Sub Main()
    Console.WriteLine("Inserion Sort Program")
    Dim mylist() As Integer = {70, 46, 43, 27, 57, 41, 45, 21, 14}
    Dim count As Integer 'to access array from left to right
    Dim current As Integer ' to hold the current item to find place in sorted list
    Dim position As Integer 'to track from left to right through the sorted list
    Dim lowerbound As Integer = 0

    For count = lowerbound + 1 To mylist.Length - 1 'starts with second item in array
        current = mylist(count)
        position = count

        While position >= 0 And mylist(position - 1) > current 'Loop Line where VB
            tries to access position -1 in array
                mylist(position) = mylist(position - 1)
                position = position - 1

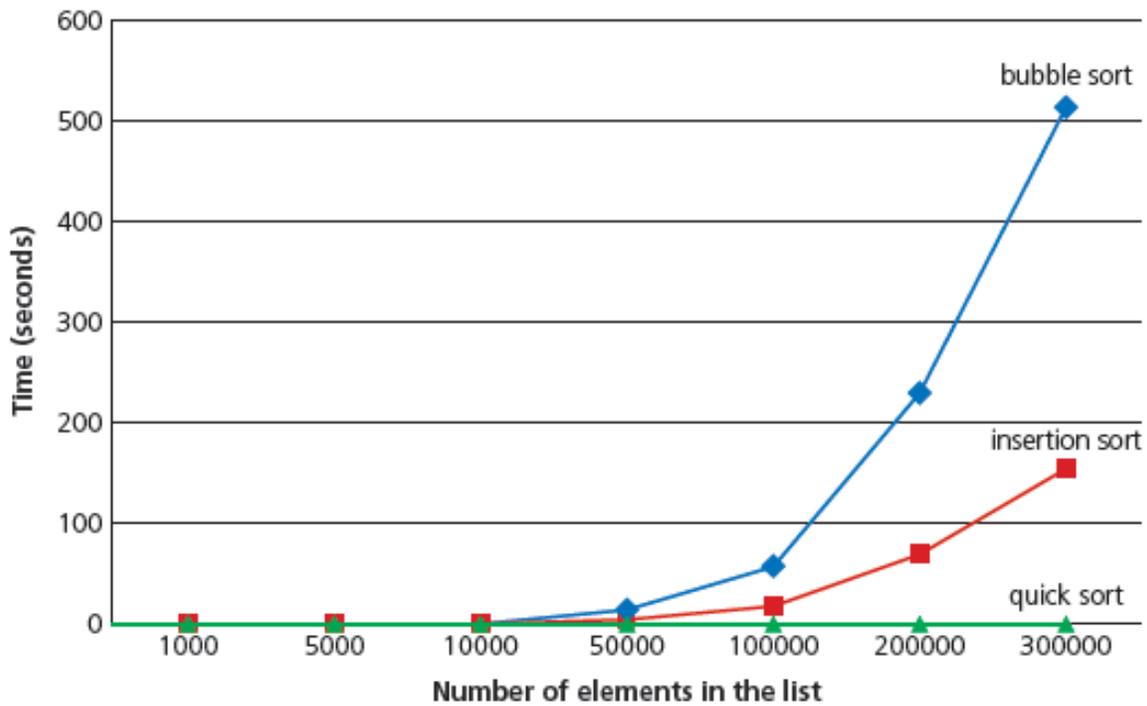
            If position = 0 Then 'This code stops VB to go to postion (-1) in array
                Exit While
            End If
        End While
        mylist(position) = current
    Next

    For index = 0 To mylist.Length - 1
        Console.WriteLine(mylist(index) & " ")
    Next
End Sub

```

Output:

The screenshot shows the Visual Studio IDE with the code from the previous block on the left and the Microsoft Visual Studio Debug Console on the right. The console output displays the sorted array: 14, 21, 27, 41, 43, 45, 46, 57, 70. Below the array, there is a message: 'C:\Users\Lenovo\source\repos\Inserion Sort Program\Debug\netcoreapp3.1\Inserion sort.exe d with code 0. To automatically close the console when debugging stops, please Tools->Options->Debugging->Automatically close console when debugging stops. Press any key to close this window . . .'



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.youtube.com/watch?v=l-kosUr1jtE>

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

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