Advanced Algorithms – Table of Contents

Solving a Large Problem by Splitting it into Several Smaller Sub-problems
Case Study: The Doomsday Algorithm

Introduction to the Doomsday Algorithm

Despite the ominous name of this algorithm, it is actually quite harmless. The doomsday algorithm, created by the eminent mathematician Professor John Horton Conway, is used to compute the day of the week given any valid Gregorian calendar date. The pictorial description below should help you understand what this means.

This remarkable algorithm is based on a certain day of the week that Professor Conway decided to call the ***doomsday***. Conway discovered that the so-called doomsday for any given year always falls on the same day of the week. For example, the doomsday for 2010 is Sunday. Using this fact and a little bit of modular arithmetic, Conway's algorithm cleverly calculates the result. In fact, with a little bit of practice, it is even possible to perform the algorithm mentally. For more information, use "doomsday algorithm" as a search phrase on www.google.ca.

Description of the Doomsday Algorithm

1. *Calculate the Doomsday for the Century Year*

(For example, if the given year were 1978, then the century year would be 1900.)

If the century year is divisible by 400, the century doomsday is Tuesday.
If the century year divided by 400 has a remainder of 100, the century doomsday is Sunday.
If the century year divided by 400 has a remainder of 200, the century doomsday is Friday.
Otherwise, the century doomsday is Wednesday.

2. *Use the Result from Step One to Calculate the Doomsday for the Given Year*

a. Calculate the number of years since the century year. Call this quantity "YSC" (e.g. 1978 – 1900 = 78)

b. Calculate the "OFFSET" from the century year doomsday:
- Determine the number of "12s" in YSC.
- Add the remainder of the previous step.
- Add the number of "4s" in the remainder from the previous step.
- Finally, determine the remainder upon division by 7.

The numbers 4, 7 and 12 are used in this step. What is their significance?

c. Calculate the doomsday for the given year by finding the remainder of the sum of YSC and OFFSET upon division by 7. (Call this quantity "DY") Why must this quantity be a whole number ranging from 0 to 6?

3. *Determine the Date of the Doomsday of the Given Month*

a. For all "even" months *except for February*, the N^{th} day of the N^{th} month is doomsday
b. For all "odd" months *except for January*,
- The $(N + 4)^{th}$ day of the N^{th} month is doomsday for the "long" months (31 days).
- The $(N – 4)^{th}$ day of the N^{th} month is doomsday for the "short" months (30 days).

c. The *last day* of February is doomsday for February (28^{th} for non-leap years, 29^{th} for leap years).
d. January 31 is doomsday for January, except in leap years, when January 32 (really Feb 1) is doomsday.

4. *Determine the Difference between the Given Day of the Month and the Doomsday for the Month*

For instance, if the doomsday is Thursday, December 12, and the given date is December 17, then December 17 must be a Monday. The best way to understand this step is to use a calendar.

Exercises

Use the Doomsday Algorithm to find the day of the week for each of the following dates.

1. July 1, 1867 (What is the significance of this date?)	**2.** February 15, 1965 (What is the significance of this date?)	**3.** December 25, 2012 (What is the significance of this date?)

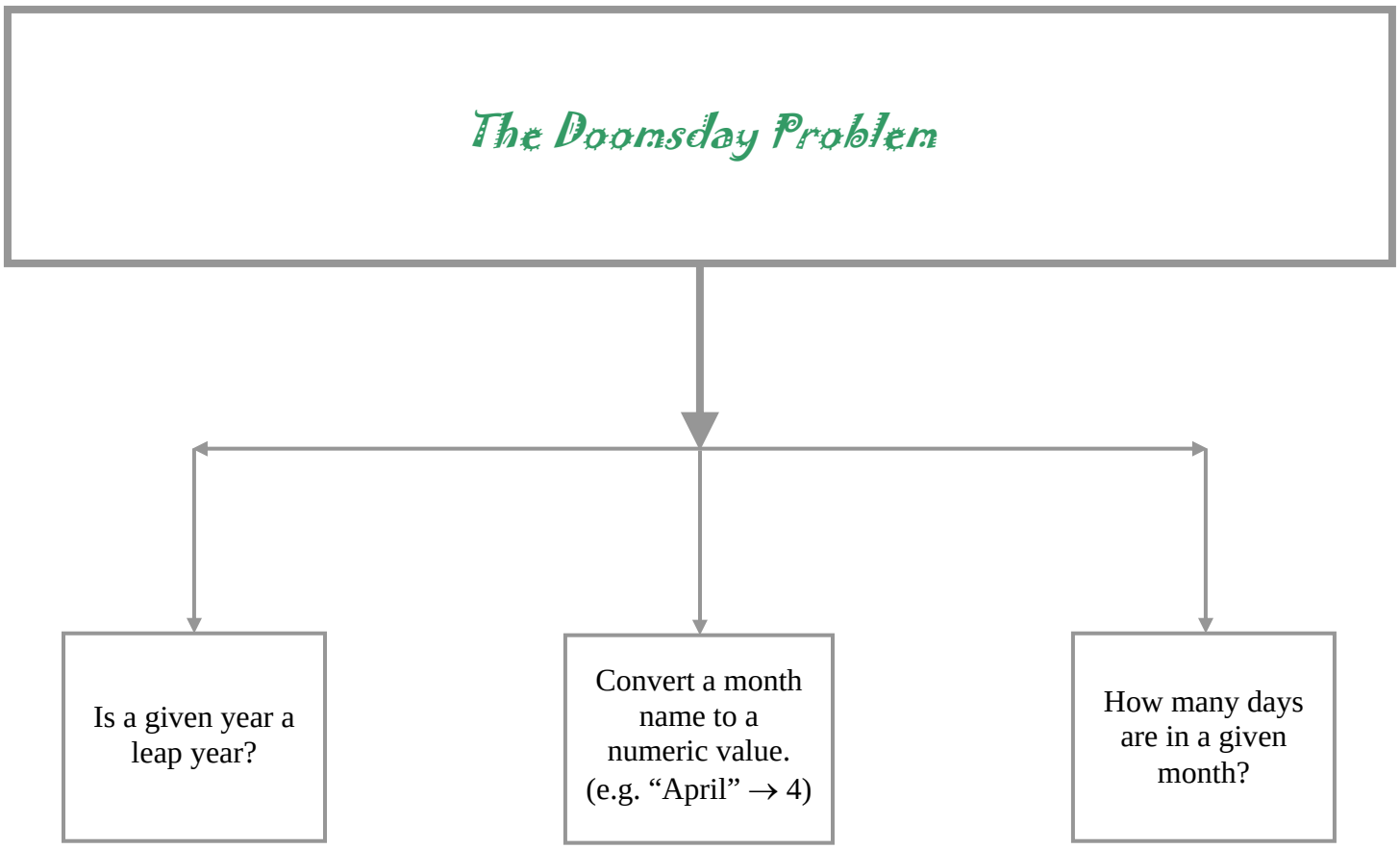
Now it's Time to Write Pseudo-Code!

Now, write pseudo-code for the Doomsday algorithm. To simplify this task, use the following abbreviations:

Y = year entered (1583 – 9999)	**CY** = century year	**YSC** = years since century year	**OFF** =for "offset" in step 2b
DY = doomsday for the given year (# from 0 to 6)*	**DM** = doomsday for the given month (# from 1 to 28, 29, 30, 31 or 32**, depending on the length of the month)		**DW** = day of the week for the entered date (# from 0 to 6)*

Note: * The integers from 0 to 6 represent the days of the week (0 = Sunday, 1 = Monday, 2 = Tuesday, …, 6 = Saturday).
** Recall that for leap years, **DM** for January is set to 32. This artificial date is needed to make the algorithm work properly.

How to Split the Doomsday Problem into Smaller Sub-Problems



The point of this diagram is to help you understand how a large, complicated problem can be broken up into a few simpler, smaller problems.

Finally, it's Time to Write Code!

You should not attempt to write code unless you have done an extensive analysis like the one shown above. If you feel that you now have a good understanding of the doomsday algorithm, you may proceed to study the code given below.

Code Found in the Code Module (modCommonCode.bas)

```
Option Explicit
Option Base 1
Public MonthOfYear As Variant, DayOfWeek As Variant
'A sub called "Main" is used to contain the code that should be executed as soon as a VB program is launched.
Public Sub Main()
    'Create an array that stores the months of the year.
    MonthOfYear = Array("January", "February", "March", "April", "May", "June", "July", "August", "September", _
                                                                    "October", "November", "December")
    'Create an array that stores the days of the week.
    DayOfWeek = Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
    frmDoomsday.Show
End Sub
```

"MonthOfYear" is a string array that stores all the months of the year.

MonthOfYear	1	2	3	4	5	6	7	8	9	10	11	12
	"January"	"February"	"March"	"April"	"May"	"June"	"July"	"August"	"September"	"October"	"November"	"December"

```
'This function performs the doomsday algorithm on any valid Gregorian Calendar date from 1583 to 9999.  The
'Gregorian Calendar was adopted in September 1582.  To keep this program as simple as possible, dates in 1582
'are not accepted.
Public Function Doomsday(ByVal Month As Integer, ByVal Day As Integer, _
                                                 ByVal Year As Integer, ByVal DayOfWeek As Variant) As Variant

    Dim CenturyYear As Integer, CenturyDoomsday As Byte, YearDoomsday As Byte, YearsSinceCentury As Byte
    Dim Offset As Byte, MonthDoomsday As Byte

    CenturyYear = Year - Year Mod 100

    If CenturyYear Mod 400 = 0 Then
        CenturyDoomsday = 2 'Tuesday
    ElseIf CenturyYear Mod 400 = 100 Then
        CenturyDoomsday = 0 'Sunday
    ElseIf CenturyYear Mod 400 = 200 Then
        CenturyDoomsday = 5 'Friday
    Else
        CenturyDoomsday = 3 'Wednesday
    End If

    YearsSinceCentury = Year - CenturyYear
    Offset = (YearsSinceCentury \ 12 + YearsSinceCentury Mod 12 + (YearsSinceCentury Mod 12) \ 4) Mod 7
    YearDoomsday = (CenturyDoomsday + Offset) Mod 7

    'Now determine doomsday for given month
    If Month Mod 2 = 0 And Month <> 2 Then 'Even months except Feb
        MonthDoomsday = Month
    ElseIf Month Mod 2 = 1 And Month <> 1 Then 'Odd months except Jan
        If NumDaysInMonth(Month) = 31 Then
            MonthDoomsday = Month + 4
        Else
            MonthDoomsday = Month - 4
        End If
    ElseIf IsLeapYear(Year) Then 'Leap years, Jan and Feb
        If Month = 2 Then
            MonthDoomsday = 29
        Else
            MonthDoomsday = 32
        End If
    ElseIf Month = 2 Then 'Non-leap years, Jan and Feb
        MonthDoomsday = 28
    Else
        MonthDoomsday = 31
    End If

    'Finally determine the difference between the given date
    'and doomsday for the month and return the result.
    If Day >= MonthDoomsday Then
        Doomsday = DayOfWeek((YearDoomsday + Day - MonthDoomsday) Mod 7 + 1)
    Else
        Doomsday = DayOfWeek((YearDoomsday + 7 - (MonthDoomsday - Day) Mod 7) Mod 7 + 1)
    End If

End Function

'This function determines whether a given year is a leap year
Public Function IsLeapYear(ByVal Year As Integer) As Boolean

    If Year >= 1583 Then 'Gregorian calendar was adopted in Sep 1582
        If Year Mod 4 = 0 Then 'year is divisible by 4
            If Year Mod 100 = 0 Then 'year is divisible by 100
                If Year Mod 400 = 0 Then 'year is divisible by 400
                    IsLeapYear = True
                Else
                    IsLeapYear = False
                End If
            Else
                IsLeapYear = True
            End If
        Else
            IsLeapYear = False
        End If
    Else
        IsLeapYear = False
    End If

End Function
```

"DayOfWeek" is a string array storing the days of the week.

DayOfWeek	
1	"Sunday"
2	"Monday"
3	"Tuesday"
4	"Wednesday"
5	"Thursday"
6	"Friday"
7	"Saturday"

```
'This function determines the number of days in a given month.  The year must be specified if the month is
'February.  Otherwise, this function will return an incorrect value.
Public Function NumDaysInMonth(ByVal Month As Integer, Optional ByVal Year As Integer) As Byte
    Select Case Month
        Case 1, 3, 5, 7, 8, 10, 12
            NumDaysInMonth = 31
        Case 4, 6, 9, 11
            NumDaysInMonth = 30
        Case 2
            If Not IsLeapYear(Year) Then
                NumDaysInMonth = 28
            Else
                NumDaysInMonth = 29
            End If
        Case Else
            NumDaysInMonth = 0 'Invalid month has been passed
    End Select
End Function

'This function converts the name of a month to a numeric value.  For example, "February" would be converted
' to 2 and "December" would be converted to 12.
Public Function ConvMonthToNumber(ByVal Month As String, ByRef MonthName As Variant) As Byte
    Dim I As Byte
    ConvMonthToNumber = 0 'In case invalid month name is passed
    Month = Trim(LCase(Month))
    For I = 1 To 12
        If Month = Trim(LCase(MonthName(I))) Then
            ConvMonthToNumber = I
            Exit Function
        End If
    Next I
End Function
```

Code Found in the Form Module (frmDoomsday.frm)

```
Option Explicit
Private Sub Form_Load()
    Dim I As Byte
    'Load months of year into combo box.
    For I = 1 To 12
        cboMonth.AddItem MonthOfYear(I)
    Next I
    'Display today's date.
    cboMonth.Text = MonthOfYear(Month(Date))
    txtDay.Text = Day(Date)
    txtYear.Text = Year(Date)
    'Display current time
    lblTime.Caption = Time
End Sub
'Use the doomsday algorithm to calculate the day of the week
'for the date entered by the user.
Private Sub cmdCalculateDay_Click()
    On Error GoTo ErrorHandler
    Dim YearEntered As Integer, MonthEntered As Integer
    Dim DayEntered As Integer, WeekDay As String
    Dim Verb As String
    DayEntered = Val(txtDay.Text)
    YearEntered = Val(txtYear.Text)
    MonthEntered = ConvMonthToNumber(Trim(cboMonth.Text), MonthOfYear)
    'Verify that a valid date has been entered.  If not, exit the sub.
    If YearEntered < 1583 Or YearEntered > 9999 Or MonthEntered < 1 Or MonthEntered > 12 Or DayEntered < 0 Or _
                                                  DayEntered > NumDaysInMonth(MonthEntered, YearEntered) Then
        MsgBox "There is a problem with one or more of the values that you have entered." _
               & Chr(10) & "The year must be between 1583 and 9999." & Chr(10) _
               & "In addition, please check the month and day that you have" _
               & Chr(10) & "entered to ensure that the values are valid.", _
               vbExclamation, "Please check the date carefully!"
        Exit Sub
    End If 'Code continues on the next page.
```

What is the purpose of “chr(10)?”

```
    'Decide if the date entered is past, present or future.
    If YearEntered < Year(Date) Then
        Verb = " was a "
    ElseIf YearEntered > Year(Date) Then
        Verb = " will be a "
    ElseIf MonthEntered < Month(Date) Then
        Verb = " was a "
    ElseIf MonthEntered > Month(Date) Then
        Verb = " will be a "
    ElseIf DayEntered < Day(Date) Then
        Verb = " was a "
    ElseIf DayEntered > Day(Date) Then
        Verb = " will be a "
    Else
        Verb = " is a "
    End If
    'Now call the "Doomsday" function to calculate the day of
    'the week on which the given date fell/falls/will fall.
    WeekDay = Doomsday(MonthEntered, DayEntered, YearEntered, DayOfWeek)

    lblDayOfWeek.Caption = Trim(cboMonth.Text) & Str(DayEntered) _
            & "," & Str(YearEntered) & Verb & WeekDay & "."
    Exit Sub
ErrorHandler:
    MsgBox Err.Description
End Sub
'Allow only numeric values to be entered in the text boxes storing the day and year entered by the user.
Private Sub txtDay_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And _
            KeyAscii <> vbKeyBack Then
        KeyAscii = 0
    End If
End Sub
Private Sub txtYear_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And _
            KeyAscii <> vbKeyBack Then
        KeyAscii = 0
    End If
End Sub

'Update clock if necessary
Private Sub tmrUpdateTime_Timer()
    If CStr(Time) <> lblTime.Caption Then
        lblTime.Caption = Time
    End If
End Sub
```

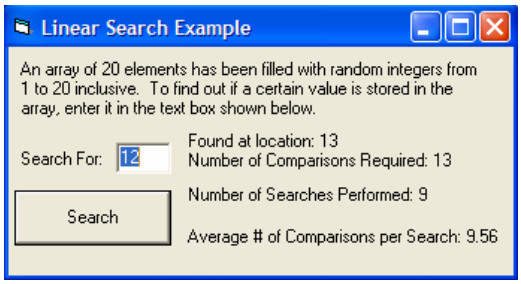
Searching Techniques

Linear Search

(Note that this linear search example VB program can be found in **I:\Out\Nolfi\Ics3m0\Searching and Sorting\Linear Search**.)

Consider the following Visual Basic program that uses a function procedure to perform a ***linear search*** (sequential search) of an array of *n* elements. For the sake of simplicity, the array to be searched is filled with random integers from 1 to 100.

```
Dim SomeArray(1 To 20) As Integer

Private Sub Form_Load()
  Dim I As Integer
  Randomize
  'Store random integers between 1 and 20 in the array.
  For I = 1 To 20
    SomeArray(I) = Int(Rnd*20+1)
  Next I
End Sub

Private Sub cmdClose_Click()
  End
End Sub

Private Sub cmdSearch_Click()
  Dim Location As Integer, NumComparisons As Integer
  Dim WhatToSearchFor As Byte, AvgNumComparisons As Single
  WhatToSearchFor = Val(txtSearchFor.Text)
  If WhatToSearchFor < 1 Or WhatToSearchFor > 20 Then
    MsgBox "Enter a value between 1 and 20", vbExclamation, "Oops!"
    Exit Sub
  End If
  Location = LinearSearch(SomeArray(),WhatToSearchFor, 20)
  If Location <> 0 Then
    NumComparisons = Location
    lblFoundAt.Caption = "Found at location:" & Str(Location) & _
                         & Chr(10)  "Number of Comparisons Required:"
                         & Str(NumComparisons)
  Else
     NumComparisons = 20
    lblFoundAt.Caption = "Not found"
  End If
  NumSearches = NumSearches + 1
  TotalComparisons = TotalComparisons + NumComparisons
  lblNumSearches.Caption = "Number of Searches Performed:" & _
                                                    Str(NumSearches)
  lblAvgComp.Caption = "Average # of Comparisons per Search:" & _
                       Str(Round(TotalComparisons / NumSearches, 2))
  'Highlight value in text box
  txtSearchFor.SelStart = 0
  txtSearchFor.SelLength = Len(txtSearchFor.Text)
End Sub

' This function performs a linear search of the array passed to the
' array parameter "A" for the value passed to the parameter "Item."
' If the item is found, its location within the array is returned.
' Otherwise, zero is returned.  It is assumed in this function that
' the array is declared with indices running from 1 to "N."

Function LinearSearch(A() As Integer, ByVal Item As Integer, _
                                      ByVal N As Integer) As Integer
  Dim I As Integer
  For I = 1 To N
    If A(I) = Item Then
      LinearSearch = I
      Exit Function
    End If
  Next I
  LinearSearch = 0 'Return 0 if required value was not found
End Function
```



Index	Data
1	3
2	12
3	14
4	1
5	1
6	6
7	11
8	19
9	17
10	20
11	5
12	7
13	12
14	2
15	14
16	19
17	8
18	18
19	7
20	9

The "linear search" method starts looking for the required element at the very "top" of the array (i.e. at the ***first*** element).

Then each element is examined in turn until either the required value is found or the end of the list is reached.

If the required value is found, its ***index*** is returned. Otherwise, ***zero*** is returned to signal that the required value was not found in the array.

Questions

1. State one advantage and one disadvantage of linear search.

2. To what kinds of applications is linear search well suited?

3. Rewrite the "LinearSearch" function above in such a way that it can search an array of strings. In addition, the rewritten function should also be able to deal with array indices running from "Low" to "High" (as opposed to indices running from 1 to N).

Binary Search

While linear search is easy to program and is reasonably fast when used to search small arrays, it is excruciatingly slow if used to search an array with a large number of elements. For instance, consider an array of one million strings. ***On average***, the linear search requires 500000 comparisons before a required value is found. ***In the worst case***, one million comparisons are needed. Obviously, this method wastes a great deal of CPU time. Fortunately, there are much faster algorithms that can be used to search very large data sets. ***Binary search***, for instance, can find any value in an array of 1000000 elements using ***10 or fewer comparisons***. In order for binary search to work, however, the array must be ***sorted***.

Example

Suppose that the following sorted array is being searched for the value "80." Unlike the linear search, the binary search begins at the middle of the array as shown below.

Index	Data
1	6
2	14
3	14
4	21
5	29
6	36
7	42
8	43
9	56
10	56
11	63
12	69
13	71
14	76
15	77
16	80
17	85
18	89
19	97
20	100

Step 1

The search begins at the ***middle of the array***. The value being sought is "80" and the value stored at the middle of the array is "56." Since 80 > 56, the first half of the list is ignored and the search continues at the middle of the second half of the array.

Index	Data
1	6
2	14
3	14
4	21
5	29
6	36
7	42
8	43
9	56
10	56
11	63
12	69
13	71
14	76
15	77
16	80
17	85
18	89
19	97
20	100

Step 2

The search continues at the ***middle of the second half of the array***, where "77" is stored. Since 80 > 77, the first half of the second half of the array is ignored and the search continues at the lowest quarter of the array.

Index	Data
1	6
2	14
3	14
4	21
5	29
6	36
7	42
8	43
9	56
10	56
11	63
12	69
13	71
14	76
15	77
16	80
17	85
18	89
19	97
20	100

Step 3

The search continues at the ***middle of the lowest quarter of the array***, where "85" is stored. Since 80 < 85, the second half of the lowest quarter of the array is ignored and the search continues.

Index	Data
1	6
2	14
3	14
4	21
5	29
6	36
7	42
8	43
9	56
10	56
11	63
12	69
13	71
14	76
15	77
16	80
17	85
18	89
19	97
20	100

Step 4

The search ends at element 16 of the array, where the required value is found. Notice that half the elements remaining are eliminated after each comparison, which means that no more than four comparisons are required to search 20 elements.

Important Programming Exercise

Load the linear search VB program from **I:\Out\Nolfi\Ics3m0\Searching and Sorting\Linear Search**. Modify it in such a way that the search is performed using the binary search algorithm. You simply need to write a new function that performs a binary search. In addition, you need a function that will sort the array. To solve this problem, look ahead to the next section and choose any of the given sorting methods.

Sorting Techniques

Exercise 1

Load "Sorting Methods.vbp" from **I:\Out\Nolfi\Ics3m0\Searching and Sorting\Sorting Methods Demo**. Carefully study the demo of each method of sorting. If necessary, repeat the demos several times until you feel that you understand all the sorting methods. Then complete the following table.

1. Use diagrams to explain the "Bubble Sort." Why is version 2 superior to version 1?	**2.** Use diagrams to explain the "Exchange Sort."	**3.** Use diagrams to explain the "Insertion Sort."

Exercise 2

Study the code in all modules in “Sorting Methods.vbp.” You will notice that the code module “modSortingSubs.bas” contains four subs that perform the four different sorting algorithms. Each of these subs contains a great deal of code that is only needed for the sorting demos. Rewrite each sub ***without*** the code needed to make the demos work.

ICS 3MO – Preparing for the Final Evaluation

The following question is similar to one that is on your final evaluation! Study it carefully!

In class, we studied the "Bubble Sort" algorithm for sorting an array of *n* elements. Presented below is an algorithm called "Exchange Sort," a simple sorting method similar to the bubble sort.

Description of the Exchange Sort Algorithm

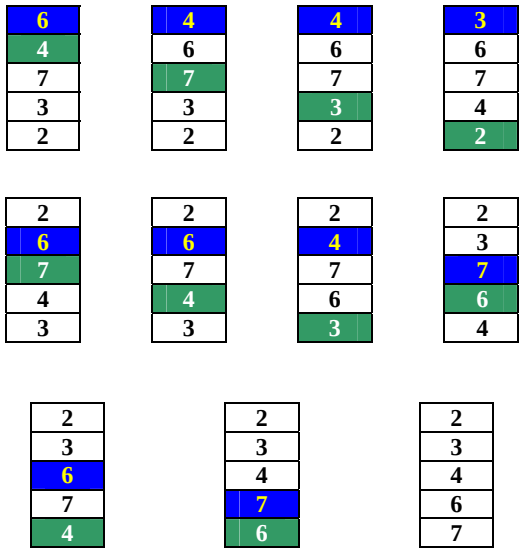
One of the simplest methods to sort an array is called an "**Exchange Sort**."

The first element of an array is compared to elements 2, … , *n*. Every time an element is found that is less than element 1, it is exchanged with element 1.

This process is then repeated with element 2. It is compared to elements 3, … , *n*. Every time an element is found that is less than element 2, it is exchanged with it.

Similarly, this process continues with elements 3, 4, 5 and so on.

Specific Example



Pseudo-Code for the Exchange Sort

Note: This pseudo-code assumes that the array has *n* elements with indices (subscripts) ranging from 1 to *n*.

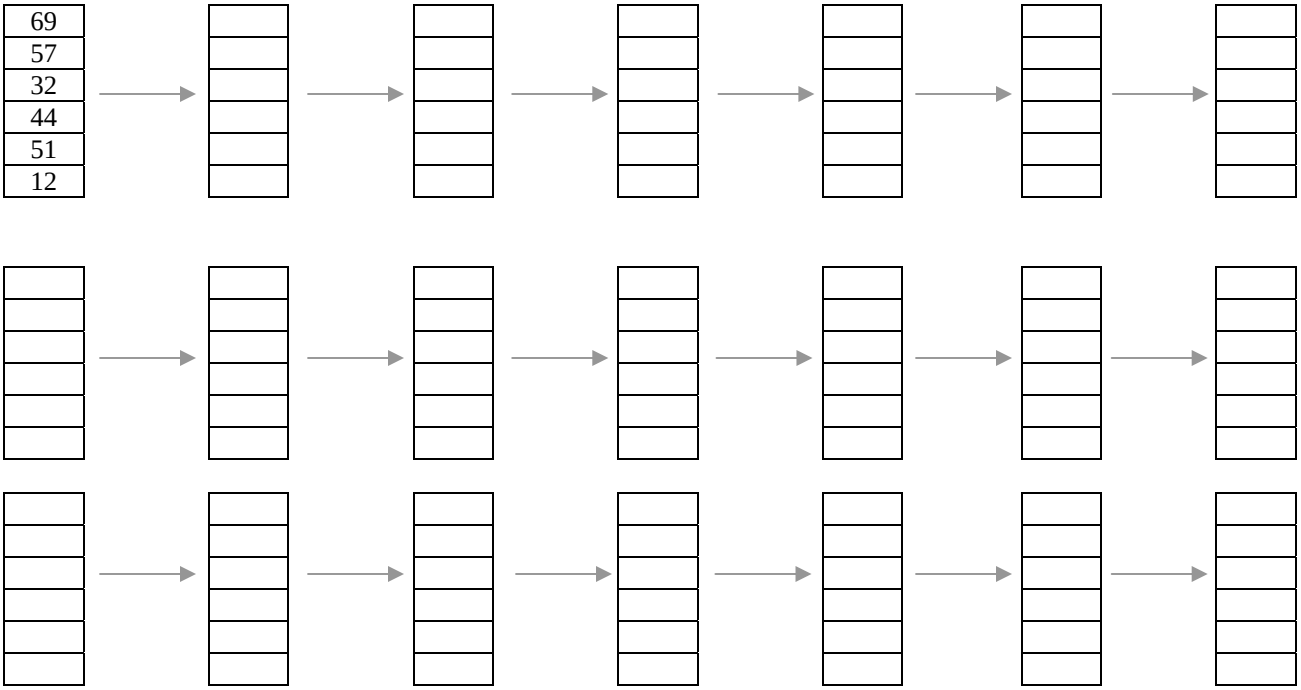
For I = 1 to *n*

Compare element I to element J, for all values of J ranging from I+1 to *n*.

If element J is < element I then *swap* (exchange) elements I and J.

Next I

a. Use the following tables to show the exact steps required to sort the given array using an exchange sort. Note that you may not necessarily need to use all the provided tables.



b. Now write VB code for the exchange sort. Use the pseudo-code above as a guide. (**Hint:** You will need a nested loop).