APPENDIX A Additional Illustrative Examples and Lab Exercises

Example 1 Swap Two Variables

To swap two variables we will use a temporary variable. First, we copy the value of any one variable in the temporary variable. Then, we copy the value of second variable in first. Finally, the value of the temporary variable is copied in the second variable. Look at Figure 1 given below to understand this concept.

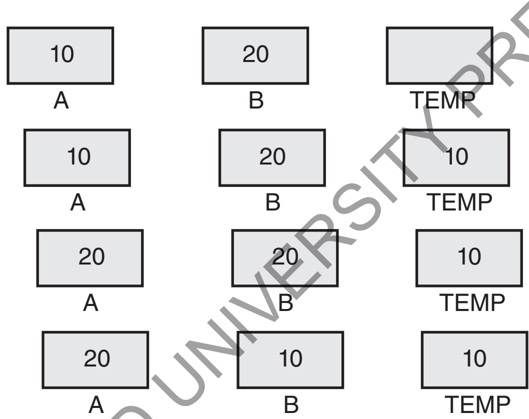


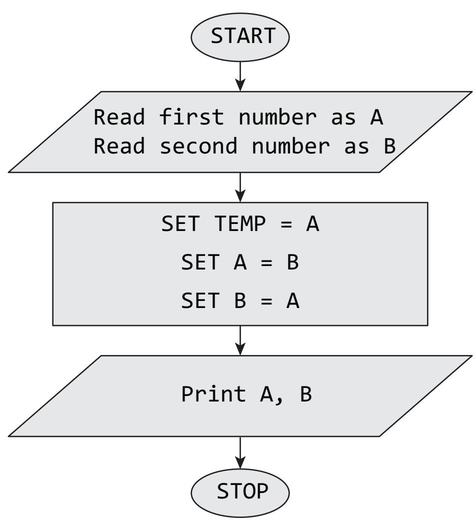
Figure 1 Steps involved in swapping two variables

Algorithm 1, Flowchart 1, and Pseudocode 1 demonstrate the step-wise solution for swapping two variables.

Algorithm 1

```
Step 1: Start
Step 2: Read the first number as A
Step 3: Read the second number as B
Step 4: SET TEMP = A
Step 5: SET A = B
Step 6: SET B = A
Step 7: PRINT A, B
Step 8: End
```

Flowchart 1



Pseudocode 1

```
Start
Read the first number as A
Read the second number as B
Set Temp = A
Set A = B
Set B = Temp
End
```

Note: Refer to Program 5.2 on Page 207 for the implementation of the concept using Python.

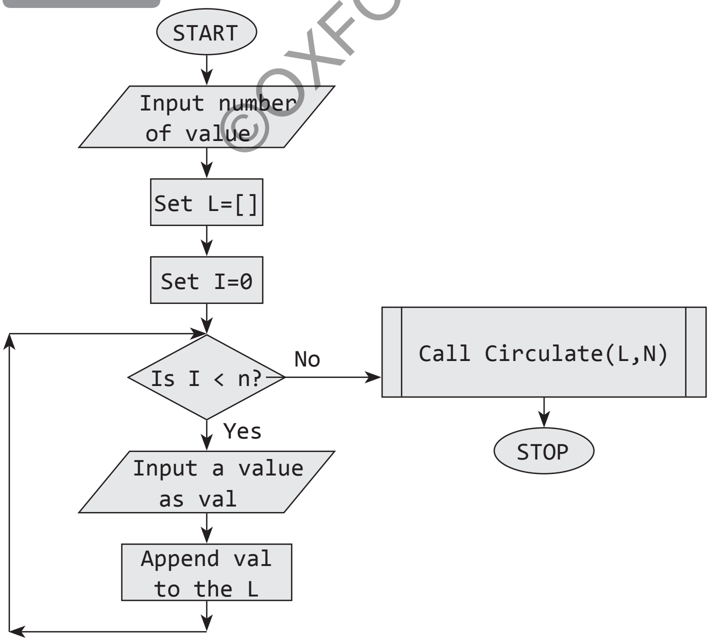
Example 2 **Circulate the Values of N Variables**

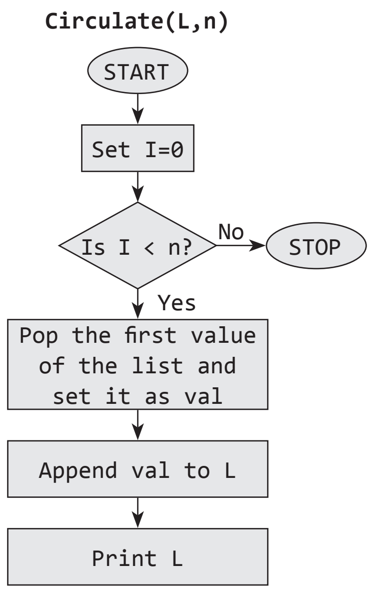
Algorithm 2

```
Step 1: Start
Step 2: Enter the number of elements in
        the list as n
Step 3: SET I = 0
Step 4: WHILE I < n
       Read an element
       Append the element to the list
       Print the list
       Calculate I = I + 1
Step 5: CALL CIRCULATE (list, n)
Step 6: End
```

```
CIRCULATE(list, n)
Step 1: Start
Step 2: Set I = 0
Step 3: WHILE I < n
        Pop the first element from the list
        Append it to the list (it now
        becomes the last element)
        Print the list
        Calculate I = I + 1
Step 4: End
```

Flowchart 2





Program 1 Write a program to circulate the values of N variables.

```
def circulate(L, n):
    print("Circulating the elements of list")
    for i in range(0,n):
        val = L.pop(0)
        L.append(val)
        print(L)

n = int(input("Enter number of values:"))
L = []
for i in range(0,n):
    val = int(input("Enter a value:"))
    L.append(val)
circulate(L,n)
```

OUTPUT

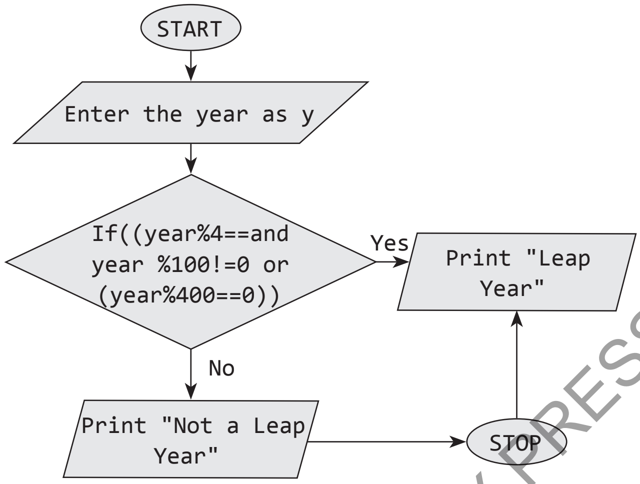
```
Enter number of values:5
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Circulating the elements of list
[2,3,4,5,1]
[3,4,5,1,2]
[4,5,1,2,3]
[5,1,2,3,4]
[1,2,3,4,5]
```


Example 3 Test for Leap Year

Algorithm 3

```
Step 1: Start
Step 2: Enter a year as year
Step 3: Check IF((year%4==0 and year %100!=0) or (year%400 == 0)),
          Then PRINT "Leap Year"
       ELSE
          PRINT "Not a Leap Year"
Step 4: End
```

Flowchart 3



Note: Refer to Program 4.8 for the implementation of the concept using Python.

Arrays as List

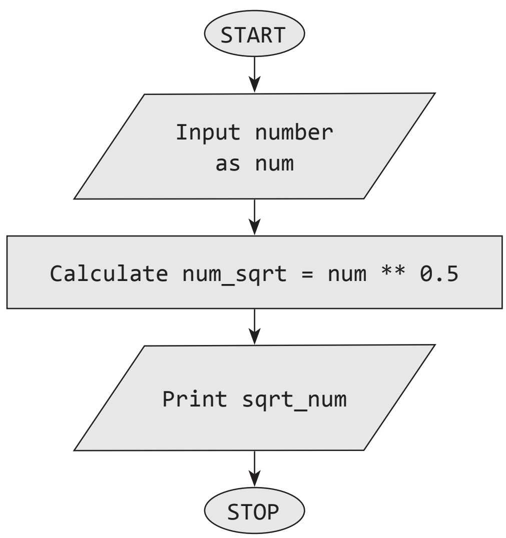
An array is one of the most fundamental data structures in any language. Though Python does not have a native array data structure, it supports list which is much more general and can also be used as a multi-dimensional array quite easily.

Example 4 **Square root of a number**

Algorithm 4

```
Step 1: Start
Step 2: Input the number as num
Step 3: Calculate square root as num **
        0.5
Step 4: Print square root as calculated in
        Step 3
Step 5: End
```

Flowchart 4



Program 2 Write a program to calculate the square root of a number.

```
x = float(input("Enter a number: "))
x_sqrt = x** 0.5
print("Square Root of %0.2f is %0.2f" %(x, x_sqrt))
```

OUTPUT

```
Enter a number: 5
Square Root of 5.00 is 2.24
```

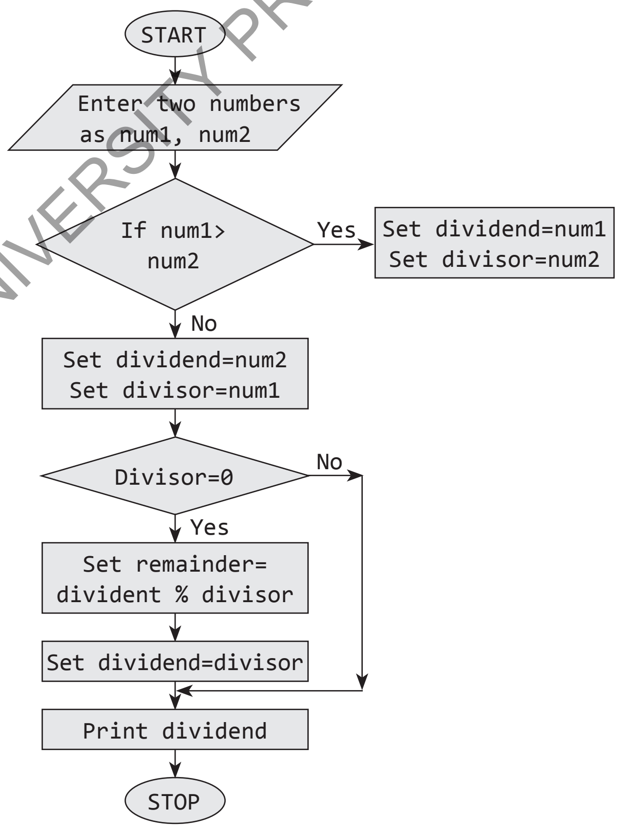
Example 5 GCD of Two Numbers

Algorithm 5

```
Step 1: Start
Step 2: Enter the two numbers as
        num1 and num2
Step 3: Set larger of the two
        numbers as dividend
Step 4: Set smaller of the two
        numbers as divisor
Step 5: Repeat Steps 6-8 while
        divisor != 0
Step 6: Set remainder = dividend
        % divisor
Step 7: Set dividend = divisor
Step 8: Set divisor = remainder
Step 9: Print dividend
Step 10: End
```

Note: Refer to Program 4.26 for the implementation of the concept using Python.

Flowchart 5



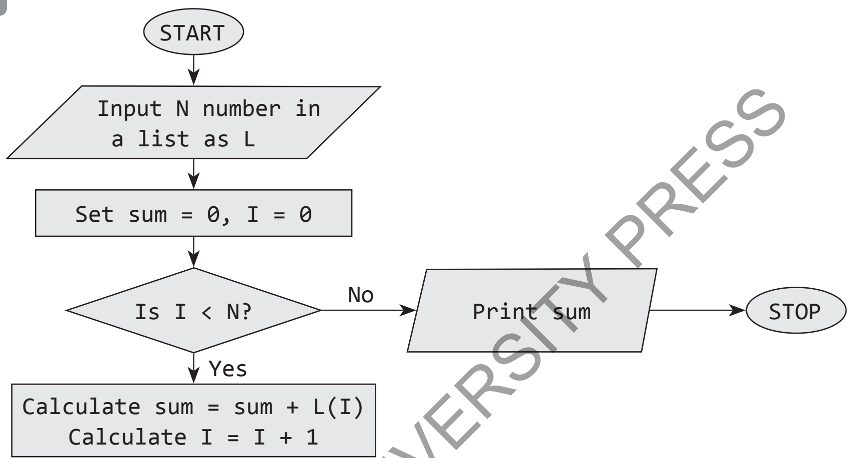
Example 6 Sum an Array of Numbers

Algorithm 6

```
Step 1: Start
Step 2: Read a list of N numbers from the user as L
Step 3: Set sum = 0, I = 0
```

```
Step 4: WHILE I < N
        Calculate sum = sum + L[I]
        Calculate I = I + 1
Step 5: Print sum
Step 6: End
```

Flowchart 6



Note: Refer to Example 8.17 for the implementation of the concept using Python.

Example 7 Counting Words in a File

We know that two consecutive words are separated from each other with a space ‘ ’. So to count the number of words written in the file, we will read the text from the file and count the number of spaces. Number of spaces + 1 gives the count of words as illustrated below using Algorithm 7, Flowchart 7, and Program 3.

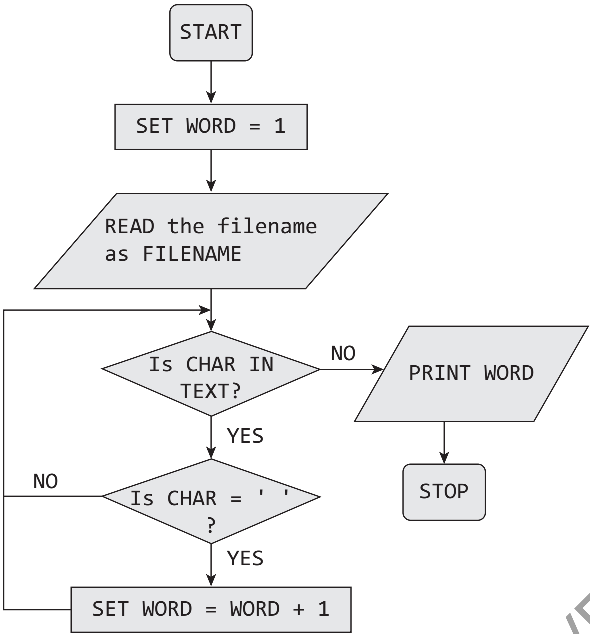
```
HELLO ALL
  WELCOME TO THE WORLD OF PROGRAMMING
```

Here, number of spaces including the new line is 7 and thus, the number of words is 8.

Algorithm 7

```
Step 1: Start
Step 2: SET WORD = 1
Step 3: READ the filename as FILENAME
Step 4: Open the file
Step 5: READ contents of the file in TEXT
Step 6: REPEAT Step 7 WHILE CHAR in TEXT
Step 7:     IF CHAR == ' '
            THEN WORD = WORD + 1
Step 8: PRINT WORD
Step 9: End
```

Flowchart 7



Program 3 Write a program to count number of words in a file.

```
word = 1
filename = input("Enter the filename : ")
with open(filename) as file:
    text = file.read()
    for char in text:
        if char == ' ':
            word = word + 1
print("WORDS = ", word)
```

OUTPUT

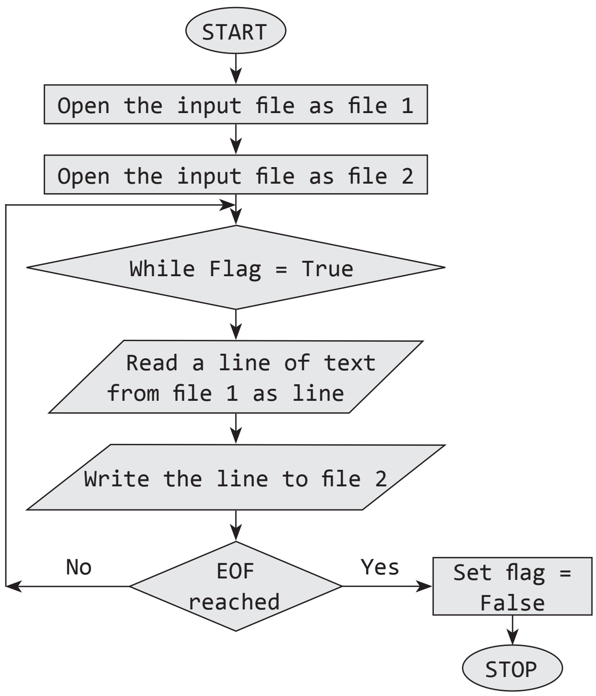
```
Enter the filename: filel.py
WORD = 8
```


Example 8 Copy a File

Algorithm 8

```
Step 1: Start
Step 2: Open the input file as file1
Step 3: Open the output file as file2
Step 4: WHILE End of file1 is not
        reached
          Read a line from file1 and
          write it in file2
Step 5: End
```

Flowchart 8



Program 4 Write a program to copy a file.

```
with open("File1.txt") as file1:
    with open("File2.txt", "w") as file2:
        for line in file1:
            file2.write(line)
```

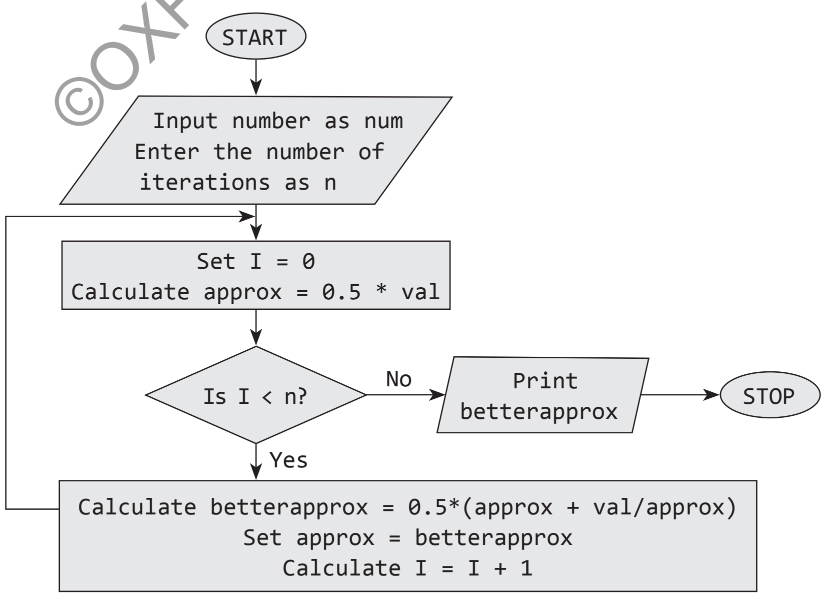
Lab Exercises

L.1 Square Root of a Number (Newton's Method)

Algorithm L.1

```
Step 1: Start
Step 2: Enter the number as num
Step 3: Enter the number of iterations as n
Step 4: Set I =0
Step 5: Calculate approx = 0.5 * val
Step 6: Repeat Steps 7-9 WHILE I < n
Step 7: Calculate betterapprox = 0.5 * (approx + val/approx)
Step 8: Set approx = betterapprox
Step 9: Calculate I = I + 1
Step 10: Print betterapprox
Step 11: End
```


Flowchart L.1



Program L.1 Write a program to find the square root of a number using Newton's method.

```
def Sqrt(val, n):
    approx = 0.5 * val
    for i in range(n):
        betterapprox = 0.5 * (approx + val/approx)
        approx = betterapprox
    return betterapprox
print(Sqrt(10, 2))
```

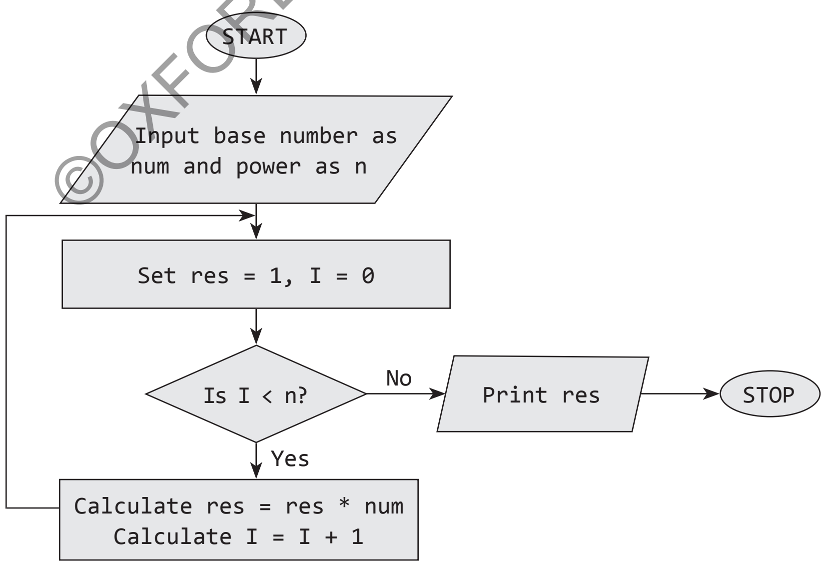
OUTPUT

```
3.178571428571429
```

L.2 Exponentiation (Power of a Number)

```
Step 1: Start
Step 2: Read the base number as num
Step 3: Read the power as n
Step 4: Set res = 1, I = 0
Step 5: Repeat Steps 6 and 7 WHILE I < n
Step 6: Calculate res =res * num
Step 7: Calculate I = I + 1
Step 8: Print res
Step 9: End
```

Flowchart L.2



Note: Refer to Program 4.35 for the implementation of the concept using Python.

L.3 Linear Search

Linear search, also called *sequential search*, is a very simple method used for searching a list for a particular value. It works by comparing every element of the list one by one in sequence until a match is found. Linear search is mostly used to search an unordered list of elements. This refers to a list in which data elements are not sorted. For example, if an list `A` is declared and initialized as

A = [10, 8, 2, 7, 3, 4, 9, 1, 6, 5]

and VAL = 7, then searching means to find out whether the value '7' is present in the list or not. If yes, then it returns the position of its occurrence. Here, the POS = 3 (index starting from 0). Figure 2 given below illustrates this concept.

Let us study the linear search mechanism of searching a list using Algorithm L.3, Flowchart L.3, and Program L.3.

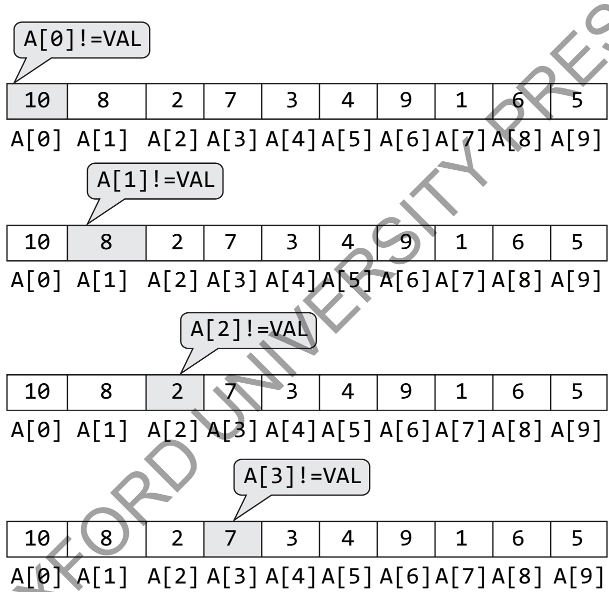
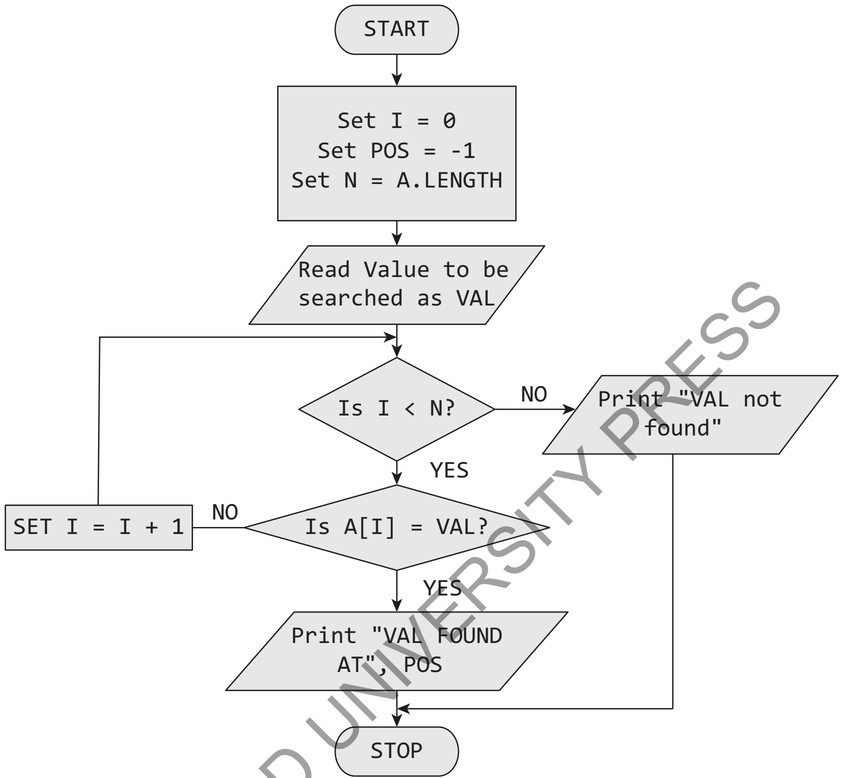


Figure 2 Steps used in Linear Search

Algorithm L.3

```
Step 1: Start
Step 2: SET POS = -1, I = 0, N = A.LENGTH
Step 3: READ number to be found as VAL
Step 4: REPEAT Step 5 WHILE I < N
Step 5: IF A[I] = VAL
                THEN  SET POS = I
                      PRINT "NUMBER FOUND AT POS"
                GOTO Step 7
        ELSE
                SET I = I + 1
Step 6: PRINT "NUMBER NOT FOUND"
Step 7: End
```

Flowchart L.3



Program L.3 Write a program to show linear search in a list.

```
def linear_search(A,ele):
    for i in range(len(A)):
        if(A[i] == ele):
            print('Element present at : %d'%(i+1))
            return
    print("Element not present.")

A = [5,1,2,9,0,6,3,8]
linear_search(A,10)
```

OUTPUT

```
Element present at : 6
```

L.4 Binary Search

We have seen that the linear search algorithm is very slow. However, if the list is sorted, we have a better and efficient alternative, known as binary search.

Binary search refers to searching an algorithm that works efficiently with a sorted list. The algorithm finds out the position of a particular element in the list. The mechanism of binary search can be better understood by an analogy of a telephone directory. When we are searching for a particular name in the directory, we will first open the directory from the middle and decide whether to look for the name in the first part of the directory or in the second part of the directory. We will open some page in the middle and the whole process will be repeated until we finally find the name.

Take another analogy. In order to find words in a dictionary, we open the dictionary somewhere in the middle, compare the first word on that page with the desired word whose meaning has to be found. If the word comes before the word that appears on the page, we will look in the first half of the dictionary; else, we will look in the second half. Again, we will open up some page in the first half of the dictionary's pages, compare the first word on that page with the desired word, and repeat the same procedure until we finally get the word. The same mechanism is applied in binary search.

Now, let us see how this mechanism will be applied to search for a value in a sorted list through Algorithm L.4, Flowchart L.4, and Program L.4.

Algorithm L.4

```
Step 1: Start
Step 2: SET BEG = 0, END = A.LENGTH, POS = - 1
Step 3: READ number to be found as VAL
Step 4: Repeat Steps 5 and 6 while BEG <= END
Step 5: SET MID = (BEG + END)/2
Step 6: IF A[MID] = VAL
           SET POS = MID
           PRINT POS
           Go to Step 6
           ELSE IF A[MID] > VAL
           SET END = MID - 1
           ELSE
           SET BEG = MID + 1
Step 7: IF POS = -1
           PRINT "VALUE IS NOT PRESENT IN THE ARRAY"
Step 8: End
```

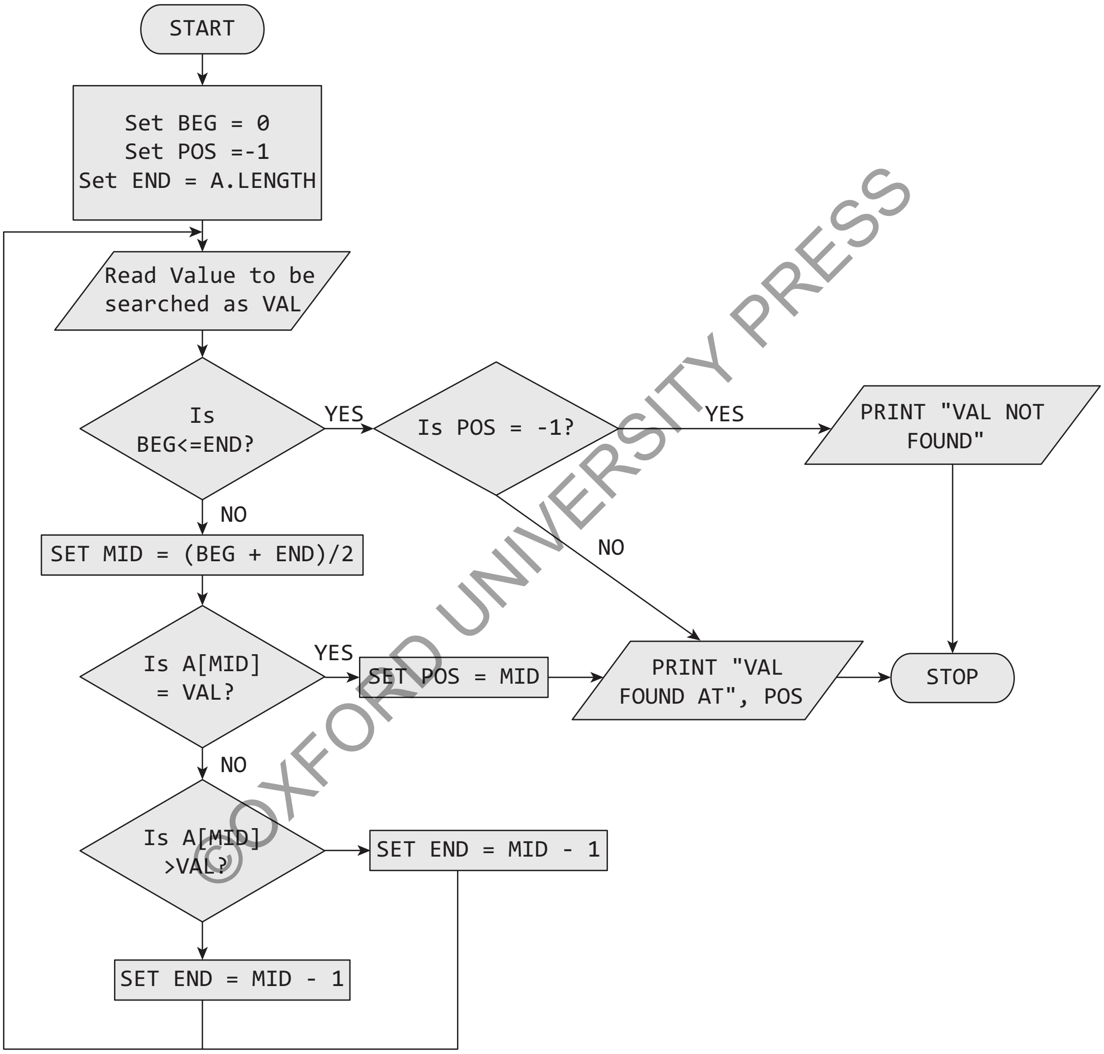

In this algorithm, we see that BEG and END are the beginning and ending positions of the segment that we are looking to search for the element. MID is calculated as (BEG + END)/2. Initially, BEG = lower_bound and END = upper_bound. The algorithm will terminate when A[MID] = VAL. When the algorithm ends, we will set POS = MID. POS is the position at which the value is present in the list.

However, if VAL is not equal to A[MID], then the values of BEG, END, and MID will change, depending on whether the VAL is smaller than or greater than A[MID].

- If VAL < A[MID], then VAL will be present in the left segment of the list. Therefore, the value of END will be changed as, END = MID − 1
- If VAL > A[MID], then VAL will be present in the right segment of the list. So, the value of BEG will be changed as, BEG = MID + 1

Finally, if the VAL is not present in the list, then eventually, END will be less than BEG. When this happens, the algorithm should terminate as it will indicate that the element is not present in the list and the search will be unsuccessful.

Flowchart L.4



Program L.4 Write a program to show binary search in a list.

```
def binary_search(A, VAL):
  first = 0
  last = len(A)-1
  found = False
```

```
  while first<=last and not found:
    mid = (first + last)//2
    if A[mid] == VAL:
      print("Value at position %d"%(mid+1))
      found = True
      break
    else:
      if VAL < A[mid]:
        last = mid-1
      else:
        first = mid+1
  if(found!= True):
    print("Value not present.")
A = [1,2,3,4,6,9,13,15,19]
binary_search(A,1)
```

OUTPUT

```
Element at position 1
```


L.5 Selection Sort

Selection sort is generally the preferred choice for sorting files with very large objects (records) and small keys. Although selection sort performs worse than insertion sort algorithm it is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

Technique Consider a list `ARR` with `N` elements. The selection sort makes `N-1` passes to sort the entire list and works as follows.

- First find the smallest value in the list and place it in the first position.
- Then find the second smallest value in the list and place it in the second position.
- Repeat this procedure until the entire list is sorted.

Consider Figure 3. In **pass 1**, find the position POS of the smallest value in the list and then swap `ARR[POS]` and `ARR[0]`. Thus, `ARR[0]` is sorted.

In **pass 2**, find the position `POS` of the smallest value in sub-list of `N-1` elements. Swap `ARR[POS]` with `ARR[1]`. Now, `A[0]` and `A[1]` is sorted.

39	9	81	45	90	27	72	18

Pass	POS	ARR[0]	ARR[1]	ARR[2]	ARR[3]	ARR[4]	ARR[5]	ARR[6]	ARR[7]
1	1	9	39	81	45	90	27	72	18
2	7	9	18	81	45	90	27	72	39
3	5	9	18	27	45	90	81	72	39
4	7	9	18	27	39	90	81	72	45
5	7	9	18	27	39	45	81	72	90
6	6	9	18	27	39	45	72	81	90
7	6	9	18	27	39	45	72	81	90

In **pass 3**, find the position POS of the smallest value in sub-list of `N–2` elements. Swap `ARR[POS]` with `ARR[2]`. Now `ARR[0]`, `ARR[1]`, and `ARR[2]` is sorted.

In **pass N–1**, find the position POS of the smaller of the elements `ARR[N–2]` and `ARR[N–1]`. Swap `ARR[POS]` and `ARR[N–2]` so that `ARR[0]`, `ARR[1]`, … , `ARR[N-1]` is sorted.

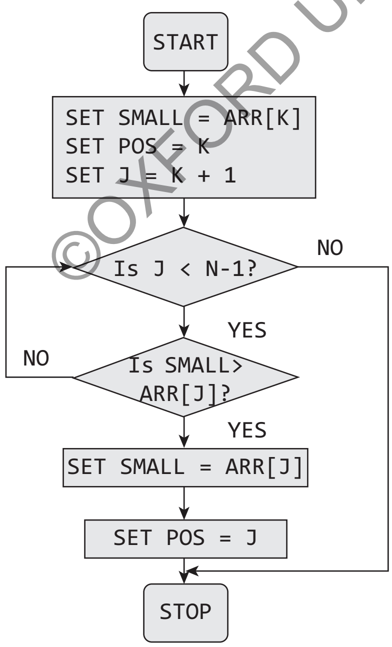
Let us study the mechanism of selection sort through Algorithm L.5, Flowchart L.5 and Program L.5.

Algorithm L.5

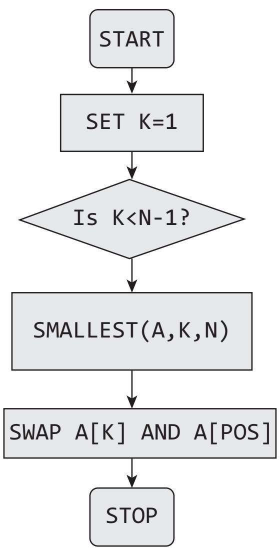
```
SMALLEST(ARR, K, N, POS)

Step 1: [INITIALIZE] SET SMALL = ARR[K]
Step 2: [INITIALIZE] SET POS = K
Step 3: Repeat for j = K+1 to N-1
            IF SMALL > ARR[J]
                  SET SMALL = ARR[J]
                  SET POS = J
            [END OF IF]
        [END OF LOOP]
Step 4: RETURN POS
```

```
SELECTION SORT(ARR, N)
Step 1: Repeat Steps 2 and 3 for K = 1
        to N-1
Step 2:   CALL SMALLEST (ARR, K, N, POS)
Step 3:   SWAP A[K] with ARR[POS]
      [END OF LOOP]
Step 4: End
```


Flowchart L.5



FLOWCHART FOR SMALLEST



FLOWCHART FOR SELECTION SORT

Program L.5 Write a program to implement selection sort.

```
def selectionsort(A):
    for i in range(len(A)):
        pos = i
        for j in range(i,len(A)):
            if(A[pos] > A[j]):
                pos = j
            if(pos!=i):
                temp = A[i]
                A[i] = A[pos]
                A[pos] = temp
    return A

A = [5,1,6,9,2,7,3]
A = selectionsort(A)
print("Sorted list : ",A)
```

OUTPUT

```
Sorted list :  [1, 2, 3, 5, 6, 7, 9]
```


The advantages of the selection sort algorithm are as follows:

- Simple and easy to implement
- Can be used for small data sets
- 60 per cent more efficient than bubble sort algorithm

L.6 Insertion Sort

Insertion sort is a very simple sorting algorithm, in which the sorted list (or list) is built one element at a time. We all are familiar with this technique of sorting as we usually use it for ordering a deck of cards while playing bridge. The main idea behind insertion sort is that it inserts each item into its proper place in the final list. To save memory, most implementations of the insertion sort algorithm work by moving the current data element past the already sorted values and repeatedly interchanging it with the preceding value until it is in its correct place.

Technique Insertion sort works as follows:

- The list of values to be sorted is divided into two sets. One that stores sorted values and the other contains unsorted values.
- The sorting algorithm will proceed until there are elements in the unsorted set.
- Suppose there are *n* elements in the list. Initially the element with index `0` (assuming `LB = 0`) is in the sorted set, rest of the elements are in the unsorted set.
- The first element of the unsorted partition has list index `1` (if `LB = 0`).
- During each iteration of the algorithm, the first element in the unsorted set is picked up and inserted into the correct position in the sorted set.

Consider a list of integers given below. Sort the values in the list using insertion sort.

39	9	45	63	18	81	108	54	72	36

39	9	45	63	18	81	108	54	72	36

A[0] is the only element in sorted list

9	39	45	63	18	81	108	54	72	36

(Pass 1)

9	39	45	63	18	81	108	54	72	36

(Pass 2)

9	39	45	63	18	81	108	54	72	36

(Pass 3)

9	18	39	45	63	81	108	54	72	36

(Pass 4)

9	18	39	45	63	81	108	54	72	36

(Pass 5)

9	18	39	45	63	81	108	54	72	36

(Pass 6)

9	18	39	45	54	63	81	108	72	36

(Pass 7)

9	18	39	45	54	63	72	81	108	36

(Pass 8)

9	18	36	39	45	54	63	72	81	108

(Pass 9)

Sorted Unsorted

Initially, `A[0]` is the only element in the sorted set. In Pass 1, `A[1]` will be placed either before or after `A[0]`, so that the list A is sorted. In pass 2, `A[2]` will be placed either before `A[0]`, in between `A[0]` and `A[1]`, or after `A[1]`, so that the list is sorted.

In pass 3, `A[3]` will be placed in its proper place so that the list A is sorted. In pass `N–1`, `A[N–1]` will be placed in its proper place so that the list `A` is sorted.

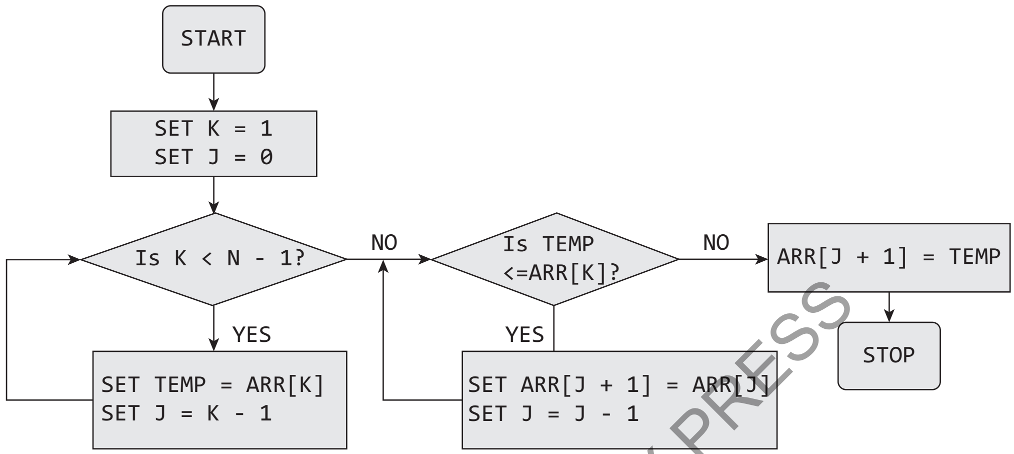
To insert the element A[K] in the sorted list `A[0], A[1], ...., A[K-1]`, we need to compare `A[K]` with `A[K-1]`, then with `A[K-2]`, then with `A[K-3]` until we meet an element `A[J]` such that `A[J] <= A[K]`. In order to insert `A[K]` in its correct position, we need to move each element `A[K-1], A[K-2], …, A[J+1]` by one position and then `A[K]` is inserted at the `(J+1)th` location.

Let us study the insertion sort mechanism through Algorithm L.6, Flowchart L.6, and Program L.6.

Algorithm L.6

```
INSERTION-SORT (ARR, N)
Step 1: Repeat Steps 2 to 5 for K = 1 to N – 1
Step 2: SET TEMP = ARR[K]
Step 3: SET J = K - 1
Step 4: Repeat while TEMP <= ARR[J]
              SET ARR[J + 1] = ARR[J]
              SET J = J - 1
Step 5: SET ARR[J + 1] = TEMP
Step 6: End
```

Flowchart L.6



Program L.6 Write a program to show insert sort.

```
def insertionSort(A):
  for index in range(1,len(A)):
    val = A[index]
    pos = index
    while pos>0 and A[pos-1]>val:
      A[pos]=A[pos-1]
      pos = pos-1
      A[pos]=val
  return A

A = [5,1,6,9,2,7,3]
A = insertionSort(A)
print("Sorted List : ",A)
```

OUTPUT

```
Sorted List :  [1, 2, 3, 5, 6, 7, 9]
```


L.7 Merge Sort

Merge sort is a sorting algorithm that uses the divide, conquer, and combine algorithmic paradigm. Divide means partitioning the n-element array to be sorted into two sub-arrays of n/2 elements. If A is an array containing zero or one element, then it is already sorted. However, if there are more elements in the array, divide A into two sub-arrays, A1 and A2, each containing about half of the elements of A. Conquer means sorting the two sub-arrays recursively using merge sort. Combine means merging the two sorted sub-arrays of size n/2 to produce the sorted array of n elements.

Merge sort algorithm focuses on two main concepts to improve its performance (running time):

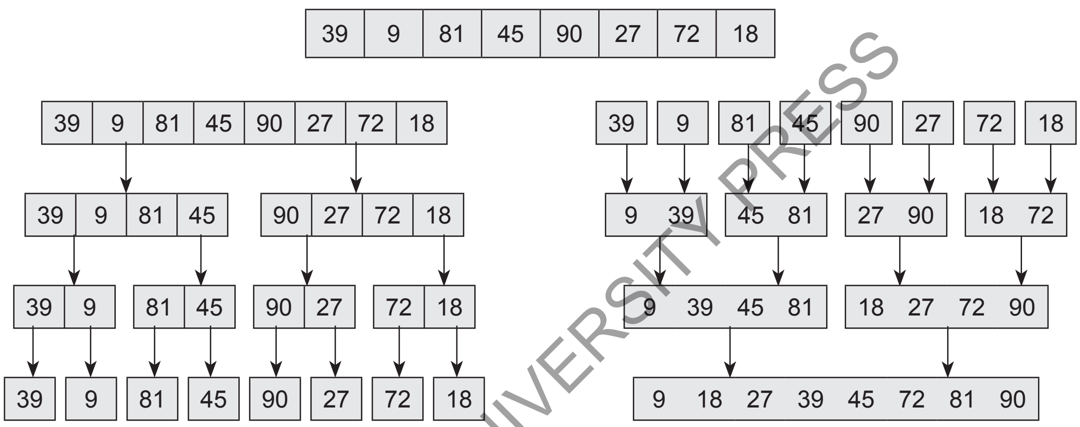
- A smaller list takes fewer steps and thus less time to sort than a large list.

- As number of steps is relatively less, thus less time is needed to create a sorted list from two sorted lists rather than creating it using two unsorted lists.

The basic steps of a merge sort algorithm are as follows:

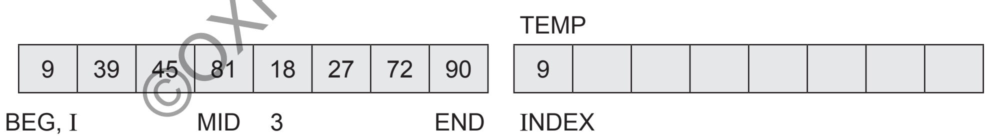
- If the array is of length 0 or 1, then it is already sorted.
- Otherwise, divide the unsorted array into two sub-arrays of about half the size.
- Use merge sort algorithm recursively to sort each sub-array.
- Merge the two sub-arrays to form a single sorted list.

Consider the array given below. Sort it using merge sort.

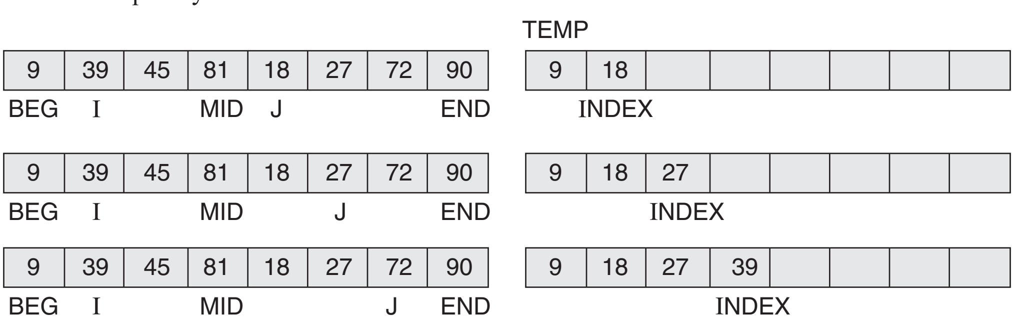


(Divide and Conquer the array) (Combine the elements to form a sorted array)

To understand the merge sort program, consider the figure below which shows how we merge two lists to form one list. For ease of understanding, we have taken two sub-lists each containing four elements. The same concept can be utilized to merge four sub-lists containing two elements, or eight sub-lists having one element each.



Compare ARR[I] and ARR[J], the smaller of the two is placed in TEMP at the location specified by INDEX and subsequently the value I or J is incremented.



9	39	45	81	18	27	72	90
BEG		I	MID			J	END

9	18	27	39	45			
				INDEX			

9	39	45	81	18	27	72	90
BEG			I,MID			J	END

9	18	27	39	45	72		
					INDEX		

9	39	45	81	18	27	72	90
BEG			I,MID				J END

9	18	27	39	45	72	81	
						INDEX	

When I is greater than MID, copy the remaining elements of the right sub-array in TEMP.

9	39	45	81	18	27	72	90
BEG			MID	I			J END

9	18	27	39	45	72	81	90
						INDEX	

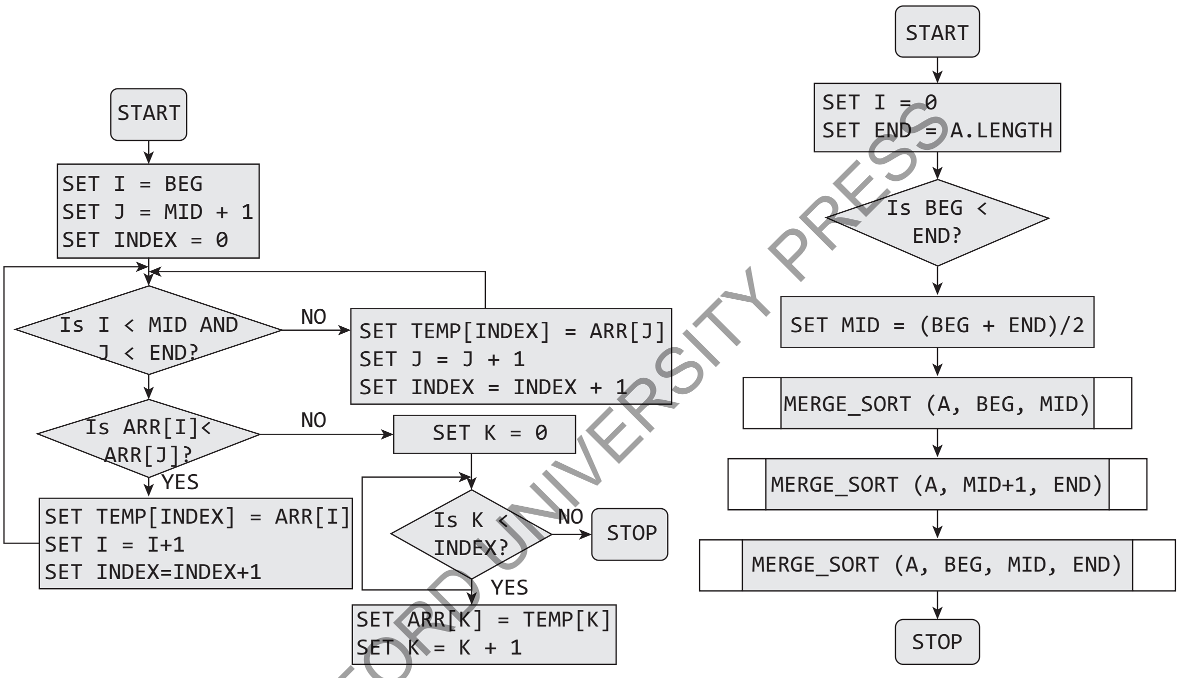
Let us study the mechanism of merge sort through Algorithm L.7, Flowchart L.7, and Program L.7.

Algorithm L.7

```
MERGE(ARR, BEG, MID, END)
Step 1: SET I = BEG, J = MID + 1, INDEX = 0
Step 2: Repeat while (I <= MID) AND (J<=END)
        IF ARR[I] < ARR[J]
        SET TEMP[INDEX] = ARR[I]
        SET I = I + 1
        ELSE
        SET TEMP[INDEX] = ARR[J]
        SET J = J + 1
        SET INDEX = INDEX + 1
Step 3: [Copy the remaining elements of right sub-array, if any]
        IF I > MID
        Repeat while J <= END
        SET TEMP[INDEX] = ARR[J]
        SET INDEX = INDEX + 1, SET J = J + 1
        [Copy the remaining elements of left sub-array, if any]
        ELSE
        Repeat while I <= MID
        SET TEMP[INDEX] = ARR[I]
        SET INDEX = INDEX + 1, SET I = I + 1
Step 4: [Copy the contents of TEMP back to ARR] SET K=0
Step 5: Repeat while K < INDEX
        SET ARR[K] = TEMP[K]
        SET K = K + 1
Step 6: End

MERGE_SORT(ARR, BEG, END)
Step 1: IF BEG < END
        SET MID = (BEG + END)/2
        CALL MERGE_SORT (ARR, BEG, MID)
```

```
        CALL MERGE_SORT (ARR, MID + 1, END)
        MERGE (ARR, BEG, MID, END)
Step 2: End
```

Flowchart L.7



Program L.7 Write a program to implement merge sort.

```
def mergesort(A):
    mid = int(len(A) / 2)
    left = A[:mid]
    right = A[mid:]
    if len(A) > 1:
        mergesort(left)
        mergesort(right)
        i, j = 0, 0
        a = left; b = right
        for k in range(len(a) + len(b) + 1):
            if a[i] <= b[j]:
                A[k] = a[i]
                i += 1
                if (i == len(a) and j != len(b)):
```

```
                    while(j != len(b)):
                        k +=1
                        A[k] = b[j]
                        j += 1
                    break
            elif a[i] > b[j]:
                A[k] = b[j]
                j += 1
                if j == len(b) and i != len(a):
                    while (i != len(a)):
                        k+= 1
                        A[k] = a[i]
                        i += 1
                    break
    return A

A = [5,1,6,9,2,7,3]
A = mergesort(A)
print("Sorted Array : ",A)
```

OUTPUT

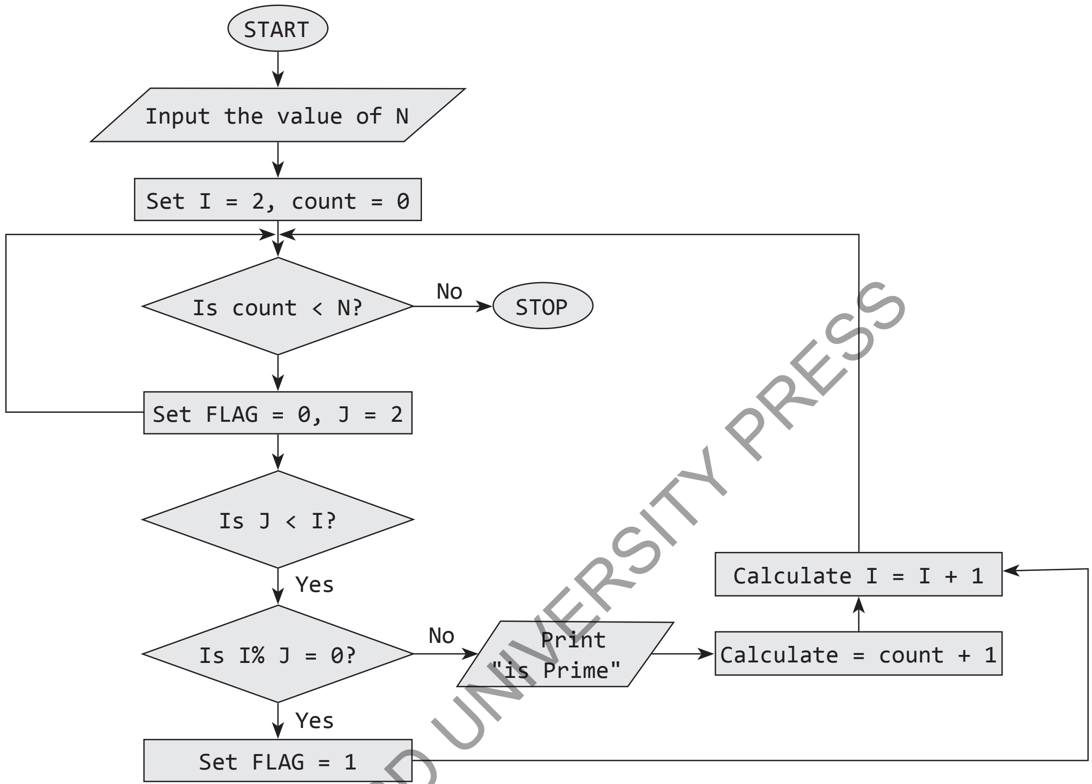
```
Sorted Array : [1, 2, 3, 5, 6, 7, 9]
```


L.8 First N Prime Numbers

Algorithm L.8

```
Step 1: Start
Step 2: Enter the value of n
Step 3: Set I = 2 and count = 0
Step 4: Repeat Steps 5 – 10 WHILE count < n
Step 5: Set flag = 0, j = 2
Step 6: Repeat Steps 7 – 8 WHILE j < I
Step 7: IF I % j = 0
Step 8: Then Set Flag = 1
        Break
Step 9: Check IF Flag =0
        Print I, "is Prime"
        Calculate count = count + 1
Step 10: Calculate I = I + 1
Step 11: End
```

Flowchart L.8



Program L.8 Write a program to print first N numbers.

```
n = int(input("Enter the number : "))
count = 0
i = 2
while(count<n):
    flag = 0
    for j in range(2,i):
        if(i%j==0):
            flag = 1
            break
    if(flag==0):
        print(i, "is prime")
        count +=1
    i=i+1
```

OUTPUT

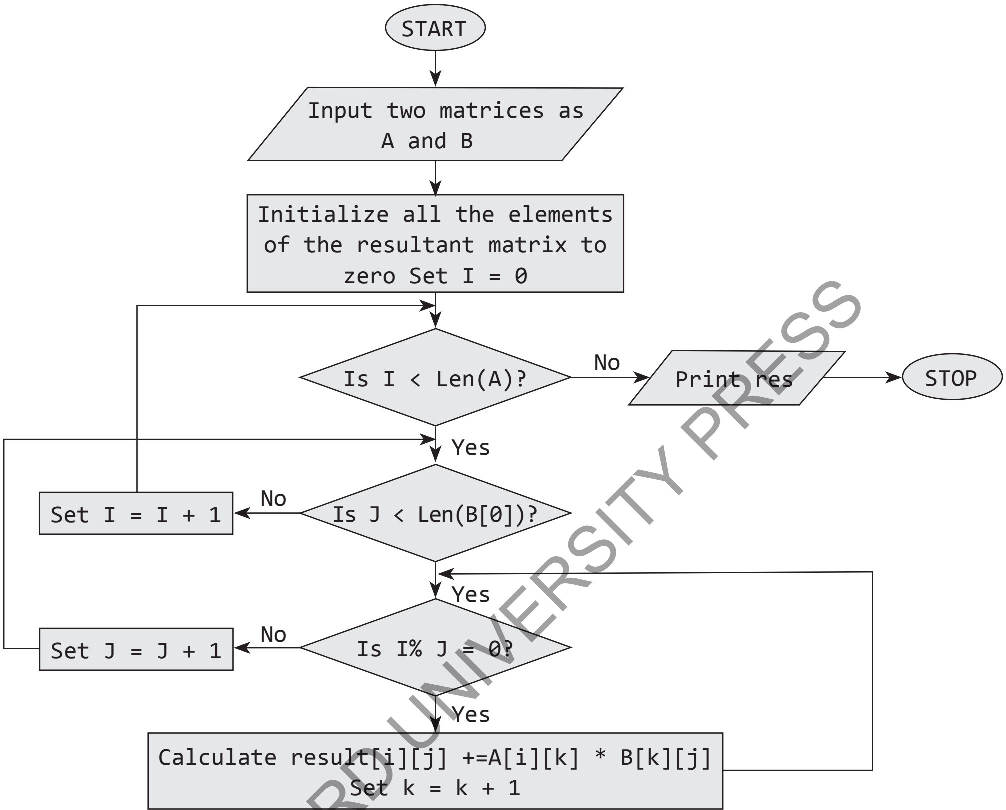
```
Enter the number : 10
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
```

L.9 Multiply two matrices

Algorithm L.9

```
Step 1: Start
Step 2: Read the two input matrices as A and B
Step 3: Initialize all the elements of resultant matrix with zeros
Step 4: Set I = 0
Step 5: Repeat steps 6 and 9 WHILE I < len(A) ……….. or number of rows in A
Step 6: Repeat step 7 and 8 WHILE j < len(B[0]) ……….. or number of columns in B
Step 7: WHILE k < len(B) ……….. or number of rows in B
        Calculate result[i][j] += A[i][k] * B[k][j]
        Set k = k +1
Step 8: Set j = j + 1
Step 9: Set I = I + 1
Step 10: Print result
Step 11: End
```

Flowchart L.9



Program L.9 Write a program to multiply two matrices.

```
A = [[1,2,3],
    [4 ,5,6],
    [7 ,8,9]]

B = [[5,6,7],
    [8,9,0],
    [4,6,3]]

result = [[0,0,0],
         [0,0,0],
         [0,0,0]]

for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            result[i][j] += A[i][k] * B[k][j]
```

```
for r in result:
   print(r)
```

OUTPUT

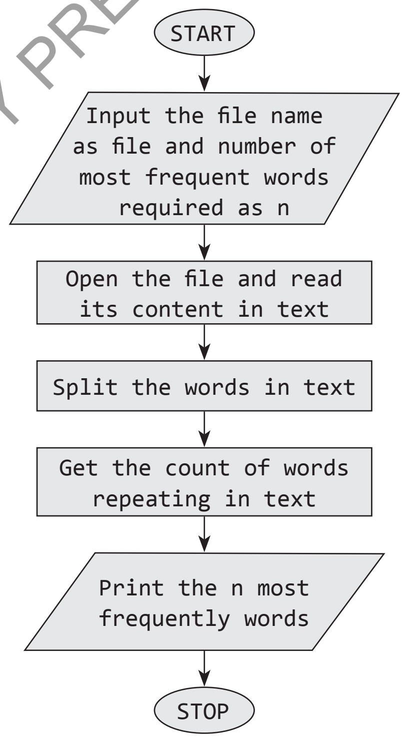
```
[33, 42, 16]
[84, 105, 46]
[135, 168, 76]
```

L.10 Find the most frequent words in a text read from a file.

Algorithm L.10

```
Step 1: Start
Step 2: Input the filename as file
Step 3: Input number of most frequent words
        required as n
Step 4: Open the file and read its content in
        text
Step 5: Split the words in text
Step 6: Get the count of words repeating in
        text
Step 7: Print the n most frequently words
Step 8: End
```

Flowchart L.10



Program L.10 Write a program to find the most frequent words in a text read from a file.

```
from collections import Counter
filename = input("Enter the filename : ")
n = int(input("How many frequent words do you want? "))
with open(filename) as file:
    text = file.read()
    split_it = text.split()
    Counter = Counter(split_it)
    most_occur = Counter.most_common(n)
print(most_occur)
```

OUTPUT

```
Enter the filename: File1.txt
How many frequent words do you want? 3
[('to', 5), ('of', 4), ('all', 3)]
```

L.11 Simulate elliptical orbit using Pygame

Algorithm L.11

```
Step 1: Start
Step 2: Open a window and display an appropriate caption
Step 3: Start the clock
Step 4: Repeat 5-8 while event is not exit
Step 5: Color the screen black
Step 6: Draw a circle at the middle of the screen
Step 7: Draw an elliptical orbit
Step 8: Draw a circle at a different position with a different angle to give an
        effect of simulation
Step 10: End
```


Program L.11 Write a program to simulate elliptical orbit using Pygame.

```
import pygame
import math
import sys
pygame.init()
# Open a window of size 640,480, and stores it in a variable called screen.
screen = pygame.display.set_mode((640, 480))
pygame.display.set_caption("Elliptical orbit")

clock = pygame.time.Clock()

while(True):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    xRadius = 250
    yRadius = 100

    for degree in range(0,360,10):
        x1 = int(math.cos(degree * 2 * math.pi / 360) * xRadius) + 300
        y1 = int(math.sin(degree * 2 * math.pi / 360) * yRadius) + 150
```

```
        screen.fill((black))     # fills the entire screen with the given color
    # Syntax: pygame.draw.circle(screen, color, (x,y), radius, thickness)

        pygame.draw.circle(screen, (255, 0, 0), [300, 150], 35)
       # Syntax pygame.draw.ellipse(screen, COLOR, [x, y, width,
height], thickness)
        pygame.draw.ellipse(screen, (255, 255, 255), [50, 50, 500, 200], 1)
        pygame.draw.circle(screen, (0, 0, 255), [x1, y1], 15)

        pygame.display.flip()    # Update the display screen
        clock.tick(5)      # To track number of frames to be rendered
```

L.12 Simulate bouncing ball using Pygame

Algorithm L.12

```
Step 1: Start
Step 2: Display the window
Step 3: Load the image of a ball
Step 4: Repeat steps 5-8 WHILE event is not exit
Step 5: Move the ball with the specified speed
Step 6: When the ball hits the horizontal boundary of the screen, it reverses the
        speed in x direction to be visible on the screen
Step 7: When the ball hits the vertical boundary of the screen, it reverses the
        speed in y direction to be visible on the screen
Step 8: Draw the circle and display the window
Step 9: End
```


Program L.12 Write a program to simulate a bouncing ball using Pygame

```
import sys, pygame
pygame.init()
size = width, height = 320, 240
speed = [2, 2]
black = 0, 0, 0
screen = pygame.display.set_mode(size)
ball = pygame.image.load("ball.bmp")
ballrect = ball.get_rect()
while 1:
for event in pygame.event.get():
if event.type == pygame.QUIT: sys.exit()
ballrect = ballrect.move(speed)
if ballrect.left < 0 or ballrect.right > width:
```

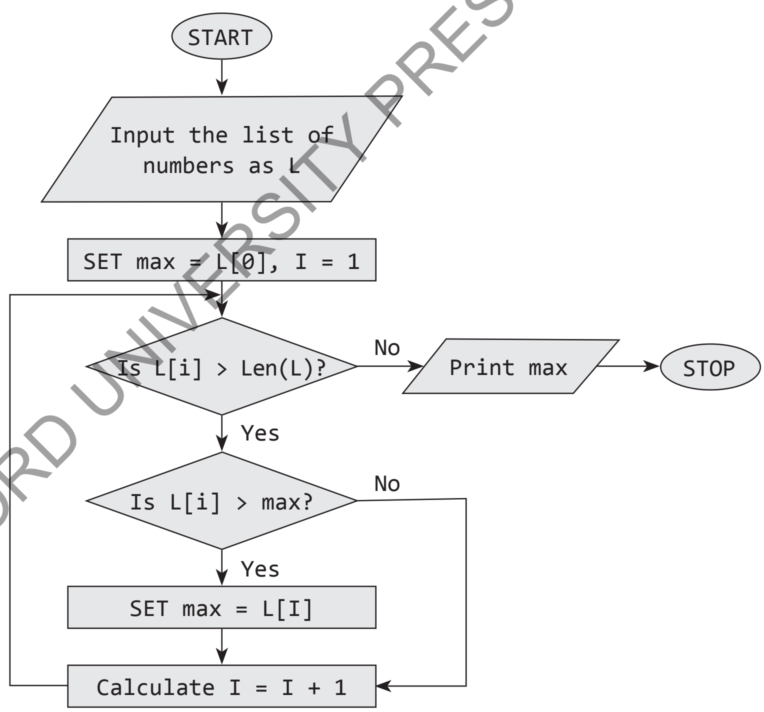
```
speed[0] = -speed[0]
if ballrect.top < 0 or ballrect.bottom > height:
speed[1] = -speed[1]
screen.fill(black)
pygame.draw.circle(screen, color, ballrect.center, radius)
pygame.display.flip()
```

L.13 Find the maximum of a list of numbers

Algorithm L.13

```
Step 1: Start
Step 2: Input the list
        of numbers as L
Step 3: Set max as L[0]
        and I as 0
Step 4: Repeat steps 5
        and 6 while I <
        len(L)
Step 5: IF L[I] > max,
        THEN
          SET max as L[I]
Step 6: Increment I
Step 7: Print max
Step 8: End
```

Flowchart L.13



Program L.13 Write a program to find the maximum of a list of numbers.

```
L = [100,34,56,90,87,99,12]
max = L[0]
for i in range(1, len(L)):
    if L[i]>max:
        max = L[i]
print("Maximum Value = ", max)
```

OUTPUT

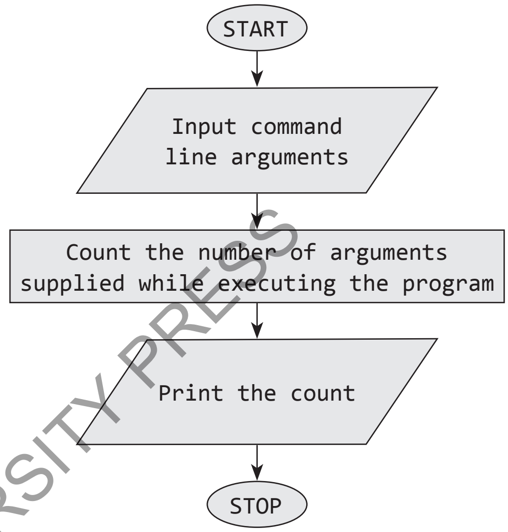
```
Maximum Value = 99
```

L.14 Program that takes command line arguments (word count).

Algorithm L.14

```
Step 1: Start
Step 2: Execute the program and supply the
        command line arguments
Step 3: Count the number of arguments
Step 4: Print the count
Step 5: End
```

Flowchart L.14



Program L.14 Write a program to do word count using command line arguments.

```
import sys
print(len(sys.argv))
print(sys.argv)
```

OUTPUT

```
C:\Python34>py try.py abc def
3
['try.py', 'abc', 'def']
```