ALGORITHM & Flowchart:

1.

ALGORITHM TO CHECK EVEN AND ODD NUMBERS

Step 1: Start

Step 2: [Take Input] Read: N

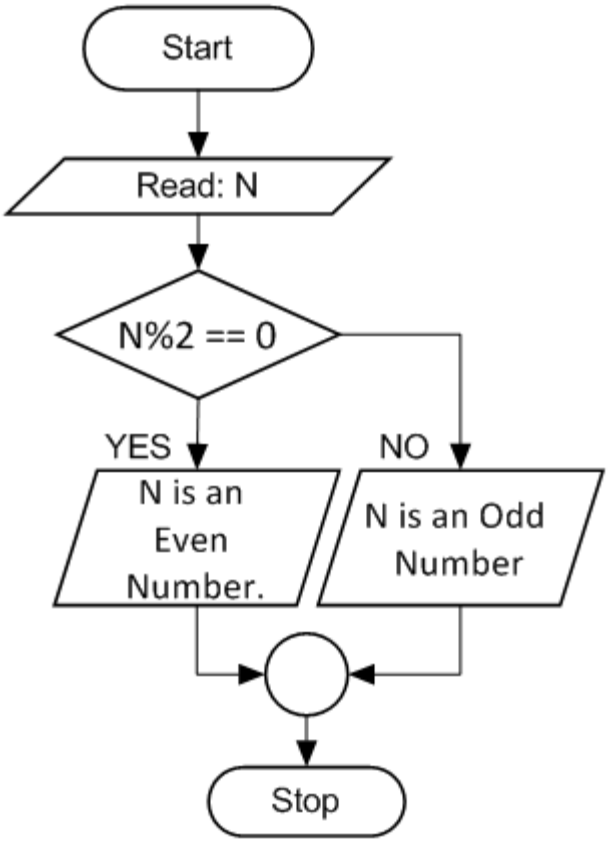
Step 3: Check: If N%2 == 0 Then

Print : N is an Even Number.

Else

Print : N is an Odd Number.

Step 4: Exit



ALGORITHM TO FIND LARGEST ELEMENT

An Array DATA of numerical values is in memory. We want to find the largest element and its location in Array DATA.

Method 1: ALGORITHM:

Suppose an array DATA having N numeric value is given. This algorithm finds the location as LOC and the largest number of the array DATA. To find the largest number we need compare the numbers one by one and for that we need an increment say k. Variable MAX is used to store the largest number.

Step 1: Start.
Step 2: Read: N inputs for DATA Array
Step 3: [Initialization] Set LOC := 1 and MAX := DATA[1]
Step 4: Repeat for loop from k = 1 to k ≤ N and Increment k by 1

If MAX < DATA[k], then
Set LOC := k and MAX := DATA[k].
[End of If structure]
[End of for loop]
Step 5: Print: LOC (Location of Largest number) and MAX(Value of the Largest number)
Step 6: Exit

Method 2:

Algorithm to find largest element or largest number can also be written using Go to statement.

Algorithm:

Step 1: Start
Step 2: Read: N inputs for DATA Array
Step 3: [Initialize] Set k = 1, LOC = 1 and MAX = DATA[1]
Step 4: [Increment Counter] Set K = K + 1
Step 5: if k > N, then:
Print: LOC, MAX and Exit
if MAX < DATA[k] then:
Set: LOC = k and MAX = DATA[k]
[End of inner If Structure]
[End of outer If Structure]
Step 6: Go to Step 2.

Method 3:

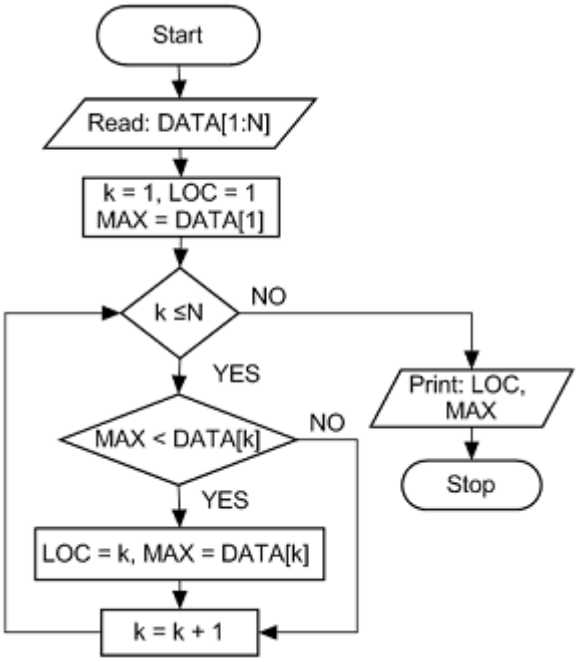
There is another method we can use to find the largest element in an array. In this method we have to sort the array in descending order and just print the first element of the array. To sort the array we can apply any sorting technique among Selection Sort, Bubble Sort or Insertion Sort. Let DATA is a numeric array. Array DATA has N elements. This algorithm finds the largest element in Array DATA. Insertion Sort algorithm has been used here to sort the Array DATA.

Step 1: Start

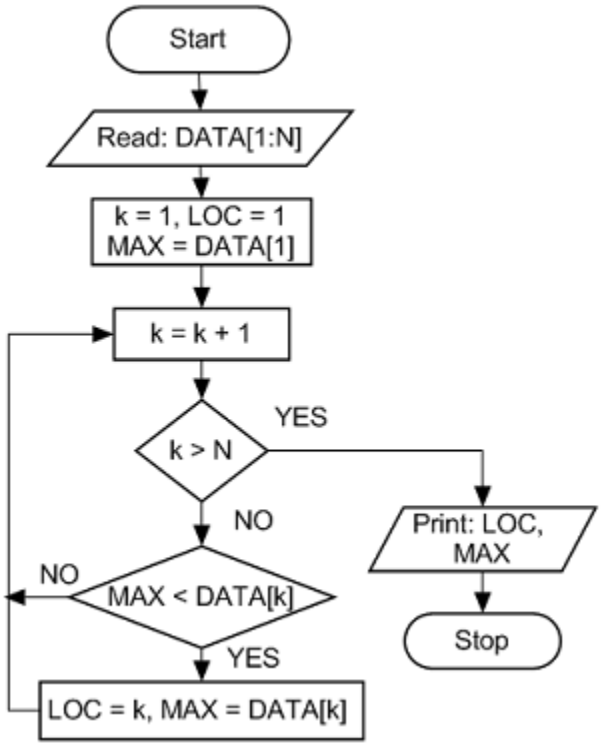
Step 2: Read: Take N numbers as Input for Array DATA
Step 3: [Sorting the Array DATA by Insertion Sort]
Repeat for i = 2 to N Increment by 1
Set: TEMP = DATA[i]
Repeat for j = i – 1 AND j >= 0 Decrement by 1
If DATA[j] < TEMP then:
Set: DATA[j+1] = DATA[j]
else
Go to Step 4
[End of If Else Structure]
Step 4: DATA[j+1] = TEMP
[End of Inner Loop]
[End of Outer Loop]
Step 5: Print: Largest Element is DATA[1]
Step 6: Exit

NOTE: If we want to print the second largest element of the array data then just print the second element of the array data. And to print the smallest element of Array, just print the last element of Array.

Method-1:


Start
Read: DATA[1:N]
k = 1, LOC = 1
MAX = DATA[1]
k ≤N
NO
YES
Print: LOC,
MAX
Stop
MAX < DATA[k]
NO
YES
LOC = k, MAX = DATA[k]
k = k + 1

Method-2:


Start
Read: DATA[1:N]
k = 1, LOC = 1
MAX = DATA[1]
k = k + 1
YES
k > N
NO
Print: LOC,
MAX
Stop
NO
MAX < DATA[k]
YES
LOC = k, MAX = DATA[k]

2.

Find Largest Number among Three Numbers

Method One:

Let X, Y and Z are three numbers. This algorithm finds the largest number among these three numbers. In this algorithm at first X is compared with Y if X is greater than Y then X is Compared with Z. If X is greater than Z, thus X is largest number otherwise Y is compared with Z. If Y is greater than Z then Y is the largest number. If X and Y both are less than Z then Z is the Largest number.

Step 1: Start

Step 2: Take Input X, Y and Z

Step 3: [comparison among X, Y and Z starts from here]

If X > Y then:

If X > Z then:

Print: X is the Largest Number.

[End of If Structure]

Else if Y > Z then:

Print: Y is the Largest Number.

Else

Print: Z is the Largest Number.

[End of If Else Structure]

Step 4: Exit

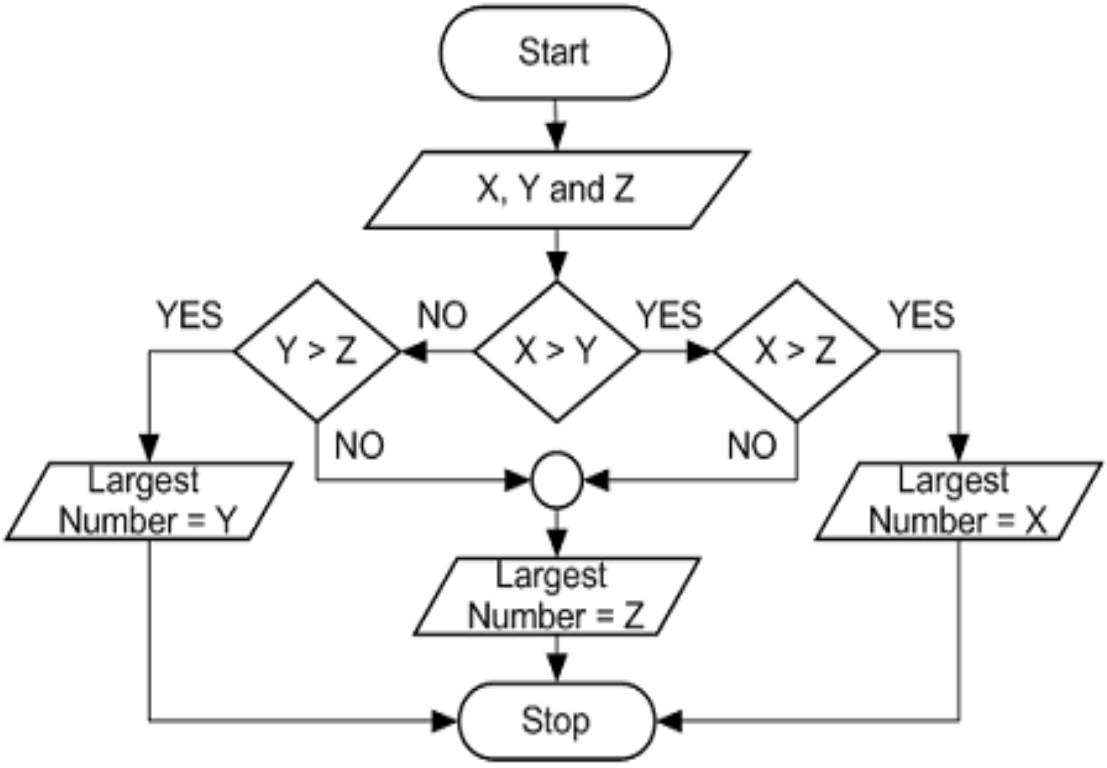
Method Two:

In this method the first number is supposed to be largest. The value of first number is assigned to a new variable BIGGEST. Then we compare BIGGEST with the Second and third numbers. If the Second number is greater than BIGGEST then value of second number is assigned to BIGGEST. Otherwise BIGGEST will not change. The same process is done with the third number. Let the three numbers be X, Y and Z. Now we can write the algorithm to find largest among X, Y and Z.

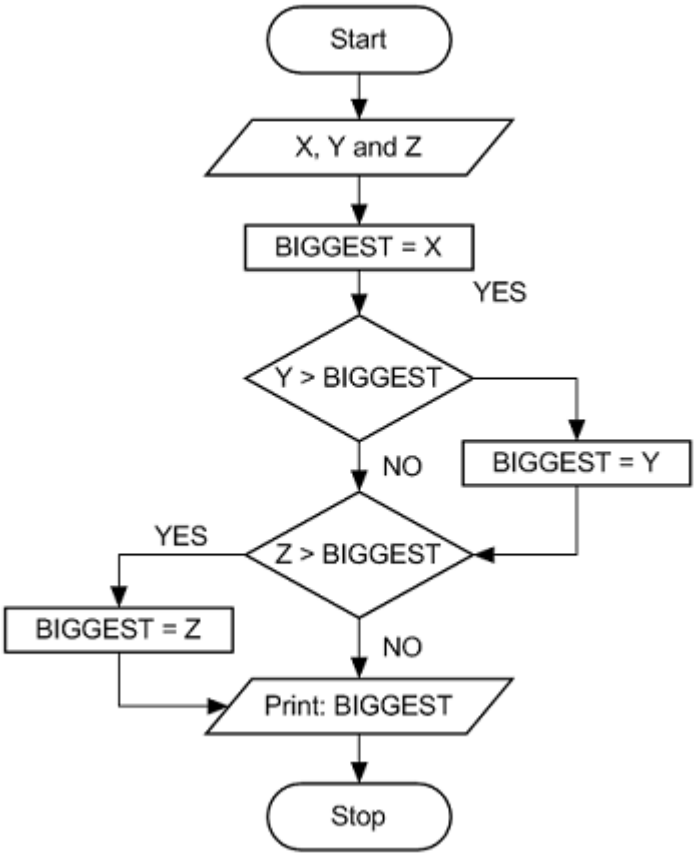
Step 1: Start
Step 2: Read: Take Inputs X, Y and Z
Step 3: [Assigning] Set: BIGGEST = X
Step 4: If Y > BIGGEST then
Set: BIGGEST = Y
[End of If Structure]
Step 5: If Z > BIGGEST then
Set: BIGGEST = Y
[End of If Structure]
Step 6: Print: The Largest Number is BIGGEST.
Step 7: Stop

FLOWCHART TO FIND LARGEST NUMBER AMONG THREE NUMBERS

Method One:



Method-2:



3.

ALGORITHM TO FIND SUM OF DIGITS OF A NUMBER

(Sum Of Digits Of A Number) Suppose N is an integer. This algorithm adds the digits of number N. As for example if N is 12345 then Sum of digits will be 15 by performing operation 1 + 2 + 3 + 4 + 5. SUM is the variable that will store the value of sum of digits of N.

Step 1: Start

Step 2: Read: Take Input N

Step 3: [Initializing SUM] Set: SUM = 0

Step 4: Repeat While N ≠ 0

Set: SUM = N%10 + SUM and

Set: N = N/10

[End of While Loop]

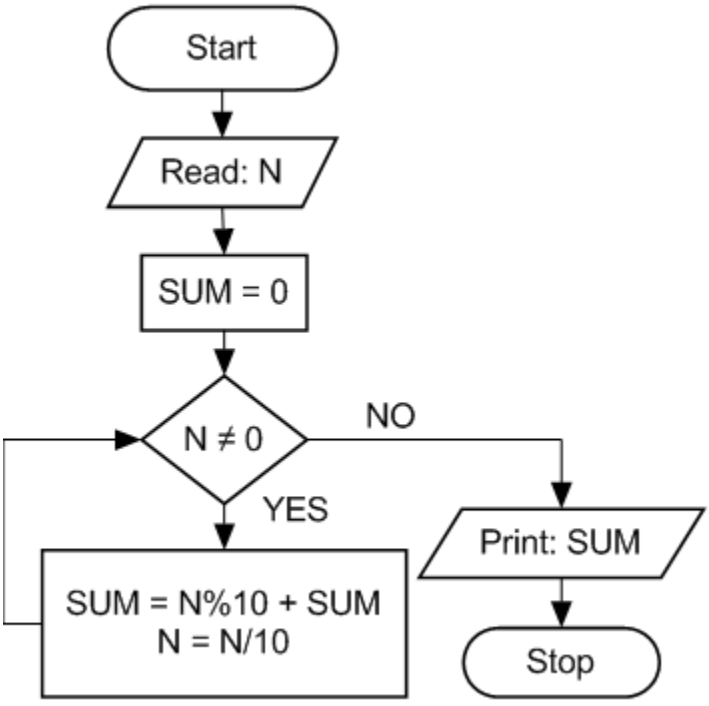
Step 5: Print: Sum of Digits of N is SUM

Step 6: Exit

Sum of digits is performed as in the while loop, N%10 is used to separate each digit from N. For example 12345%10 will produce 5. N/10 is used to remove last digit from N. For example 12345/10 will produce 1234 not 1234.5 because in this algorithm integer division is performed. The while loop runs till N is not 0.

FLOWCHART TO FIND SUM OF DIGITS OF A NUMBER

The Flowchart to find sum of digits of a number N is shown below.



4.

ALGORITHM FOR REVERSE OF A NUMBER

This Algorithm finds Reverse of a Number. Here NUMBER is input variable and REV_NUM is used to store output in computer memory. It is initialized with 0. Step 4 serves purpose of a while loop. This loop continues until NUMBER becomes 0. If it is 0 then control is transferred to Step 5. In Step 5 Reverse of the given Number is printed. A formal presentation of this algorithm is as follows.

Step 1: Start

Step 2: Read: Take input for NUMBER

Step 3: [Initialize] REV_NUM = 0

Step 4: [Check Value] If NUMBER > 0 then:

[Compute] REV_NUM = REV_NUM*10 + NUMBER%10 and

NUMBER = NUMBER/10

[Repeat loop] Go to Step 4.

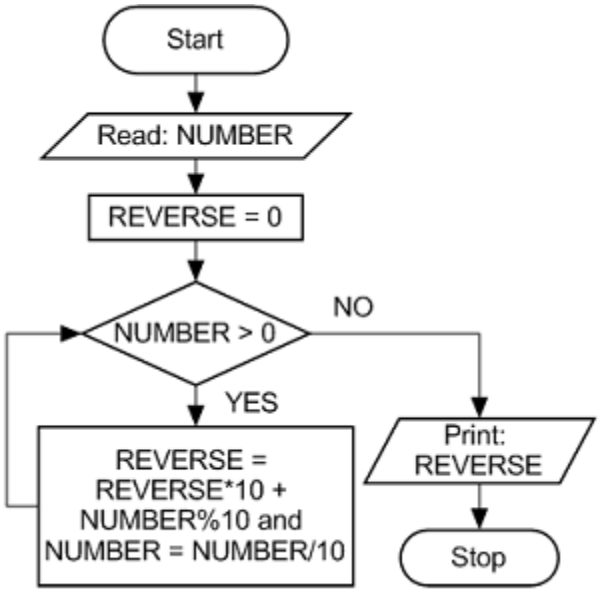
[End of If Structure]

Step 5: Print: REV_NUM

Step 6: Exit

FLOWCHART FOR REVERSE OF A NUMBER

Flowchart to find Reverse of a Number is shown below. For Example Reverse of 12345 is 54321.



5.

ALGORITHM TO FIND FACTORIAL

The product of positive integers from 1 to N is called factorial N. It can also be defined as The factorial of a number N is defined as the product of first N Natural Numbers. Factorial N is denoted by N!
Where, N! = 1.2.3..... (N-2)(N-1)N
This function is defined for all positive integers including 0. Factorial of some positive integers are given below.

0! = 1
1! = 1
2! = 1.2 = 2
3!= 1.2.3 = 6
4! = 1.2.3.4 = 24
5! = 1.2.3.4.5 = 120
6! = 1.2.3.4.5.6 = 720

We can also write 6! = 6.5! And 5! = 5.4!
Thus factorial function may be defined as.

1. If N = 0 then N! = 1
2. If N > 0 then N! = N(N – 1)

Factorial of fractions and negative numbers are not defined.

FACTORIAL ALGORITHM BY WHILE LOOP

(**Factorial Algorithm**) Suppose N be positive integer and this algorithm uses while loop structure to find the factorial of N.

Step 1: Start

Step 2: Read: Take input N

Step 3: [Initializing] Fact = 1

Step 4: If N == 0 then:

Set Fact = 1 and Go to Step 5

Else

Repeat while loop N ≠ 0

Set Fact = Fact * N and

N = N -1

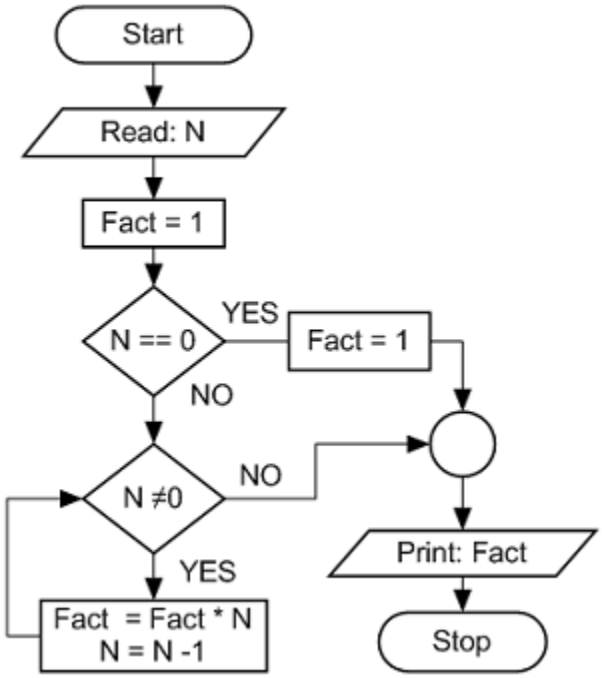
[End of while loop]

[End of If Structure]

Step5: Print: Fact

Step6: Exit

FLOWCHART TO FIND FACTORIAL BY WHILE LOOP



FACTORIAL ALGORITHM BY FOR LOOP

FACTORIAL (Fact, N)

This algorithm finds factorial of a non negative integer. Here N is a non negative integer and variable Fact stores the value of N! in the memory. In this algorithm for loop is used to find factorial.

Step 1: Start

Step 2: Read: Take Value of N

Step 3: [Initializing Fact] Set Fact = 1

Step 4: [Checking value of N] If N == 0 then

Set Fact = 1 and Go to Step 5

Else

Repeat for k = 1 to N by 1

Set Fact = Fact*k

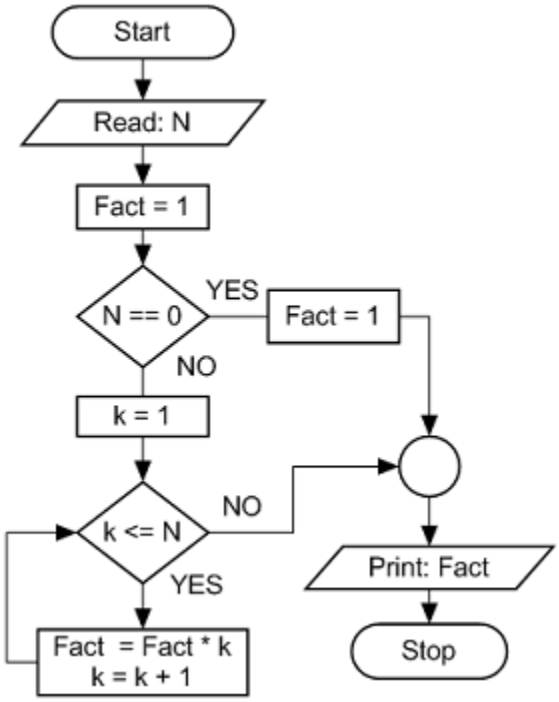
[End of for loop]

[End of If Structure]

Step 5: Print: Fact

Step 6: Exit

FLOWCHART TO FIND FACTORIAL BY FOR LOOP



ALGORITHM BY RECURSION

FACTORIAL (Fact, N)

This algorithm use recursive approach to find factorial of N.

Step 1: Start

Step 2: Read: Take input N

Step 3: If N == 0 then Print: Fact = 1 and Exit

Step 4: Call FACTORIAL (Fact, N – 1)

Step 5: Set Fact = N*Fact

Step 6: Return

6.

ARMSTRONG NUMBER ALGORITHM AND FLOWCHART

An Armstrong Number is a Number whose sum of cubes of its digits is equal to itself. The smallest Armstrong Number is a three digit number and it is 153. Other Armstrong Numbers are 370, 371 and 407.

ALGORITHM TO CHECK ARMSTRONG NUMBER

CHECK ARMSTRONG (N)

Let N be a three digit Integer. This algorithm checks that whether N is an Armstrong Number or not.

Step 1: Start

Step 2: Take Input N as Integer

Step 3: [initializing SUM] Set: SUM = 0

Step 4: Set: Number = N

Step 5: Repeat While Number ≠ 0

SUM = (Number%10)**3 + SUM

Number = Number/10

[End of While Loop]

Step 6: [Checking ?]

If SUM == N then

Print: N is an Armstrong Number.

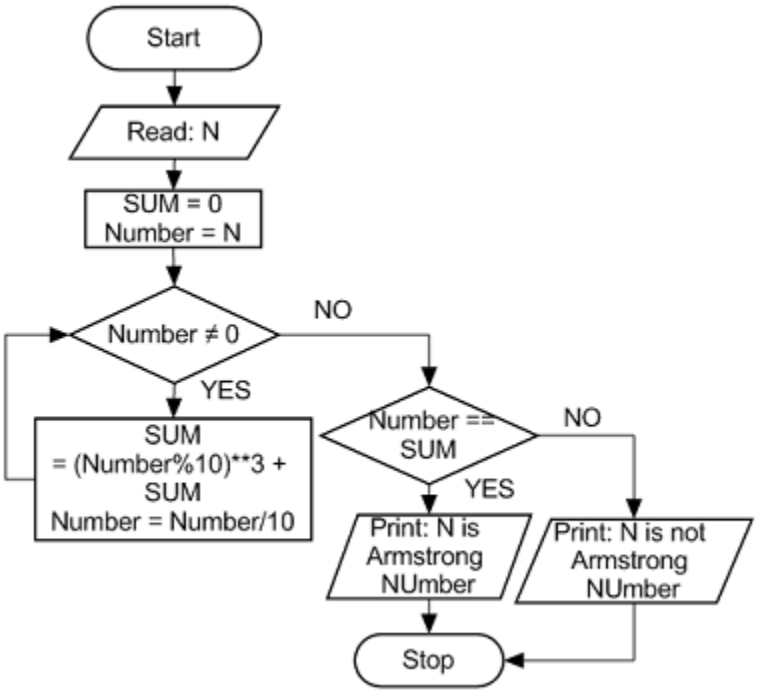
Else

Print: N is not an Armstrong Number.

[End of If-Else Structure]

Step 7: Exit

FLOWCHART TO CHECK ARMSTRONG NUMBER



ALGORITHM TO GENERATE ARMSTRONG NUMBERS

GENERATE ARMSTRONG (LL, UL)

Let LL and UL represents Lower Limit and Upper Limit respectively. This algorithm generates Armstrong Numbers between Lower Limit and Upper Limit.

Step 1: Start

Step 2: Read: Take Inputs for LL and UL

Step 3: [Initializing SUM] Set: SUM = 0

Step 4: Repeat While LL ≠ UL

Set: N = LL

Repeat While N ≠ 0

SUM = (N%10)**3 + SUM

N = N/10

[End of While Loop]

If LL == SUM then:

Print: LL

[End of If Structure]

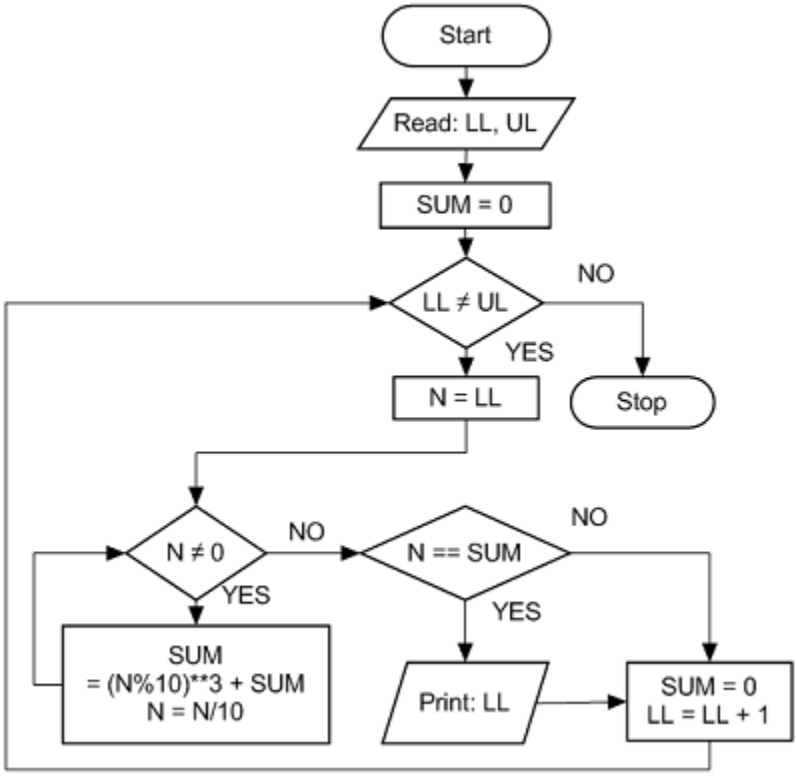
SUM = 0

LL = LL + 1

[End of While Loop]

Step 5: Exit

FLOWCHART TO GENERATE ARMSTRONG NUMBER



7.

ALGORITHM FOR PALINDROME

Any word, number or phrase which reads same from both ends is palindrome. For example 77, 121, 1331 and 12321 are palindrome numbers. Similarly Dalda and Madam are palindrome words.

ALGORITHM TO CHECK PALINDROME NUMBER

(Palindrome Number) Suppose N is a number. Now we have to find the reverse number of N. Then we have compare N with its reverse number say R. If both numbers are same then N is palindrome number. If N and R have different values then N is not a palindrome number.

Step 1: Start

Step 2: Read: N

Step 3: Initialize R = 0

Step 4: Repeat While N ≠ 0

Compute: R = R*10 + R%10

Compute: N = N/10

[End of Loop]

Step 5: Check: If N == R Then

Print: N is a Palindrome Number.

Else

Print: N is not a Palindrome Number.

[End of If Else Structure]

Step 6: Exit.

ALGORITHM TO CHECK PALINDROME NUMBER BY FUNCTION

(Palindrome Number) In this algorithm a subalgorithm also called function will be used to find the reverse of the given number.

Step 1: Start

Step 2: Read: N

Step 3: Set: R = REVERSE(N)

Step 4: Ckeck: If N == R Then

Print: N is a Palindrome number.

Else

Print: N is not a Palindrome number.

Step 5: Exit

Subalgorithm

REVERSE(n)

Step 1: [initialize] n = 0

Step 2: Repeat while n ≠ 0

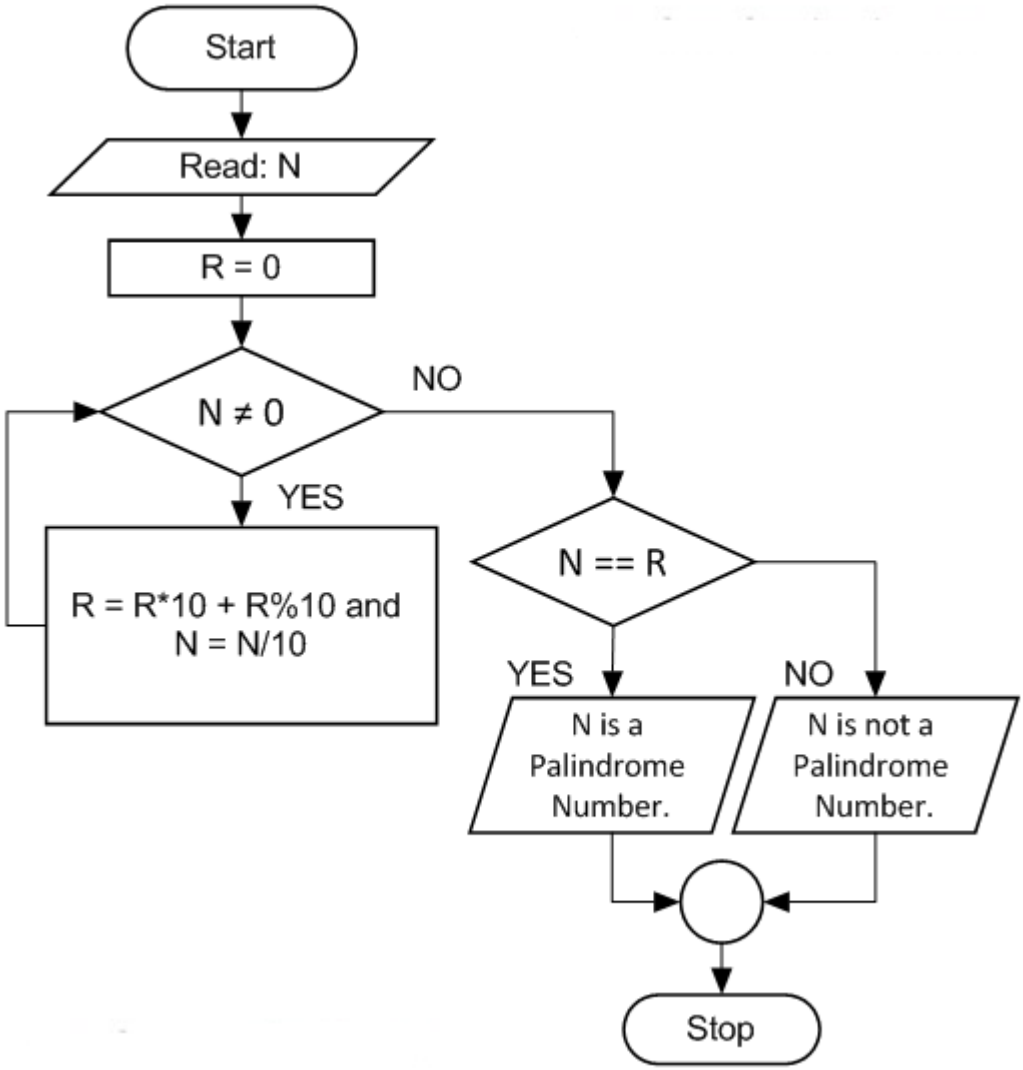
Compute: n = n*10 + n%10

Compute: n = n/10

[End of While Loop]

Step 3: Return(n)

FLOWCHART TO CHECK PALINDROME NUMBER



ALGORITHM TO CHECK STRING PALINDROME

(Palindrome String) Suppose W is a string variable of dynamic length. W can store a word of any length. Let W' can store the data of W in reversed order. W will be a palindrome string if W and W' read same. Otherwise W is not a palindrome string. The algorithm is as follows.

Step 1: Start

Step 2: Read: W

Step 3: [initialize] F = 0

Step 3: [Compute Length of String] Set: L = LENGTH(W)

Step 5: Repeat for I = 1 to L Increment by 1

Set: W'[I++] = W[L–]

[End of For Loop]

Step 6: Repeat for I = 1 to L Increment by 1

[Checking] If W[I] ≠ W'[I] Then

F = 1 and Break the Loop

[End of If Structure]

I = I + 1

[End of For Loop]

Step 7: [Checking] If F == 0 Then

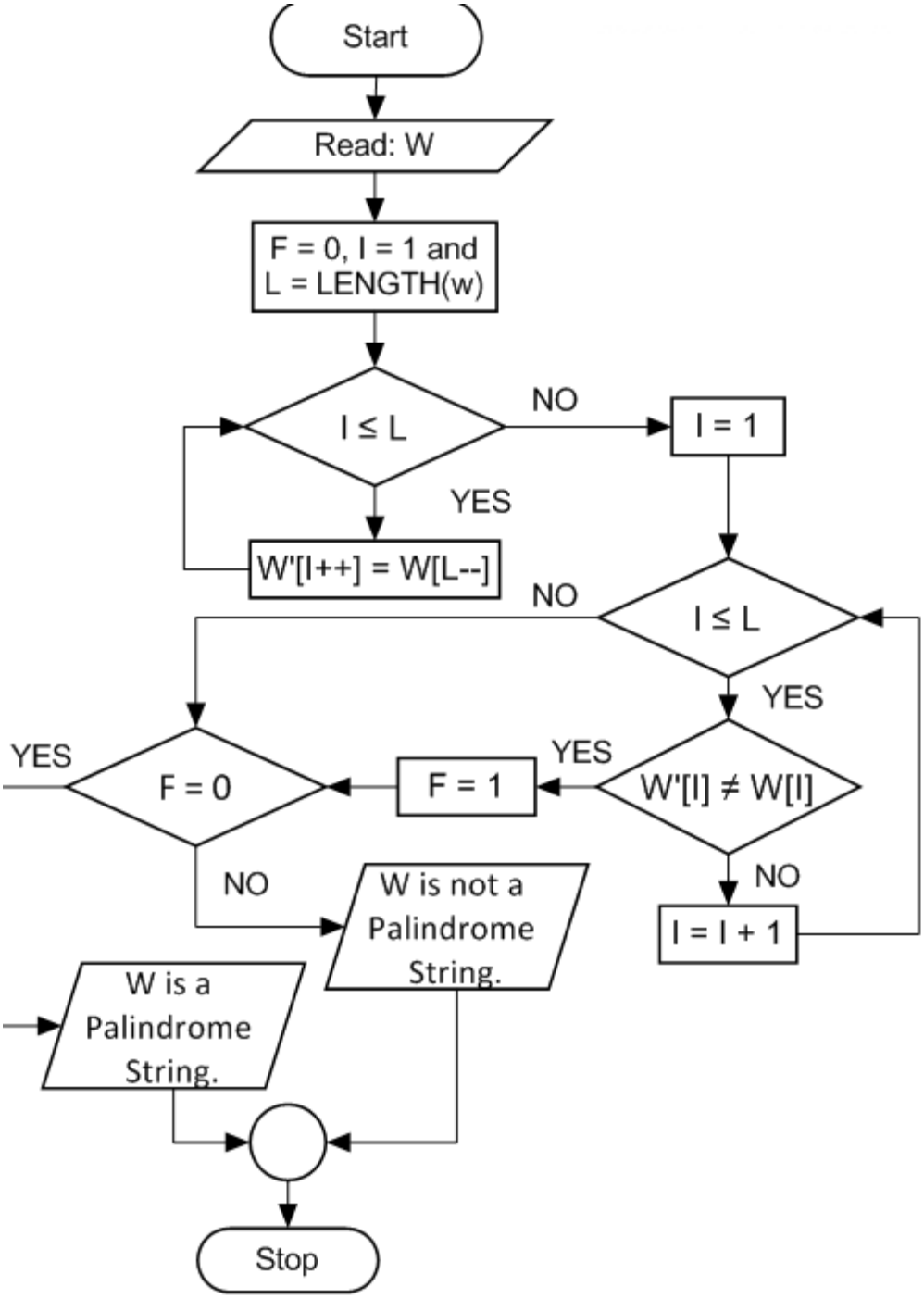
Print: W is Palindrome.

Else

Print: W is not Palindrome.

Step 8: Exit

FLOWCHART TO CHECK STRING PALINDROME



8.

PERFECT NUMBER ALGORITHM AND FLOWCHART

A number is said to be a perfect number if the sum of factors of the number (**excluding number itself as a factor**) is equal to the number itself. Example of perfect numbers are 6, 28, 496, 8128 and so on. The factors of a given number are those numbers which divide the given number completely. For example factors of 28 are 1, 2, 4, 7, 14 and 28. Now the sum 1 + 2 + 4 + 7 + 14 = 28. We can see that 28 is not included in the sum.

ALGORITHM TO CHECK PERFECT NUMBER

(**Perfect Number)** Let N be positive Integer. This algorithm will check whether N is perfect number or not. To do so we have to find the factors of N. And then we have to add all factors except N. Now we have to compare N with sum of factors (Denoted By SOF in Algorithm). If N is equal to SOF then N is a perfect number. And if N is not equal to SOF then N is not a perfect number.

Step 1: Start

Step 2: [Take Input] Read: N

Step 3: [Initialize] Set: SOF = 0 and I = 1

Step 4: Repeat While I < N

Check If N%I == 0 Then

Compute: SOF = SOF + I

[End of If Structure]

Compute: I = I + 1

[End of While Loop]

Step 5: Check If N == SOF Then

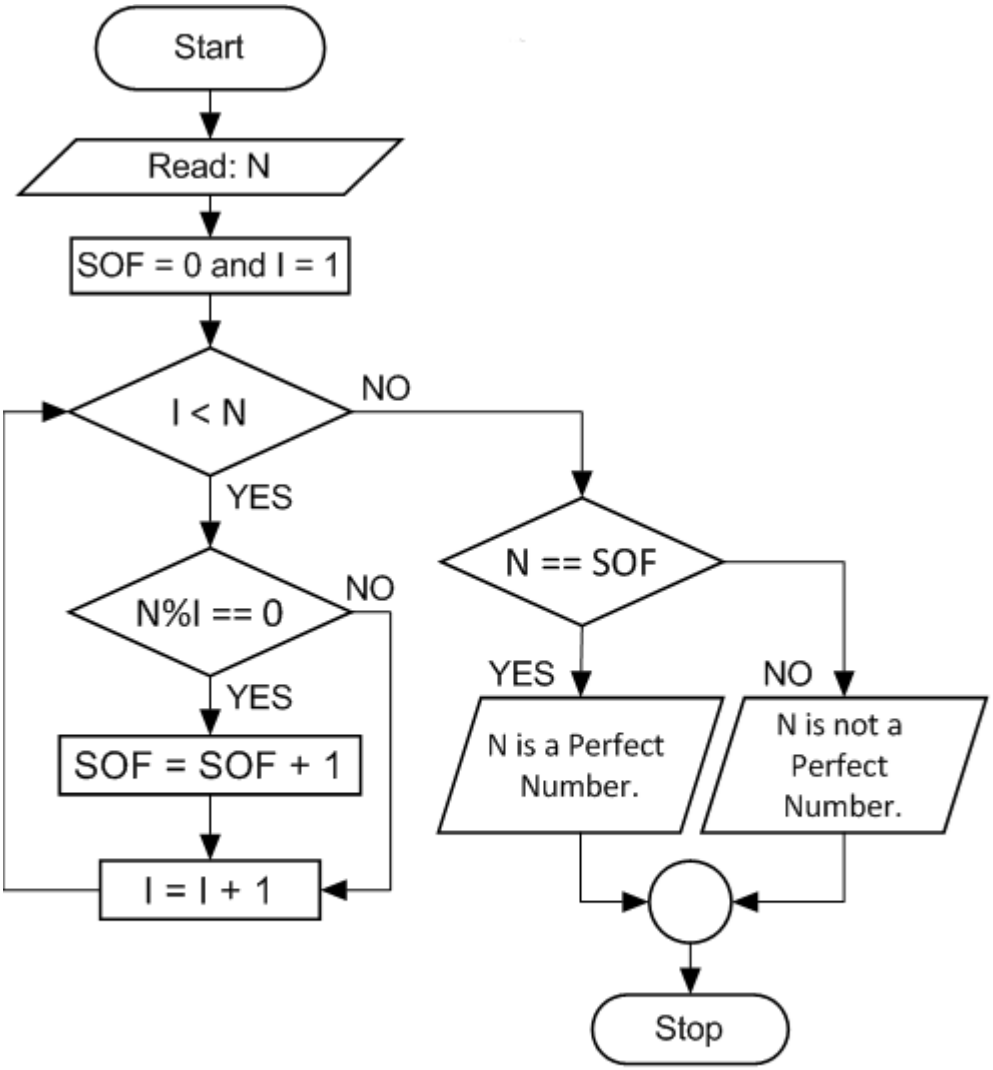
Print: N is a Perfect Number.

Else

Print: N is not a Perfect number.

Step 6: Exit

FLOWCHART TO CHECK PERFECT NUMBER



ALGORITHM TO GENERATE PERFECT NUMBERS

(Generate Perfect Numbers) This algorithm generates perfect numbers within a range. Let R be the range. Each number from 1 to range R is checked whether it is a perfect number or not. If a number is found to be a perfect number then it is printed.

Step 1: Start

Step 2: [Take Input] Read: R

Step 3: [Initialize] Set: SOF = 0 and a = 1

Step 4: Repeat While a ≠ R

Set: N = a and I = 1

[Sub Step 4a] Repeat While I < N

Check If N%I == 0 Then

Compute: SOF = SOF + I

[End of If Structure]

Compute: I = I + 1

[End of Sub Step 4a][End of Inner While Loop]

[Sub Step 4b] Check If N == SOF Then

Print: a

[End of Sub Step 4b] [End of If Structure]

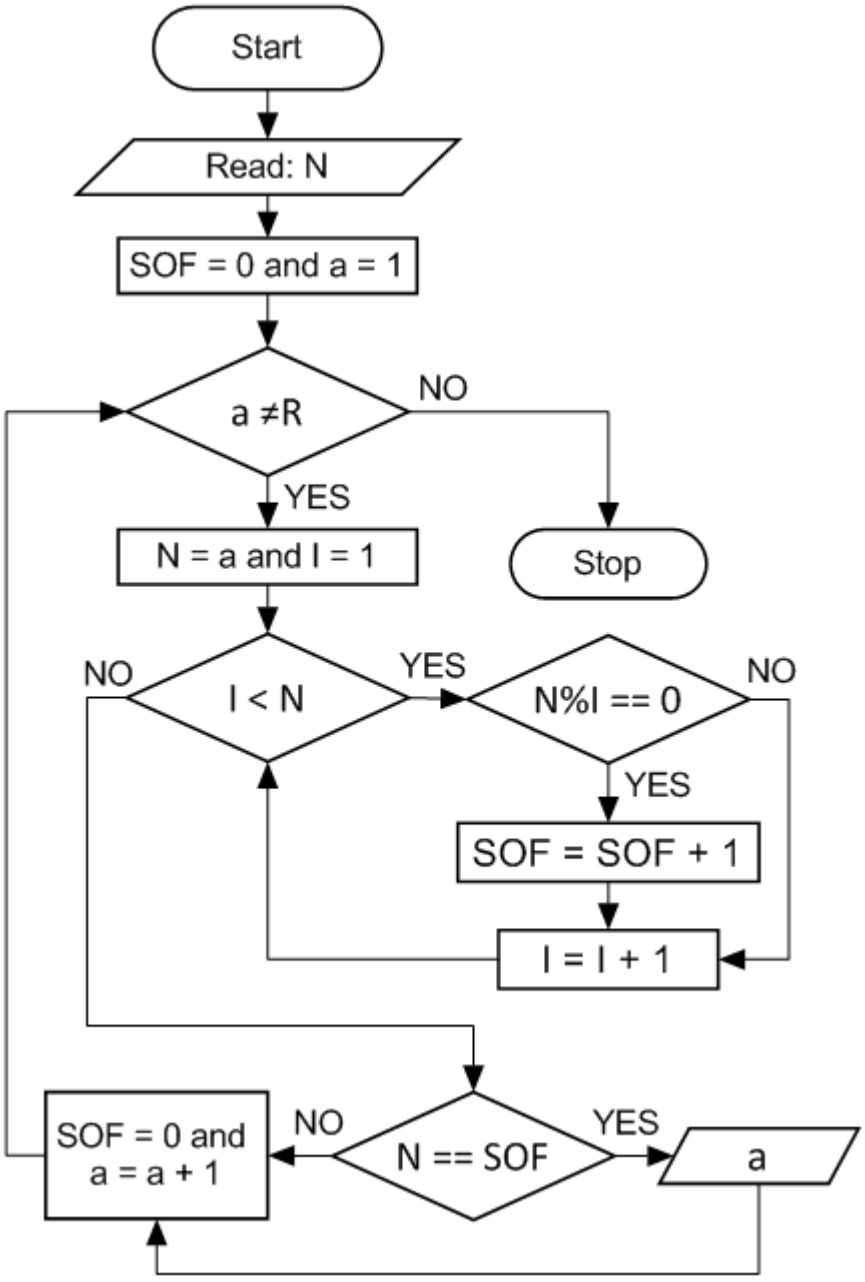
Set: SOF = 0

Compute: a = a + 1

[End of Outer While Loop]

Step 5: Exit

FLOWCHART TO GENERATE PERFECT NUMBERS



9

ALGORITHM AND FLOWCHART FOR COPRIME NUMBER

A coprime numbers comes in pair of integers. Thus they are defined as those paired numbers which can be divided by the common number 1 are called coprime numbers. Suppose a and b are two positive integers and they are divisible by only 1. Then a and b are co-prime number. For example 3 and 5 are only divisible by 1. Thus 3 and 5 are co-prime number. If a and b are divisible by some more numbers other than 1 then they are not co-prime number. For example 6 and 10 are not co-prime number. Because 6 and 10 are divisible 2 also other than 1. A coprime number is also known as relatively prime and mutually prime number.

ALGORITHM TO CHECK COPRIME NUMBER

Suppose X and Y are two Integers. We have to check that whether these two numbers are coprime numbers. A number of methods are listed below.

FIRST METHOD:

(Check Coprime Number) Let I is a counter and it is initialized from 2. F is a Boolean variable and its value is 0. In this algorithm both numbers X and Y will be divided by I simultaneously. This division process will go on till I becomes equal to variable X. If X and Y are divisible for any value of I. Then value of F is changed to 1 and the loop will discontinue. If F is equal to 0 then X and Y are co-prime number. And if F is equal to 1 then both numbers are not co-prime.

Step 1: Start

Step 2: [Take Inputs] Read: X and Y

Step 3: [Initializing Variables] Set: I = 2 and F = 0

Step 4: Repeat While I <= X

Check If X%I == 0 AND Y%I == 0 Then

Set: F = 1 and break the loop

[End of If Structure]

Compute: I = I + 1

[End of While Loop]

Step 5: Check If F == 1 Then

Print: X and Y are Co-prime number.

Else

Print: X and Y are not Co-prime number.

Step 6: Exit

SECOND METHOD:

(Check Coprime Number) In this algorithm we will use an alternative definition of coprime number. If the GCD of two numbers is 1 then both number are co-prime. This algorithm first finds the GCD of X and Y. If GCD of X and Y is 1 then X and Y are coprime number. And if GCD of X and Y is some other value than 1 then X and Y are not coprime.

Step 1: Start

Step 2: [Take Inputs] Read: X and Y

Step 3: Repeat While A ≠ B

Step 3a: Check If A > B Then

Compute: A = A – B

Else

Set: Temp = A, A = B and B = Temp

Go To Step 3a

[End of If Else Structure]

[End of While Loop]

Step 4: set: GCD = A

Step 5: Check If GCD == 1 Then

Print: A and B are Co-prime number.

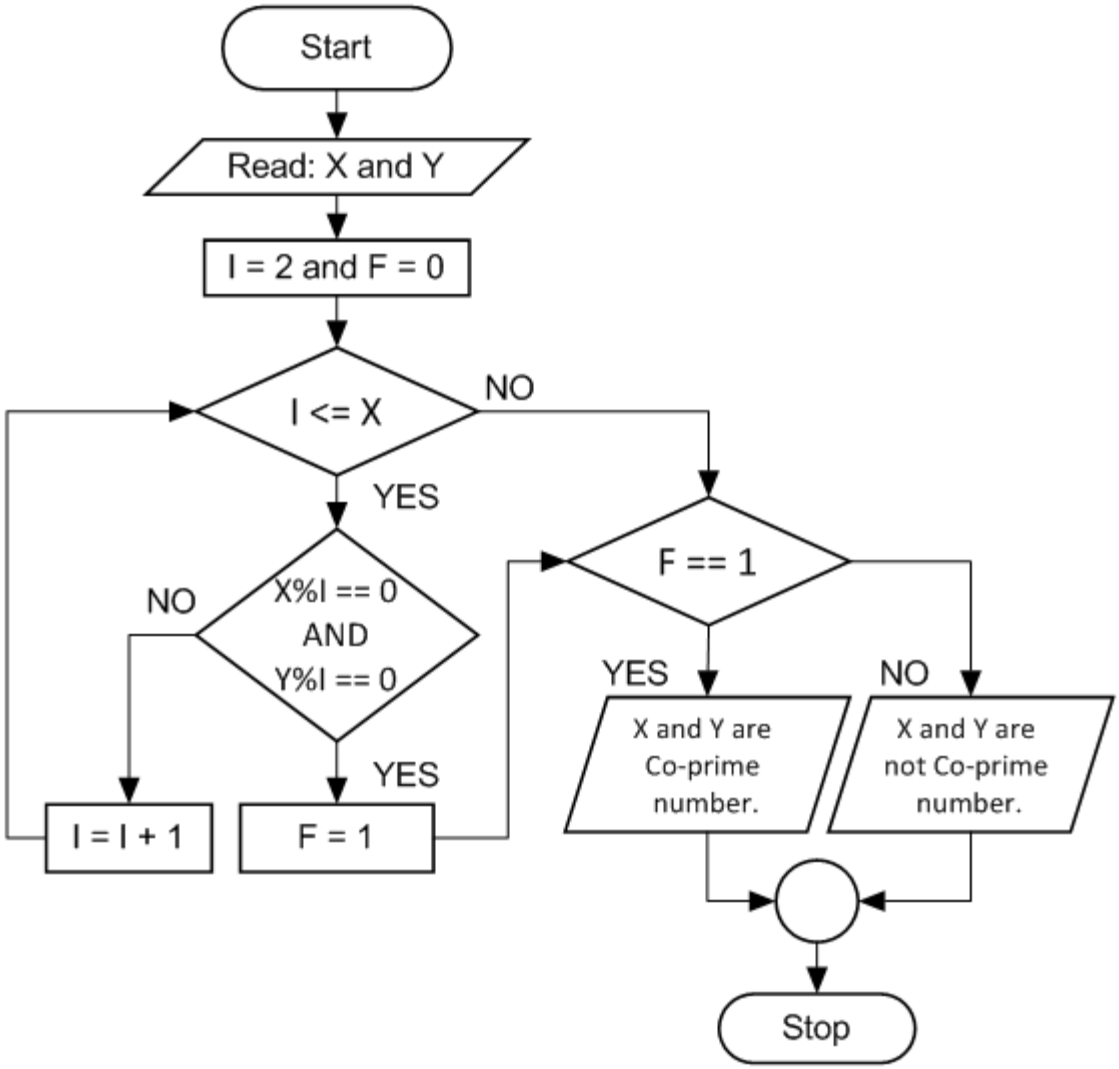
Else

Print: A and B are not Co-prime number.

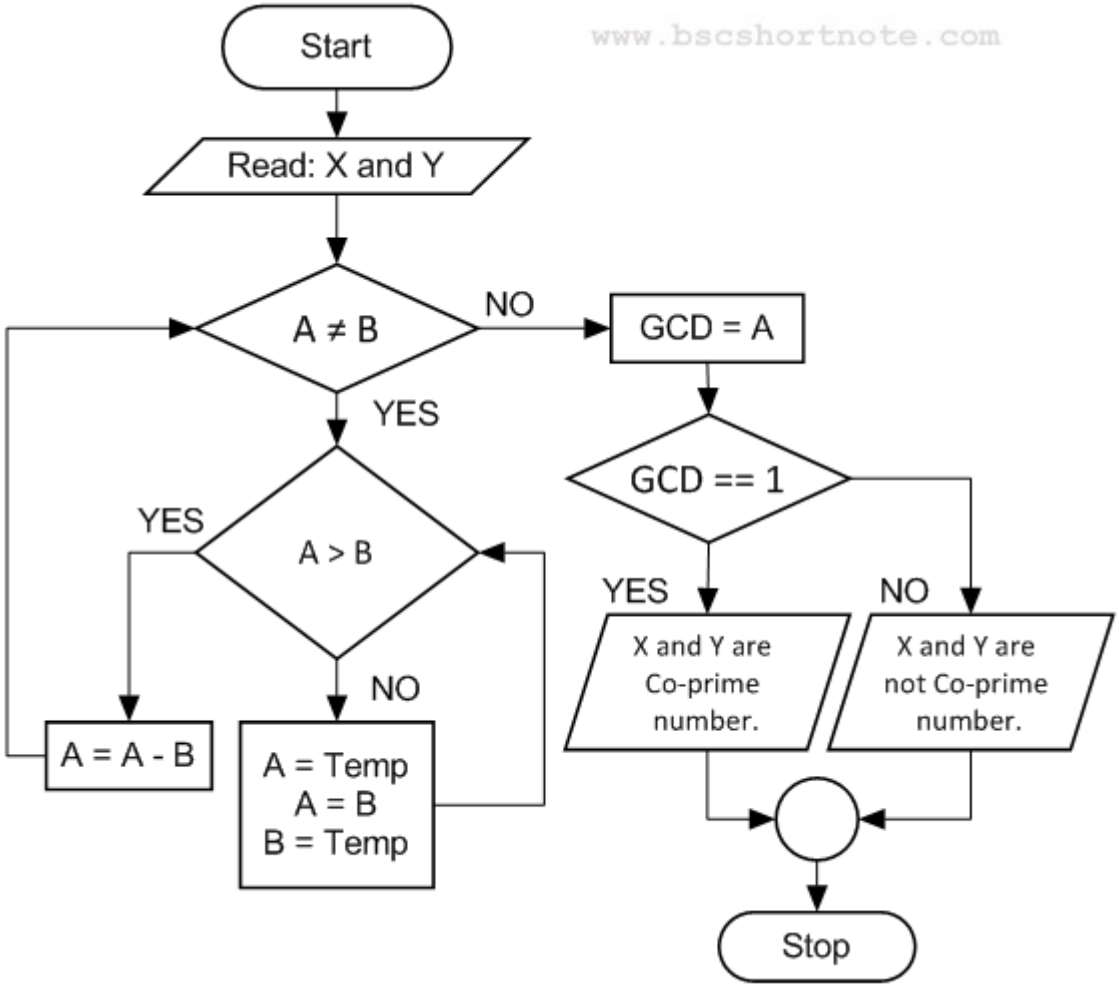
Step 6: Exit

FLOWCHART TO CHECK COPRIME NUMBER

FIRST METHOD:



SECOND METHOD:



10.

ALGORITHM AND FLOWCHART FOR COMPOSITE NUMBER

Those numbers which are divisible by other than itself and 1 are known as composite numbers. The number 4 can is divisible by 1, 2 and 4 therefore 4 is a composite number. But 7 is divisible by only 1 and 7 thus 7 is not a composite number. For example 8, 6, 9, 25, 102, 502 etc. The smallest composite number is 4. 1, 2 and 3 are not composite numbers. 2 and 3 are prime numbers.

A composite number can also be defined as if a number has factors other 1 and itself then it is a composite number. For example factors of 12 are 1, 2, 3, 4, 6 and 12 thus 12 is a composite number. The factors of a number

are those numbers which divide the number completely (leaving no remainder)

ALGORITHM TO CHECK COMPOSITE NUMBER

(Composite Number) Suppose N is a positive Integer. N will be a composite number if it has a factor or factors other than 1 and N. In this algorithm we will divide number N from numbers 2 to N – 1 one by one. If N is divisible by any number from 2 to N – 1 then N will be a composite number. Or we can say if we N has a factor from 2 to N – 1 then N is a composite number.

Step 1: Start

Step 2: [Take Input] Read: N

Step 3: [Initialize Counter] I = 2 and F = 0

Step 4: Repeat While I < N

Check If N%I == 0 Then

Set: F = 1 and Break the Loop

[End of If Structure]

Compute: I = I + 1

[End of While Loop]

Step 5: Check If F == 1 Then

Print: N is a composite Number.

Else

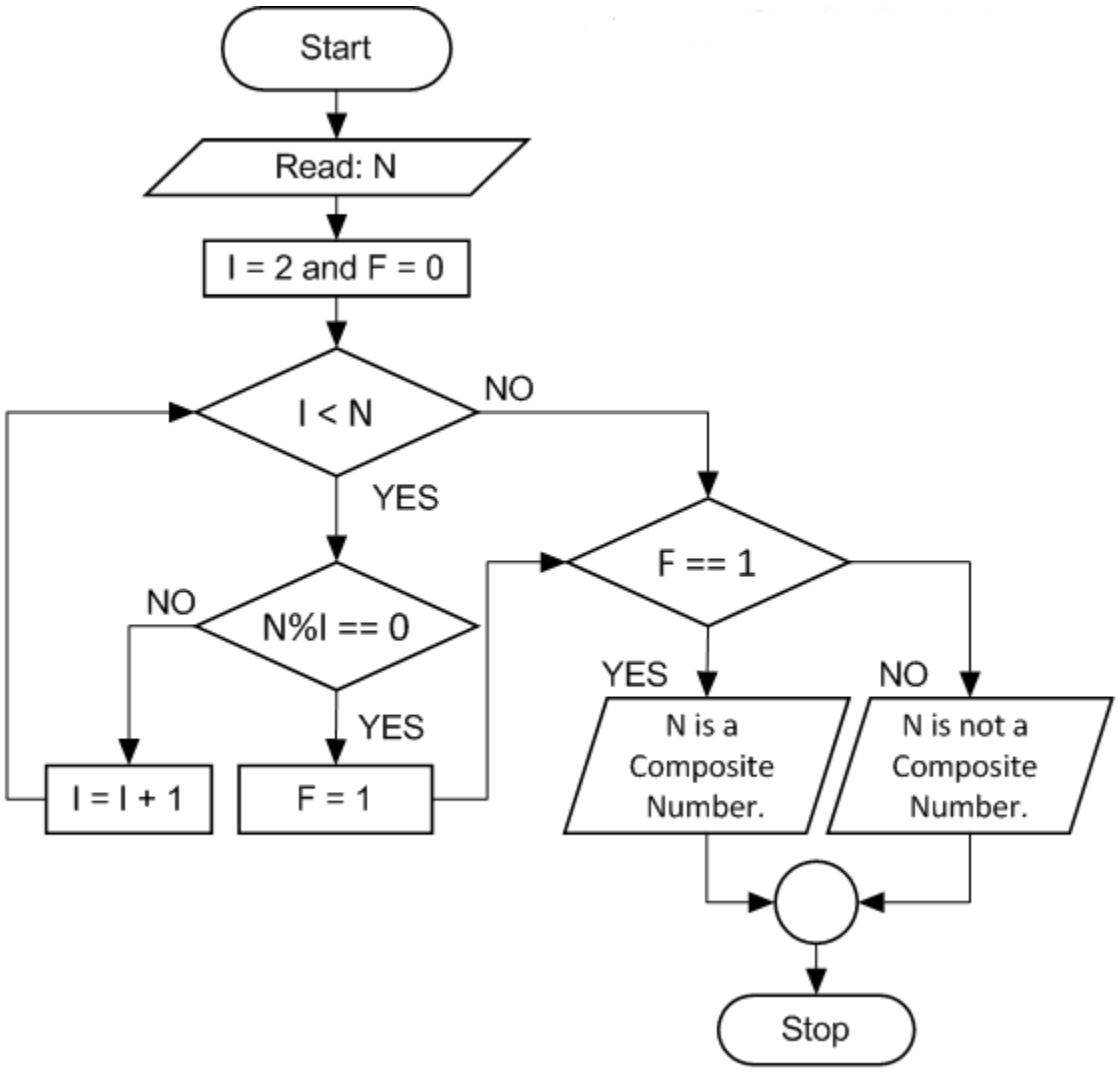
Print: N is not a Composite Number.

[End of If Else Structure]

Step 6: Exit

FLOWCHART TO CHECK COMPOSITE NUMBER

The flowchart to check whether a given number is a composite number or not is shown below.



ALGORITHM TO GENERATE COMPOSITE NUMBERS

(Generate Composite Numbers) Since the smallest composite number is 4 therefore the Lower Limit (LL) will start from 4 and Upper Limit (UL) will

be chosen by the user. If the value stored in LL is a composite number then it will be printed otherwise values stored in LL will be increased by 1. The above process will continue untill LL is not equal to UL.

Step 1: Start

Step 2: [Take Input For Range] Read: UL

Step 3: Set: LL = 4

Step 4: Repeat While while LL ≠ UL

Step 4a: Set: N = LL, F = 0 and I = 2

Step 4b: Repeat While I < N

Check If N%I == 0 Then

Set: F = 1 and Break the Inner while Loop

[End of If structure]

Compute: I = I + 1

[End of Inner While Loop]

Step 4c: Ckeck If F == 1 Then

Print: N

[End of If structure]

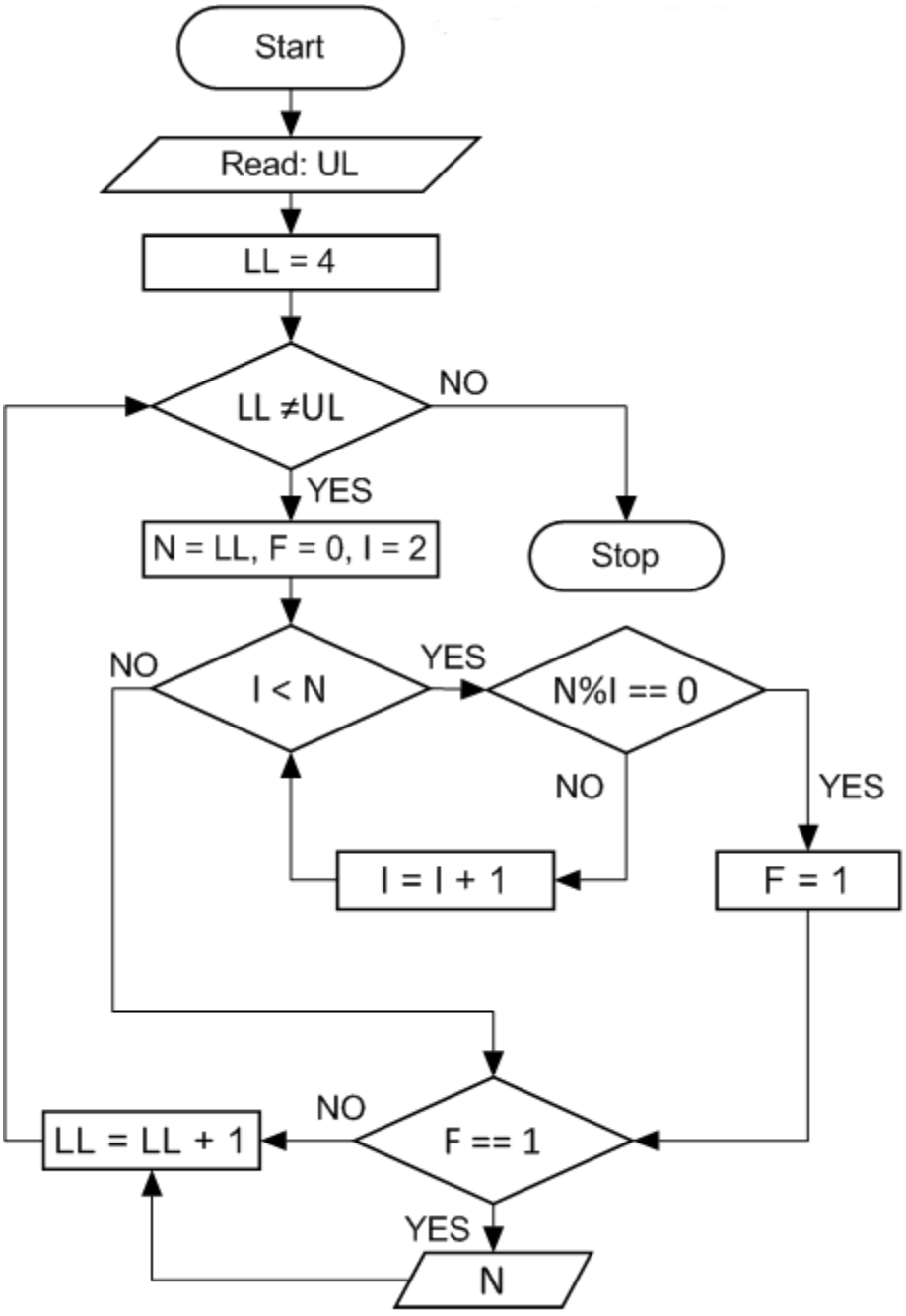
Step 4d: Compute: LL = LL + 1

[End of Outer While Loop]

Step 5: Exit

FLOWCHART TO GENERATE COMPOSITE NUMBERS

The given below flowchart generates composite numbers with a given range.



11.

ALGORITHM AND FLOWCHART FOR PRIME NUMBER

Every Prime number has two factors and these are 1 and the number itself. If N is Prime Number then its factors will be 1 and N. Thus a Prime Number is divisible by 1 and itself. Example of Prime Numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 and so on. The smallest Prime Number is 2 and it is an Even Number. Except 2 all Prime Numbers are Odd Numbers.

ALGORITHM TO CHECK PRIME NUMBER

(Check Prime Number) Let N be a Positive Integer. The Number N will be a Prime Number if it is only divisible by 1 and N. If N is divisible by one or more numbers other than 1 and N then it will not be Prime Number. Since a number is always divisible by 1 and itself. Thus in this algorithm N is divided by 2 then 3 then 4 till N – 1. If N is divisible by any number form 2 to N – 1 then is not a Prime Number. Otherwise it is a Prime Number.

Step 1: Start

Step 2: [Take Input] Read: N

Step 3: [Initializing Counter and Boolean Variable] Set: I = 2 and Flag = 0

Step 4: Repeat this step While I < N

Check If N%I == 0 Then

Set: Flag = 1 and Break the Loop

[End of If Structure]

[Incrementing Counter] Compute: I++

[End of While Loop]

Step 5: Check If Flag == 1 Then

Print: N is not a Prime Number.

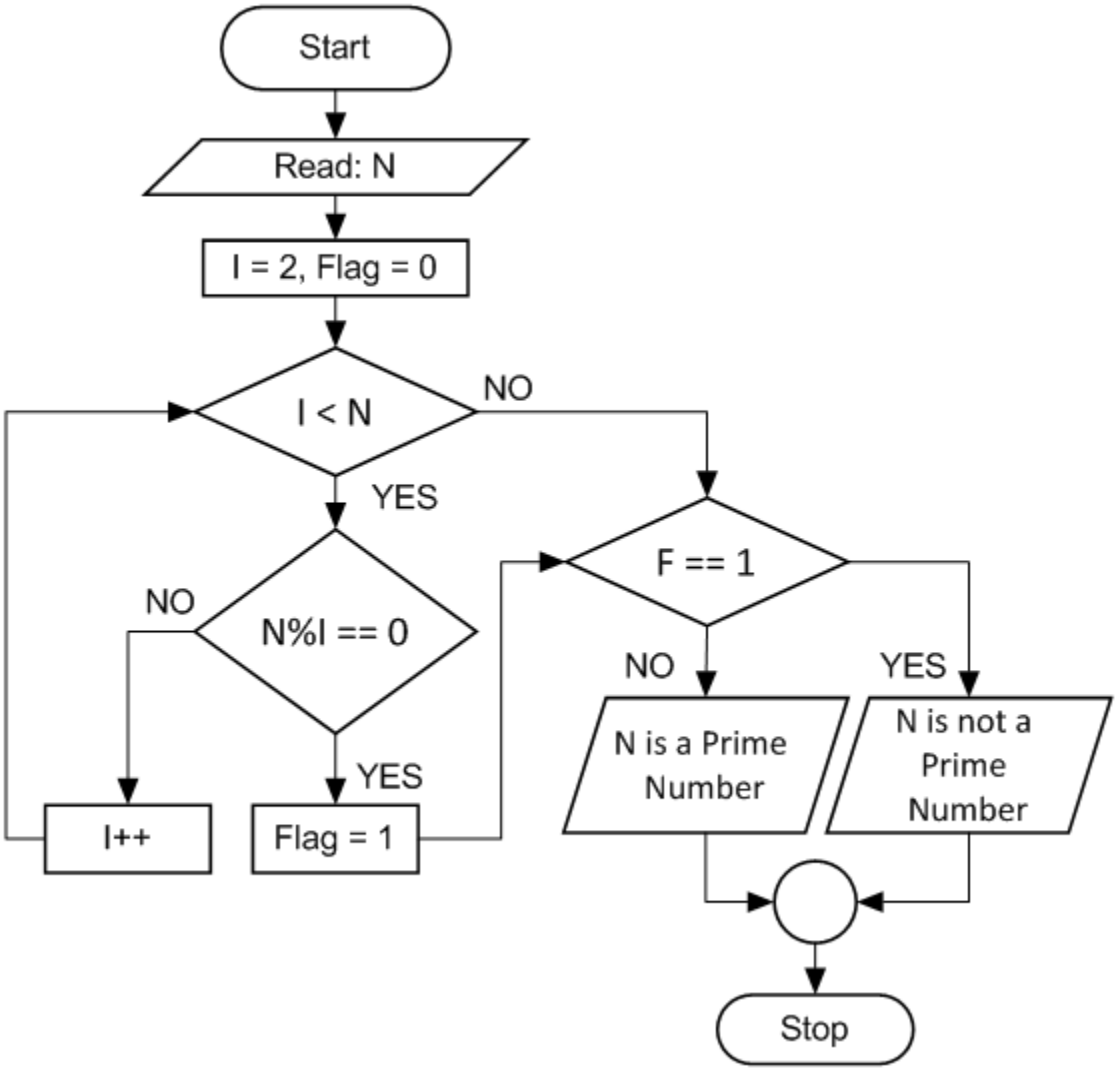
Else

Print: N is a Prime Number.

[End of If Else Structure]

Step 6: Exit

FLOWCHART TO CHECK PRIME NUMBER



ALGORITHM TO GENERATE PRIME NUMBERS

(Generate Prime Numbers) Since the smallest Prime number is 2 therefore the Lower Limit (LL) of the range will start from 2. And the Upper Limit (UL) of the range will be given by the user. Since 2 is already a Prime number therefore the series will start from 2. If the value LL is not a Prime number then it will not be printed. Thus value stored in LL will be increased by 1 after end of each loop. The above process will continue untill value of LL is not equal to value of UL.

Step 1: Start

Step 2: [Take Input For Range] Read: UL

Step 3: Set: LL = 2

Step 4: Repeat While while LL ≠ UL

Step 4a: Set: N = LL, Flag = 0 and I = 2

Step 4b: Repeat While I < N

Check If N%I == 0 Then

Set: Flag = 1 and Break the Inner while Loop

[End of If structure]

Compute: I++ [Incrementing Counter]

[End of Inner While Loop]

Step 4c: Ckeck If Flag == 0 Then

Print: N

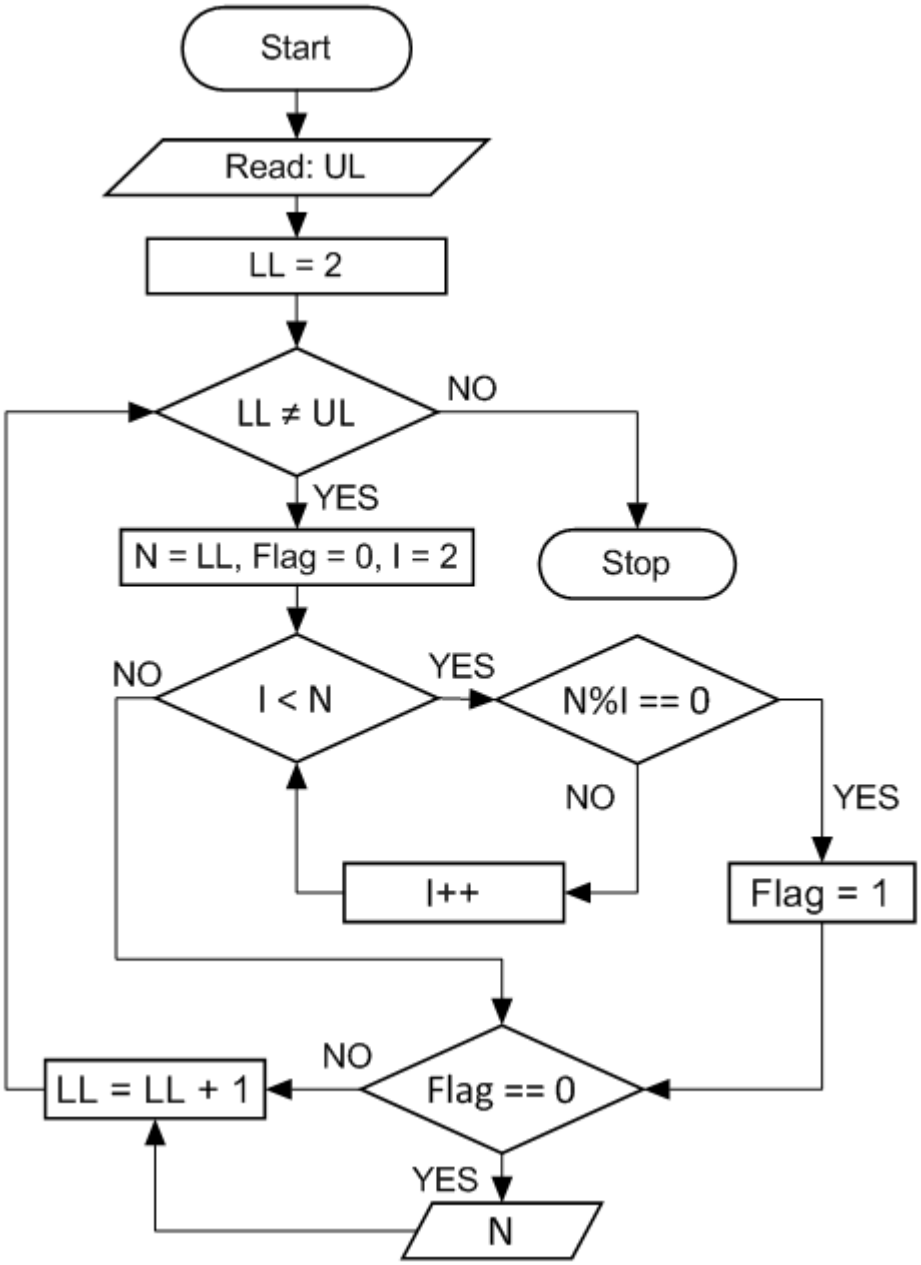
[End of If structure]

Step 4d: Compute: LL = LL + 1 [Incrementing Lower Range]

[End of Outer While Loop]

Step 5: Exit

FLOWCHART TO GENERATE PRIME NUMBER



ALGORITHM TO FIND GREATEST COMMON DIVISOR

The GCD or greatest common divisor of two or more positive numbers is the greatest number which divides each number completely (Leaving no remainder). For example 6 and 18 are divisible by 2, 3, and 6. But 6 is the greatest number among 2, 3 and 6. Thus 6 is the GCD of 6 and 18. Similarly GCD of 10, 15 and 35 is 5. GCD is also known as HCF or Highest Common Factor.

ALGORITHM TO FIND GCD OF TWO NUMBERS

(GCD of Two Numbers) Let A and B are two positive integers. This algorithm finds the GCD of A and B. In this algorithm first A and B will be compared. If A is equal to B then GCD is also equal to A or B. If A is greater than B then A = A – B and repeat the above process. If A is less than B then interchange the values of A and B, and the repeat the same process when A is greater than B. This process will be going on till A is not equal to B. After completion of these steps the GCD will be equal to changed value A.

Step 1: Start

Step 2: [Take Inputs] Read: A and B

Step 3: Repeat While A ≠ B

Step 3a: Check If A > B Then

A = A – B

Else

[Swapping is Done here] Set: Temp = A, A = B and B = Temp

Goto Step 3a.

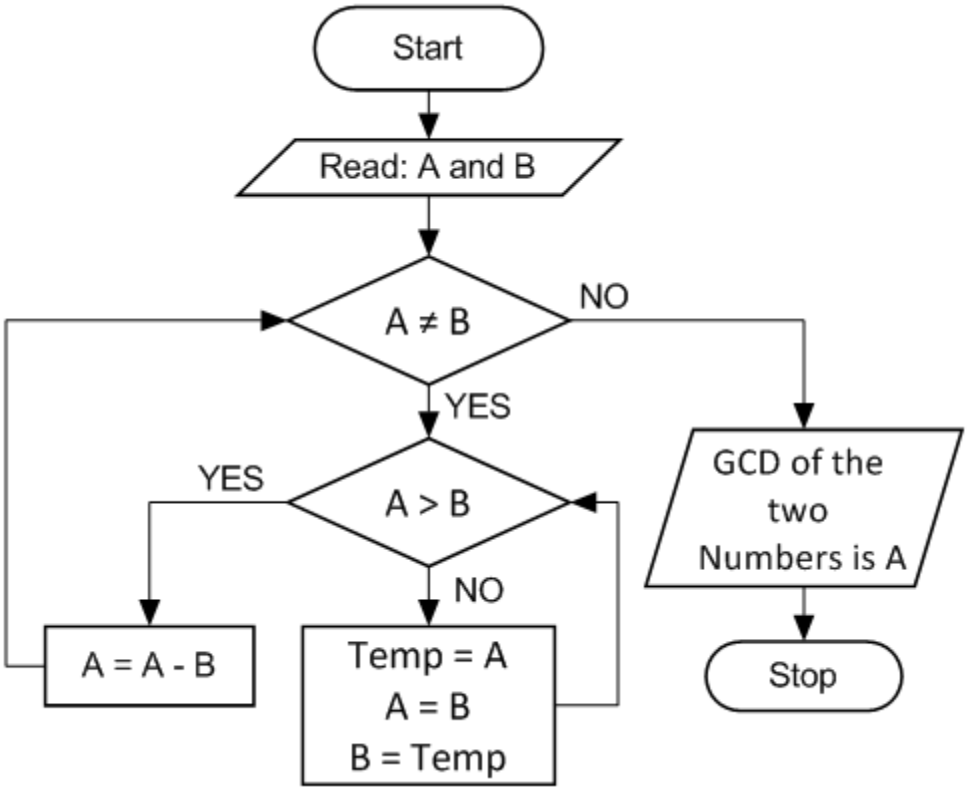
[End of If Else Structure]

[End of While Loop]

Step 4: Print: GCD of the two Numbers is A.

Step 5: Exit

FLOWCHART TO FIND GCD OF TWO NUMBERS



EUCLID'S ALGORITHM TO FIND GCD OF TWO NUMBERS

Euclid's algorithm was given by an ancient Greek mathematician Euclid. This algorithm is used to find the GCD of two number by reducing them untill they becomes equal. This algorithm will find the GCD of numbers A and B.

Step 1: Start

Step 2: [Take Inputs] Read: A and B

Step 3: Compute: X = A – (A/B)*B

Step 4: Repeat While X ≠ 0

Set: A = B

Set: B = X

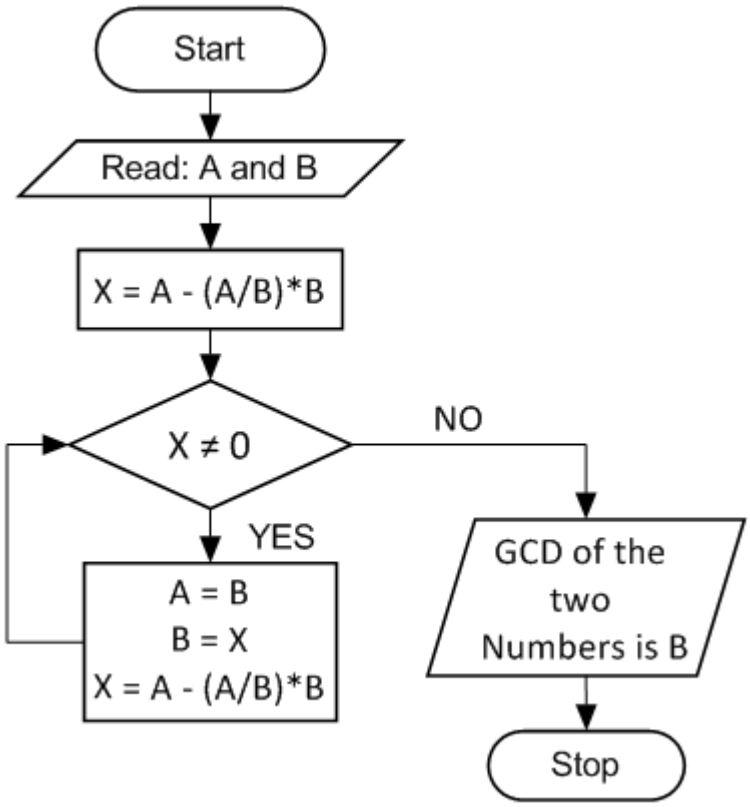
Compute: X = A – (A/B)*B

[End of While Loop]

Step 5: Print: GCD of the two Numbers is B.

Step 6: Exit

FLOWCHART TO FIND GCD OF TWO NUMBERS BY EUCLID'S ALGORITHM



ALGORITHM TO FIND GCD OF THREE NUMBERS

(GCD of Three Numbers) Let A, B and C are three positive Integers. This algorithm will find GCD of A, B and C. The GCD of A, B and C will be found in two sections. In the first section GCD of A and B will be found. And in the second section GCD of C and the value obtained from GCD of A and B will be found. And at the end we will get the GCD of A, B and C.

Step 1: Start

Step 2: [Take Inputs] Read: A, B and C

Step 3: Compute: X = A – (A/B)*B

Step 4: Repeat While X ≠ 0

Set: A = B

Set: B = X

Compute: X = A – (A/B)*B

[End of While Loop]

[GCD of A and B is stored in B]

Step 5: Compute: X = B – (B/C)*C

Step 6: Repeat While X ≠ 0

Set: B = C

Set: C = X

Compute: X = B – (B/C)*C

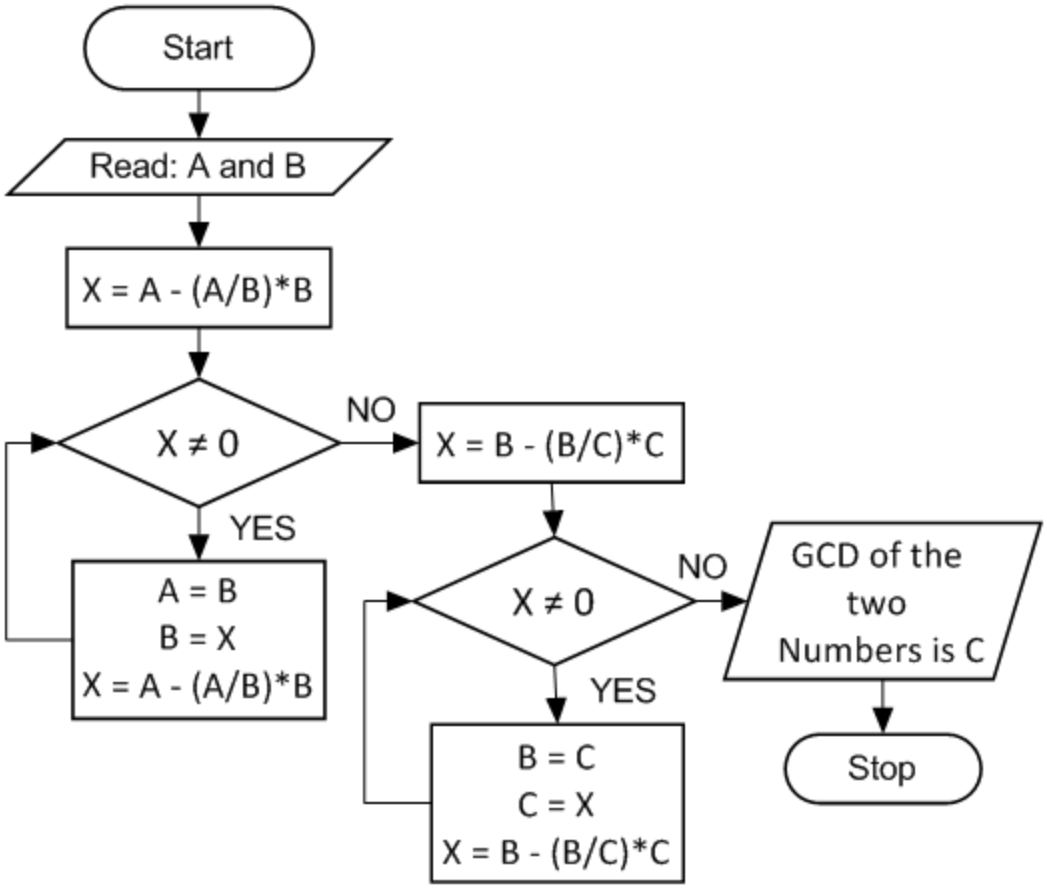
[End of While Loop]

[GCD of B and C is stored in C]

Step 7: Print: GCD of the Three Numbers is C.

Step 8: Exit

FLOWCHART TO FIND GCD OF THREE NUMBERS BY EUCLID'S ALGORITHM



12.

SORTING ALGORITHM

There are different sorting algorithm like Selection Sort, Bubble Sort, Insertion Sort, Merge Sort and Heap Sort. This topic only covers two of

them these are Selection Sort Algorithm and Bubble Sort Algorithm. Each algorithm is explained below.

SELECTION SORT ALGORITHM

Before we go to write algorithm for Selection Sort, we must know what is Selection Sort and how Selection Sort works. Selection Sort is a process of sorting numbers either in ascending order or in descending order. We can divide the process of Selection Sort in Passes. If there are N number in the Array then there will be N – 1 Passes to Sort the Array. In Pass First number Array is compared with rest of the numbers of Array one by one. And according to requirement first number of Array is replaced with one of rest numbers of Array. For example if we want to Sort Array in Ascending Order then the smallest number will be replaced with first number. And if we want to sort the Array in Descending Order then the largest number interchanges its position with the first number of Array. In Second Pass same process starts from the Second number. This process goes on until whole Array is not sorted. Let's have an example, suppose an Array contains 4 numbers and these are 12, 34, 10 and 36. We have to sort this Array in ascending order. Since this Array has 4 numbers therefore it will take only 3 Passes to sort this Array. Each Pass of sorting is discussed below.

Pass 1: In the First Pass 12 will be compared with 34 but 34 is greater than 12 so alteration will not occur. Again 12 will be compared with 10 and this time alteration will take place because 10 is less than 12. Then 10 will be compared with 36 since 36 is greater than 10 therefore no interchange will take place. Now the first place is sorted and the order is as.

10, 34, 12 and 36

Pass 2: As the first place is sorted so will start comparing from second place. Compare 34 with 12 since 12 is less than 36 so alteration is occurred. Now compare 12 with 36 since 36 is greater than 12 thus no

alteration will take place. Thus the Second place is sorted. Partially sorted Array is as.

10, 12, 34 and 36

Pass 3: Now 34 will is compared with 36 and here no alteration will be occurred because 34 is less than 36. Thus the Array if fully sorted and it is as.

10, 12, 34 and 36

Now the numbers are sorted in ascending order. Similarly the Array can be sorted in descending order.

(**Selection Sort Algorithm**) Let an Array DATA having N numeric value is given. This algorithm sorts the given numbers in ascending order. i and j are two counters.

Step 1: Start

Step 2: Read: Take N Inputs for Array DATA

Step 3: [Sorting Array in Ascending Order]

Repeat for i = 1 to N by 1

Repeat for j = i + 1 to N by 1

If DATA[i] > DATA[j] then:

[Swapping is Done Here]

Set: Temp = DATA[i]

Set: DATA[i] = DATA[j]

Set: DATA[j] = DATA[i]

[End of If Structure]

[End of inner for loop]

[End of outer for loop]

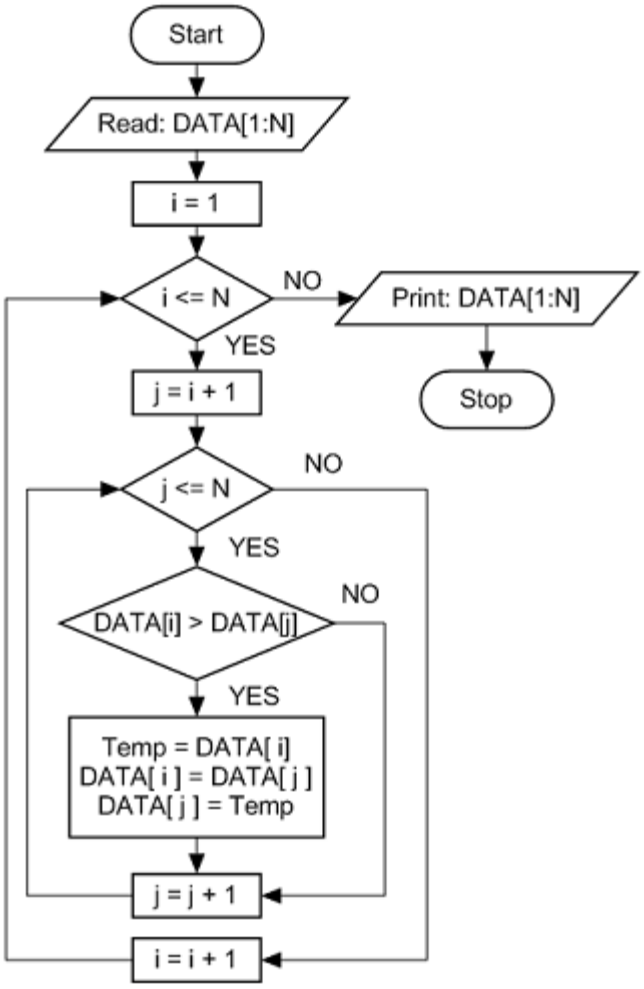
Step 4: Print: Sorted Array DATA[1:N]

Step 5: Exit

NOTE:

To sort the Array DATA in descending order just replace the statement with If DATA[i] > DATA[j] with Statement with DATA[i] < DATA[j] and the algorithm will sort the Array DATA in descending order.

FLOWCHART FOR SELECTION SORT



BUBBLE SORT ALGORITHM

Bubble Sort is another technique for sorting Number, Alphabets or Files either in ascending or descending order. Bubble Sort technique can also be divided into many Passes. The number of Passes will be N – 1 where N is the total Number of elements in the Array. Suppose we want to sort the Array in descending order then in first Pass, First number is compared with Second number. If First number is less than Second number then they interchange their positions. Then Second number is compared with Third number. If second number is greater than Third number then they will not interchange their positions. Then compare Third number with Fourth number. If Third number is less than Fourth number then they will interchange their positions. This Pass goes on until the last two numbers are not compared. At last the smallest number achieves the last position in the Array. In the Second Pass same process goes on up to position N – 1 that means in Second Pass last comparison is done between $(N - 2)^{th}$ and $(N - 1)^{th}$ numbers. In the Last Pass only one comparison is done. This comparison is done between First number and Second number. Now the Array is sorted in descending order. Let an Array list is as 32, 51, 85 and 13. We have to sort this Array in descending order. Since there are 4 numbers in the Array thus 3 Passes are required to sort this Array.

Pass One:

comparison 1: Compare 32 with 51, since 32 is less than 51 therefore 32 and 51 will change their positions. After this comparison Array is as: 51, 32, 85 and 13
comparison 2: Compare 32 and 85, since 32 is less than 85 therefore they will change their positions. Now Array is as follows: 51, 85, 32 and 13
comparison 3: Compare 32 and 13, since 32 is greater than 13 therefore no interchange will take place. Thus Array is as: 51, 85, 32 and 13

We can at the end of first Pass the smallest number is at last position and Array is 51, 85, 32 and 13.

Pass Two: Now the second Pass will start.

Comparison 1: Compare 51 with 85, since 51 is less than 85 therefore they will change their positions. Thus the Array is as follows: 85, 51, 32 and 13
Comparison 2: Compare 51 and 32, since 51 is greater than 32 therefore no interchange will take place. Thus the Array is as follows: 85, 51, 32 and 13

At the end of second Pass second last position is sorted. One more thing we must notice that the Array is completely sorted now but still third Pass will execute.

Pass Three:

comparison one: Compare 85 and 51, since 85 is greater than 51 therefore they will not interchange their positions. Thus the Array is as follows: 85, 51, 32 and 13

We must notice one thing that the number of comparisons in each is decreasing by 1. Now we will write the Algorithm for Bubble Sort.

(Bubble Sort Algorithm) Suppose an Array DATA have N Numeric values. This algorithm sorts the given Numbers in ascending order.

Step 1: Start

Step 2: Read: Take N Inputs for Array DATA

Step 3: Repeat for i = 1 to N by 1

Repeat for j := 1 to N – 1 by 1

If DATA[j] > DATA[j + 1] then:

Temp = DATA[j]

DATA[j] = DATA[j + 1]

DATA[j + 1] = Temp

[End of If structure]

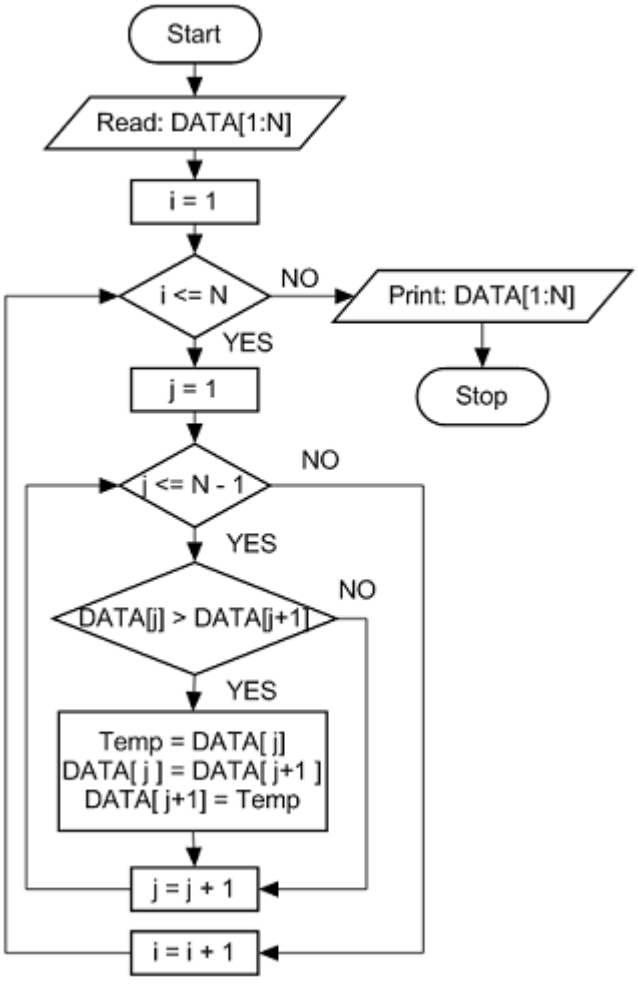
[End of inner for loop]

[End of outer for loop]

Step 4: Print: Array DATA[1:N]

Step 5: Exit

FLOWCHART FOR BUBBLE SORT



13.

ALGORITHM FOR MATRIX OPERATIONS

Their are different Matrix operations like Matrix addition, Matrix Subtraction, Matrix Multiplication, Inverse of a Matrix and Transpose of a Matrix. Here algorithm for matrix operations is explained one by one.

LIST OF ALGORITHM FOR MATRIX OPERATIONS

1. Algorithm for Matrix Addition
2. Algorithm for Matrix Subtraction
3. Algorithm for Matrix Multiplication
4. Algorithm for Matrix Transpose

ALGORITHM FOR MATRIX ADDITION

If A and B are two matrices of same order m x n (to be read as m by n matrix) then their addition is defined by A + B. For example if A is a matrix of order 2×3 and B is a matrix of same order 2×3 then addition is possible and the resultant matrix will be of order 2×3. If a matrix A has its order 2×3 and another matrix B has its order 3×2 then addition is not possible. So before adding any number of matrices we must check that whether each of the matrices are of same order or not then we need to proceed further.

ALGORITHM:

(**Matrix addition Algorithm**) Suppose A and B are two matrix arrays of order m x n, and C is another matrix array to store the addition result. i, j are counters.

Step1: Start

Step2: Read: m and n

Step3: Read: Take inputs for Matrix A[1:m, 1:n] and Matrix B[1:m, 1:n]

Step4: Repeat for i := 1 to m by 1:

Repeat for j := 1 to n by 1:

C[i, j] := A[i, j] + B[i, j]

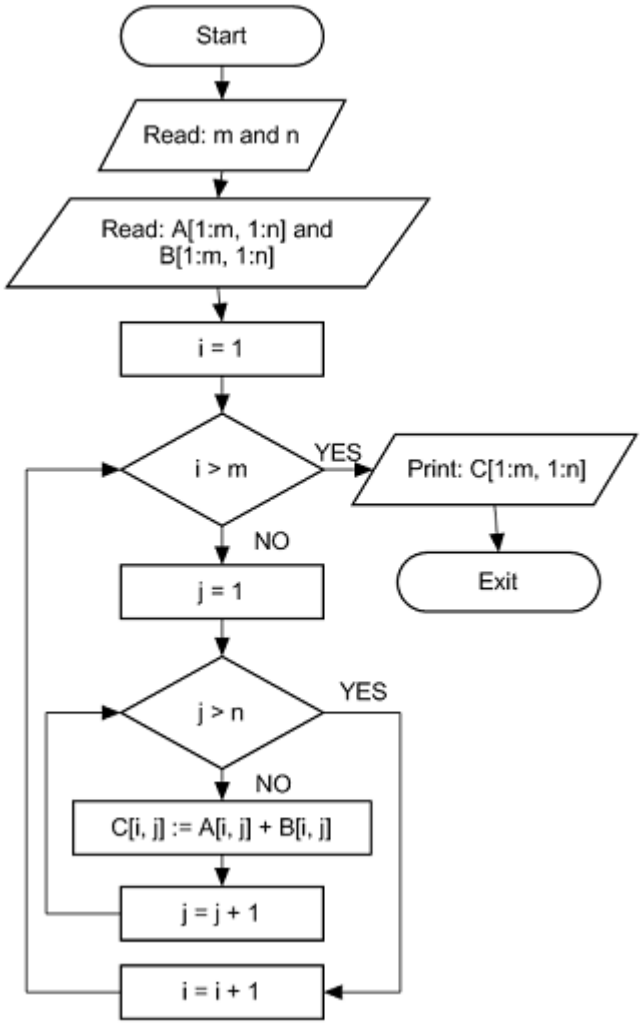
[End of inner for loop]

[End of outer for loop]

Step5: Print: Matrix C

Step6: Exit.

FLOWCHART FOR MATRIX ADDITION



ALGORITHM TO FIND MATRIX SUBTRACTION

If A and B are two matrices having same order m x n and C be the resultant Matrix then we can write C = A – B. The order of Matrix C will also be m x n. To perform subtraction, order of each matrix should be same otherwise subtraction is not possible as it stated above in the case of Matrix Addition.

ALGORITHM:

(Matrix Subtraction Algorithm) Suppose A and B are two matrix arrays of order m x n, and C is another matrix array to store the addition result. i, j are counters.

Step1: Start

Step2: Read: m and n

Step3: Read: Take inputs for Matrix A[1:m, 1:n] and Matrix B[1:m, 1:n]

Step4: Repeat for i := 1 to m by 1:

Repeat for j := 1 to n by 1:

C[i, j] := A[i, j] – B[i, j]

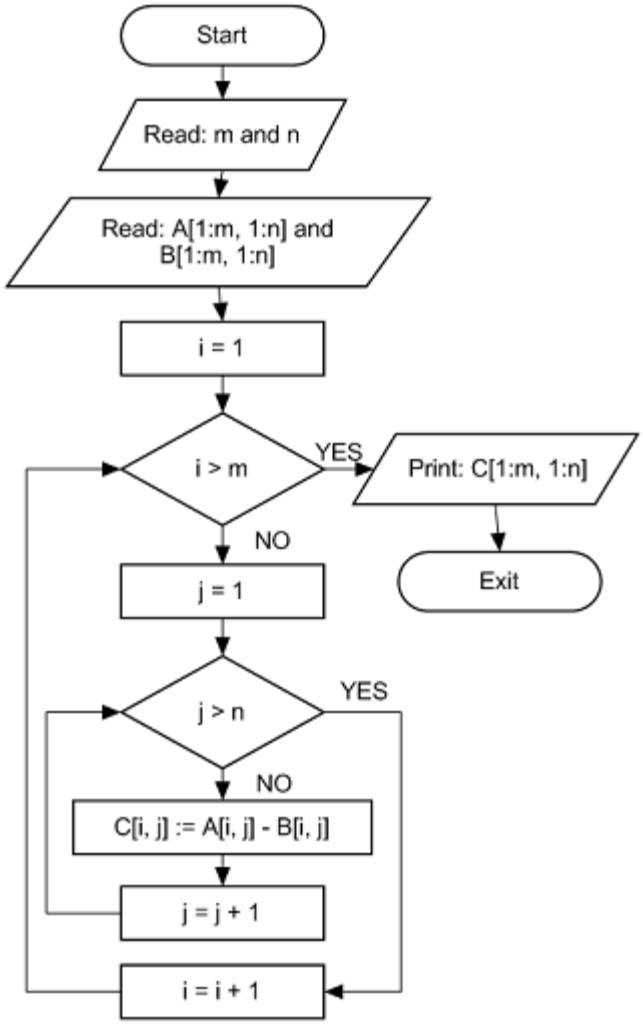
[End of inner for loop]

[End of outer for loop]

Step5: Print: Matrix C[1:m, 1:n]

Step6: Exit.

FLOWCHART FOR MATRIX SUBTRACTION:



ALGORITHM TO FIND MATRIX MULTIPLICATION

If A and B are two matrices of order m x n and p x q then their multiplication possible only if the number of columns in first matrix is equal to the number of rows in second matrix. Suppose C a matrix where the result of A x B will be stored then the order of C is m x p.

ALGORITHM

(Matrix Multiplication Algorithm) Suppose A and B are two matrices and their order are respectively m x n and p x q. i, j and k are counters. And C to store result.

Step1: Start.

Step2: Read: m, n, p and q

Step3: Read: Inputs for Matrices A[1:m, 1:n] and B[1:p, 1:q].

Step4: If n ≠ p then:

Print: Multiplication is not possible.

Else:

Repeat for i := 1 to m by 1:

Repeat for j := 1 to q by 1:

C[i, j] := 0 [Initializing]

Repeat k := 1 to n by 1

C[i, j] := C[i, j] + A[i, k] x B[k, j]

[End of for loop]

[End of for loop]

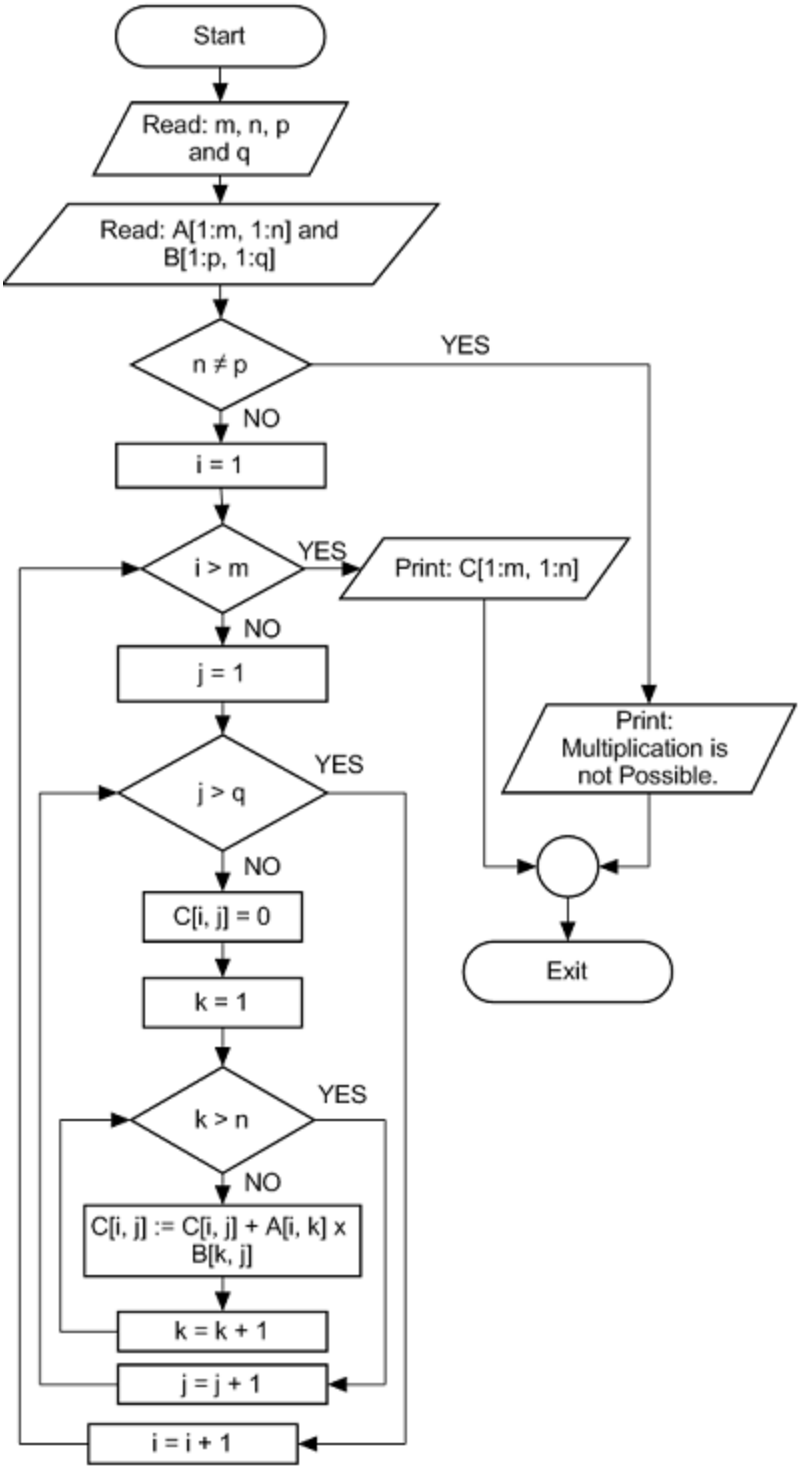
[End of for loop]

[End of If structure]

Step5: Print: C[1:m, 1:q]

Step6: Exit.

FLOWCHART FOR MATRIX MULTIPLICATION



ALGORITHM TO FIND TRANSPOSE OF A MATRIX

Suppose A is a matrix of order m x n then transpose of the matrix can be found by interchanging row of the matrix with columns of the matrix. The transpose of a matrix A is denoted by A' or A^T.

ALGORITHM:

(Matrix Transpose Algorithm) Suppose A is a matrix array of order m x n. And B is a matrix array of order n x m to store result. i and j are two counters.

Step1: Start.

Step2: Read: m and n

Step3: Read: Take inputs for Matrix A[1:m, 1:n].

Step4: If m == n then:

Repeat for i = 1 to m by 1

Repeat for j = 1 to n by 1

B[i, j] = A[j, i]

[End of for loop]

[End of for loop]

Else:

temp = m

m = n

n = temp

Repeat for i = 1 to m by 1

Repeat for j = 1 to n by 1

B[i, j] = A[j, i]

[End of for loop]

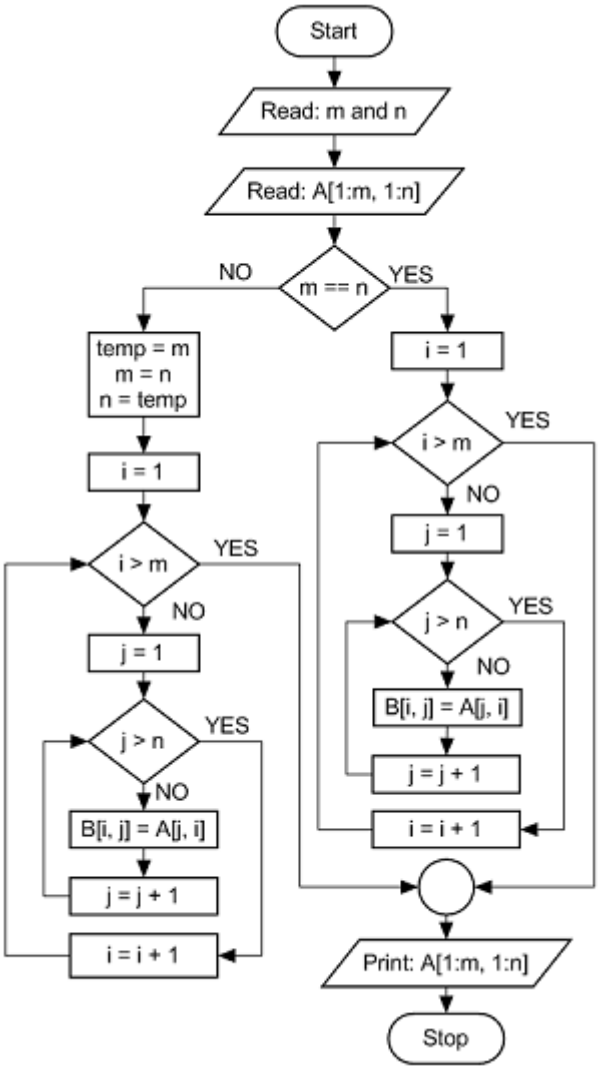
[End of for loop]

[End of If structure]

Step5: Print: B[1:m, 1:n]

Step6: Exit.

FLOWCHART TO FIND TRANSPOSE OF A MATRIX



15.

ALGORITHMS TO FIND MEAN, MODE AND MEDIAN

ALGORITHM TO FIND MEAN OR AVERAGE

(**Average Algorithm**) This algorithm is used to find the mean or Average of N numbers. Suppose DATA is an array variable and it has N numeric value. Formula to find Average or Mean of N Numbers is given below.

MEAN = (Sum of all N Numbers) / N

Algorithm for this problem is as:

Step1: Start

Step2: Read: DATA[1: N]

Step3: [Initialize] Sum := 0

Step4: Repeat for i := 1 to N by 1

Sum := DATA[i] + Sum

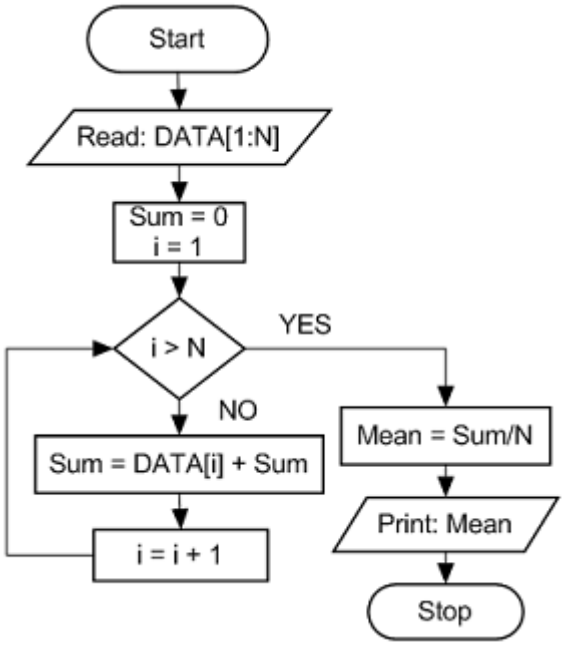
[End of for loop]

Step5: Mean := Sum / N

Step6: Print: 'Average of N number is Mean'

Step7: Stop

FLOWCHART TO FIND MEAN OR AVERAGE



ALGORITHM TO FIND MODE

(Mode Algorithm) Suppose an array having name DATA can have N numeric values is given. This algorithm will find the mode of the N numbers.

Step1: Start

Step2: [Initialize] Count := 0

Step3: Take N input for DATA array

Step4: Repeat for i := 1 to N by 1

[Initialize] Temp_count := 0

Repeat for j:= 1 to N by 1

If (DATA[i] == DATA[j]) then:

Temp_count := Temp_count + 1

[End of If structure]

[End of inner loop]

If (Temp_count>Count) then:

Count := Temp_count

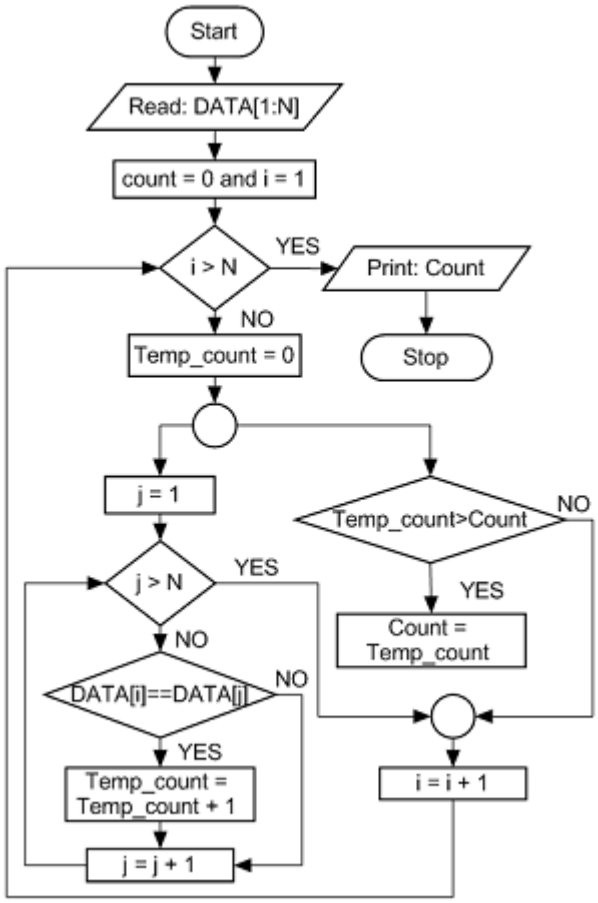
[End of if structure]

[End of outer loop]

Step5: Print: 'Mode of N numbers is Count'

Step6: Stop

FLOWCHART TO FIND MODE



ALGORITHM TO FIND MEDIAN

(Median Algorithm) Imagine an array DATA with N numeric values is given. Primarily the array is unsorted. This algorithm first sort the array in ascending order then it finds the median of N numbers. Depending upon the N there can be two possibilities: When N is odd then the median is (N + 1)/2. And when N is even then the median is average of the values of N/2 and N/2 + 1.

Step1: Start

Step2: Take N input for DATA array

Step3: Print: 'Sort the DATA array in ascending order'

Step4: Repeat for i := 1 to N by 1

Repeat for j := i + 1 to N by 1

If DATA[i] > DATA[j] then:

temp = DATA[i]

DATA[i] = DATA[j]

DATA[j] = temp

[End of If structure]

[End of inner loop]

[End of outer loop]

Step5: If N%2 ≠ 0

Median := (N + 1)/2

Print: 'Median is Median.'

Else

P := N/2

Q := N/2 + 1

Median := (DATA[P] + DATA[Q])/2

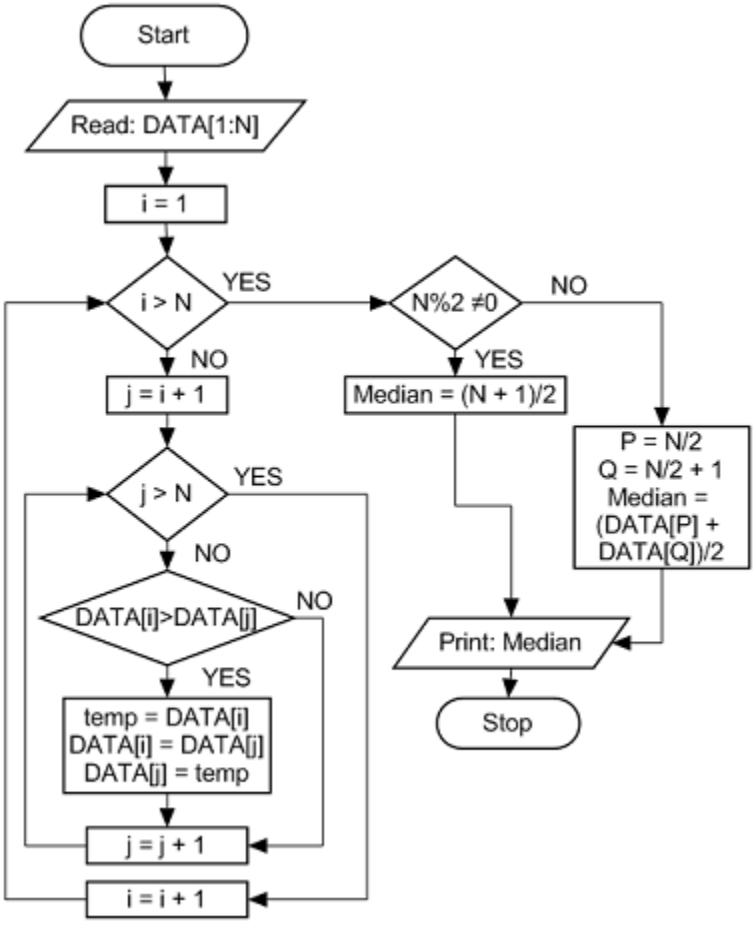
Print: ‘Median is Median.’

[End of If structure]

Step6: Stop

FLOWCHART TO FIND MEDIAN

When Numbers are not Sorted. To find Median of N numbers, first these numbers have to be sorted.



16.

SEARCH ALGORITHM

LINEAR SEARCH ALGORITHM

Suppose a linear Array DATA contains N elements and no other information is given about DATA. An element called ITEM is given. The most easiest way to search the ITEM in DATA is by comparing ITEM with each element of DATA. First Compare ITEM with DATA[1] if ITEM == DATA[1] then location of ITEM is 1 otherwise compare ITEM with DATA[2] and so on until last element is compared. This Method of searching the ITEM in the Array DATA is called ***Linear Search***. There can be many ways of writing algorithms for Linear Search. I have mentioned two ways over here.

1. Without Inserting ITEM Into Array DATA
2. By Inserting ITEM Into Array DATA

Without Inserting ITEM Into Array DATA:
(***Linear Search Algorithm***) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the ITEM and its location LOC in the array DATA. A Linear search algorithm solves the problem by comparing ITEM with each elements of Array one by one. In this algorithm Step 4 will continue till LOC is 0. As the value of LOC will change Step 4 will be break and Step 5 will start. If ITEM is found then its location will be printed otherwise a message "***ITEM is not fount***" will be printed.

Step 1: Start

Step 2: Read: Take Input for DATA[1:N] Array

Step 3: [Initializing] Set k = 1 and LOC = 0

Step 4: Repeat while LOC == 0 and k ≤ N

If ITEM == DATA[k], then:

Set LOC = k

[End of If Structure]

Set k = k + 1. [Incrementing counter]

[End of Step 4 loop]

Step 5: If LOC == 0 then:

Print: ITEM is not in the Array DATA.

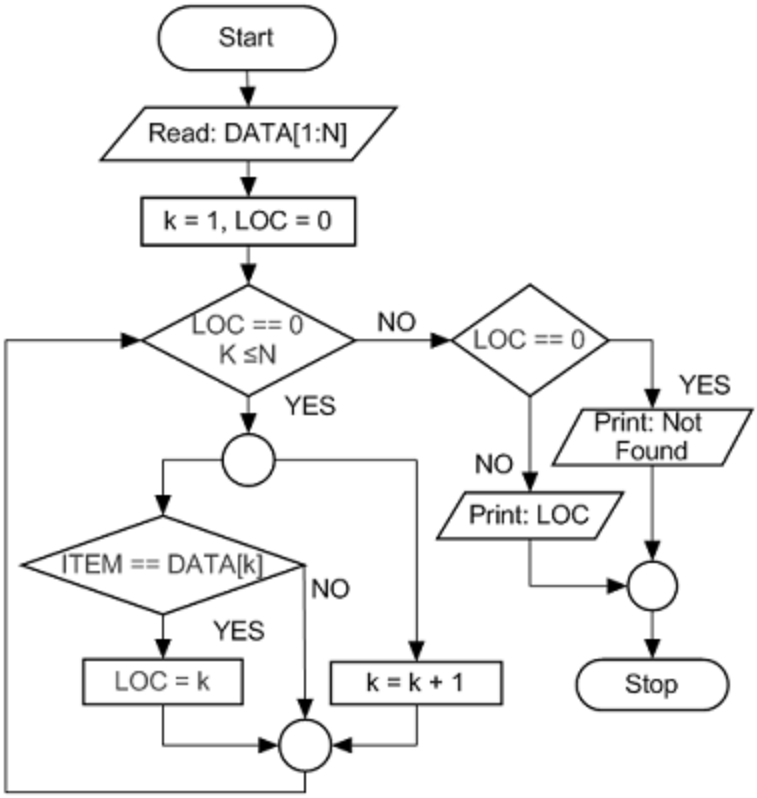
Else:

Print: LOC is the location of ITEM in the Array DATA.

[End of If-Else structure]

Step 6: Exit

FLOWCHART FOR LINEAR SEARCH BY METHOD ONE



By Inserting ITEM Into Array DATA

(*Linear Search Algorithm*) In this algorithm Array DATA and ITEM are same. Location of ITEM is also denoted by LOC. In this algorithm ITEM is assigned a position that is next to the last element of DATA. Or we can say ITEM is placed at potion N + 1. Where N is the Number of elements in Array DATA.

Step 1: Start

Step 2: Read: Take Input of Array DATA[1:N]

Step 3: [initializing LOC] LOC = 1

Step 4: [Assigning ITEM a Position] Set DATA[N + 1] = ITEM

Step 5: Repeat While loop DATA[LOC] ≠ ITEM [Searching ITEM in Array]

Set LOC = LOC + 1

[End of While loop]

Step 6: If LOC == N + 1 then:

Print: ITEM is not in the Array DATA.

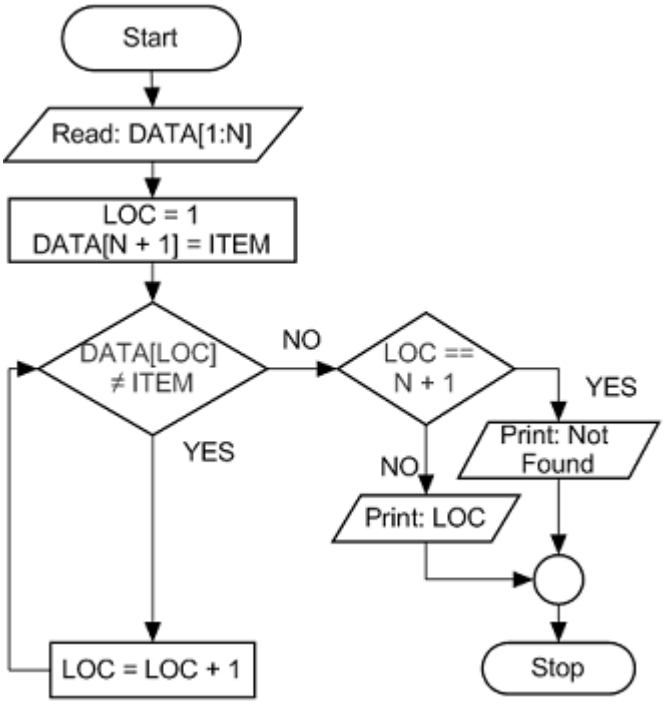
Else

Print: LOC is the location of ITEM in the Array DATA.

[End if If-Else Structure]

Step 7: Exit

FLOWCHART FOR LINEAR SEARCH METHOD TWO



BINARY SEARCH ALGORITHM

Binary search method is another way of searching a number in an Array. This method is better than Linear Seach method. Because the number of comparision in Binary Search is less than that in Linear Search. Binary Search method works when Array is sorted otherwise Array has to be sorted before Binary Search is applied. This technique divides the Array from middle into two Parts. Then compare the *Number* which is to be

found with the *Middle Number*. If *Middle Number* is the *Number* then searched is completed. If *Number* is less than the *Middle Number* then the second half is discarded and now search is performed on the first half of Array. Again the first half is divided from middle and the *Number* is compared with the *Middle Number* of First Half. If the number is greater than the *Middle Number* then the same process is repeated as it was done before until number is not found. If *Number* is not found then a message is printed that "*Number is not found"*.

(*Binary Search Algorithm*) BINARY SEARCH SORTED(DATA, LB, UB, ITEM, LOC)

Suppose a sorted array DATA having N numeric value is given. LB and UB are *Lower Bound* and *Upper Bound* of the array. BEG, END and MID represent *Beginning, End* and *Middle* locations respectively. This algorithm finds the location denoted by LOC of an ITEM in the Array DATA.

Step 1: Start

Step 2: Take Input for Array DATA[N:1]

Step 3: [Initializing Lower Bound, Upper Bound and LOC]

Set: LB = DATA[BEG], UB = DATA[END] and LOC = 0

Step 4: Repeat while LB <= UB and LOC = 0

[Computing Middle Location] MID = INT((BEG + END)/2)

If ITEM < DATA[MID] then:

Set: UB = MID – 1

[End of If Structure]

If ITEM > DATA[MID] then:

Set: LB = DATA[MID]

[End of If Structure]

If ITEM == DATA[MID] then:

Set: LOC = MID

[End of If Structure]

[End of while loop]

Step 5: If LOC == 0 then:

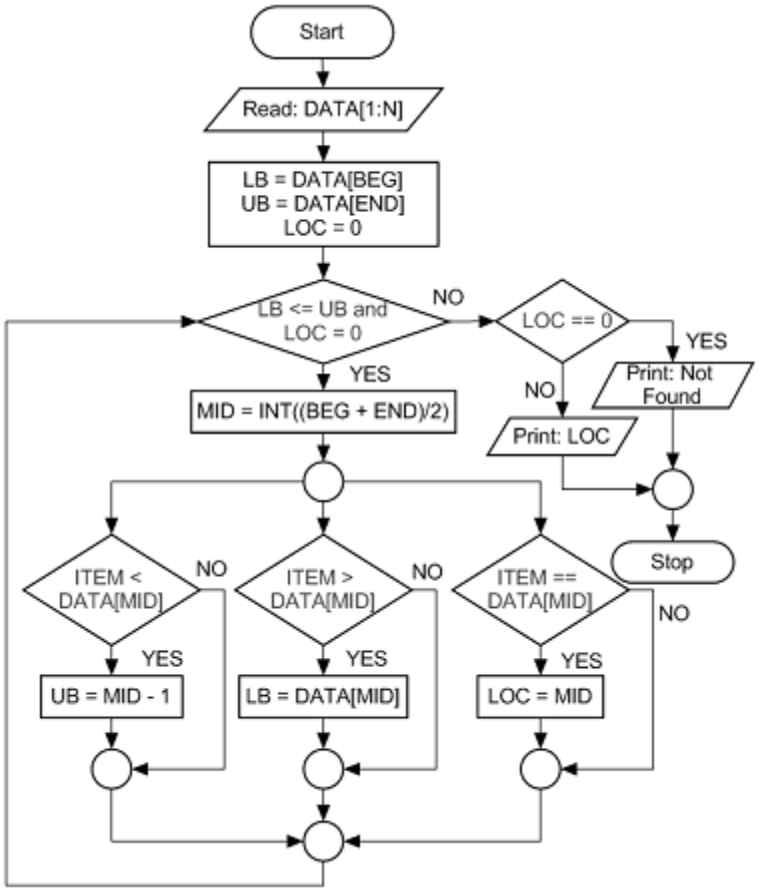
Print: ITEM not found in Array DATA

Else

Print: Location of the ITEM is LOC

Step 6: Exit

FLOWCHART FOR BINARY SEARCH



(*Binary Search Algorithm*) BINARY SEARCH UNSORTED(DATA, LB, UB, ITEM, LOC)

Let DATA is an unsorted Array having N numbers. We will apply Selection Sort method to sort the Array DATA.

Step 1: Start

Step 2: Take N Inputs for Array DATA

Step 3: [Initialize] Temp = 0

Step 4: [Sorting Array DATA in Ascending Order]

Repeat for i = 1 to N by 1

Repeat for j = i + 1 to N by 1

If DATA[i] > DATA[j] then:

[Interchanging Locations of DATA[i] and DATA[j]]

Set: Temp = DATA[i]

Set: DATA[j] = Temp

Set: DATA[i] = DATA[j]

[End of If structure]

[End of inner for loop]

[End of outer for loop]

Step 5: LOC = CALL BINARY SEARCH(DATA, ITEM)

Step 6: If LOC == 0 then:

Print: ITEM not found in Array DATA

Else

Print: Location of the ITEM is LOC

Step 7: Exit

BINARY SEARCH(DATA, ITEM)

BEG, END, MID, LB and UB represent *Beginning, End*, *Middle*, *Lower Bound* and *Upper Bound* locations respectively.

Step 1: [Initializing LB, UB and POSITION]

Set: LB = DATA[BEG], UB = DATA[END] and POSITION = 0

Step 2: Repeat while LB <= UB and POSITION == 0

[Computing Middle Location] MID = INT((BEG + END)/2)

If ITEM < DATA[MID] then:

Set: UB = MID – 1

[End of If Structure]

If ITEM > DATA[MID] then:

Set: LB = DATA[MID]

[End of If Structure]

If ITEM == DATA[MID] then:

Set: POSITION = MID

[End of If Structure]

[End of while loop]

Step 3: Return POSITION