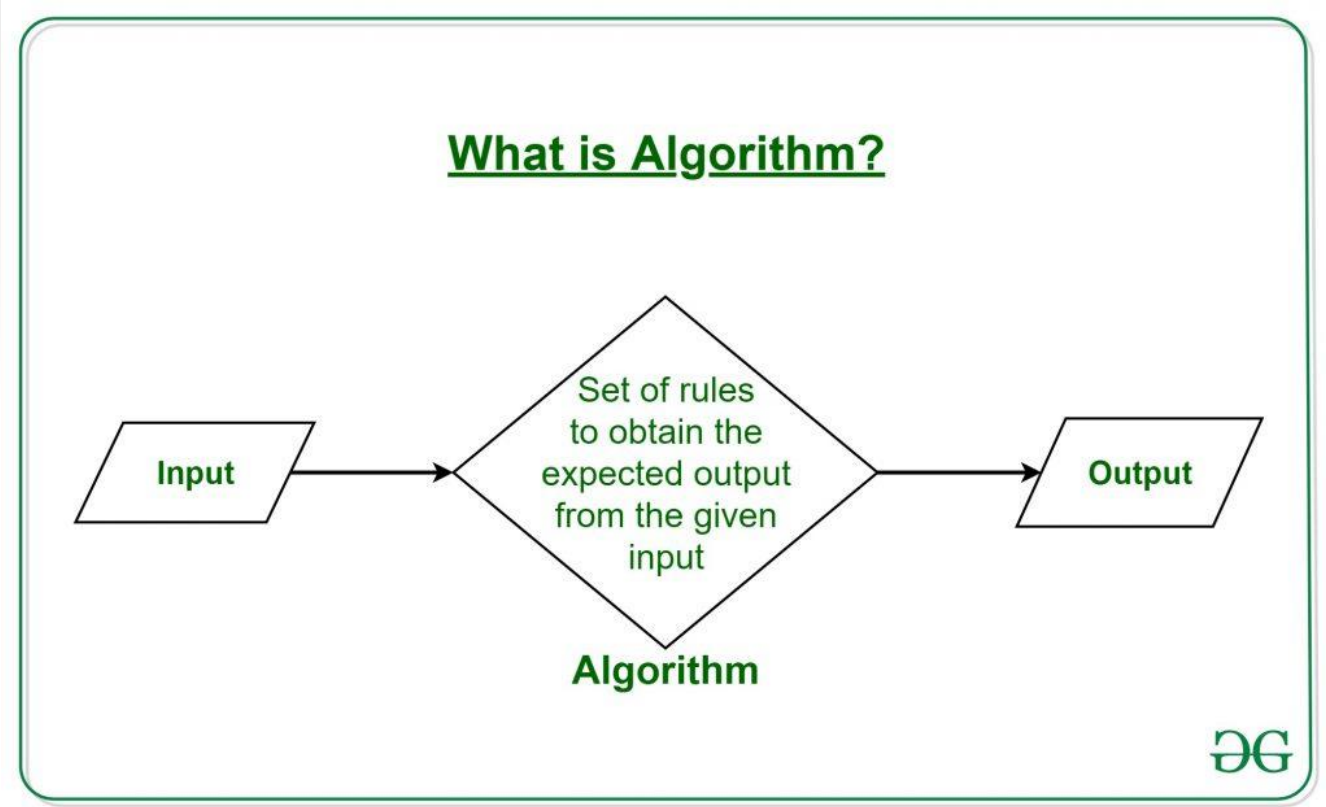
Discrete Mathematics 7
Algorithm

Dr. Maged Kassab

What is an Algorithm?

An algorithm is a process or a set of rules required to perform calculations or some other problem-solving operations especially by a computer. The formal definition of an algorithm is that it contains the finite set of instructions which are being carried in a specific order to perform the specific task. It is not the complete program or code; it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudocode.

What is an Algorithm?



Characteristics of an Algorithm

The following are the characteristics of an algorithm:

Input: An algorithm has some input values. We can pass 0 or some input value to an algorithm.

Output: We will get 1 or more output at the end of an algorithm.

Unambiguity: An algorithm should be unambiguous which means that the instructions in an algorithm should be clear and simple.

Finiteness: An algorithm should have finiteness. Here, finiteness means that the algorithm should contain a limited number of instructions, i.e., the instructions should be countable.

Characteristics of an Algorithm

Finiteness: An algorithm should have finiteness. Here, finiteness means that the algorithm should contain a limited number of instructions, i.e., the instructions should be countable.

Effectiveness: An algorithm should be effective as each instruction in an algorithm affects the overall process.

Language independent: An algorithm must be language-independent so that the instructions in an algorithm can be implemented in any of the languages with the same output.

Characteristics of an Algorithm

Finiteness: An algorithm should have finiteness. Here, finiteness means that the algorithm should contain a limited number of instructions, i.e., the instructions should be countable.

Effectiveness: An algorithm should be effective as each instruction in an algorithm affects the overall process.

Language independent: An algorithm must be language-independent so that the instructions in an algorithm can be implemented in any of the languages with the same output.

Dataflow of an Algorithm

- **Problem:** A problem can be a real-world problem or any instance from the real-world problem for which we need to create a program or the set of instructions. The set of instructions is known as an algorithm.
- **Algorithm:** An algorithm will be designed for a problem which is a step by step procedure.
- **Input:** After designing an algorithm, the required and the desired inputs are provided to the algorithm.
- **Processing unit:** The input will be given to the processing unit, and the processing unit will produce the desired output.
- **Output:** The output is the outcome or the result of the program.

Why do we need Algorithms?

We need algorithms because of the following reasons:

•**Scalability:** It helps us to understand the scalability. When we have a big real-world problem, we need to scale it down into small-small steps to easily analyze the problem.

•**Performance:** The real-world is not easily broken down into smaller steps. If the problem can be easily broken into smaller steps means that the problem is feasible.

Let's understand the algorithm through a real-world example. Suppose we want to make a lemon juice, so following are the steps required to make a lemon juice:

Why do we need Algorithms?

Step 1: First, we will cut the lemon into half.

Step 2: Squeeze the lemon as much you can and take out its juice in a container.

Step 3: Add two tablespoon sugar in it.

Step 4: Stir the container until the sugar gets dissolved.

Step 5: When sugar gets dissolved, add some water and ice in it.

Step 6: Store the juice in a fridge for 5 to minutes.

Step 7: Now, it's ready to drink.

Why do we need Algorithms?

The above real-world can be directly compared to the definition of the algorithm. We cannot perform the step 3 before the step 2, we need to follow the specific order to make lemon juice. An algorithm also says that each and every instruction should be followed in a specific order to perform a specific task.

Now we will look an example of an algorithm in programming. We will write an algorithm to add two numbers entered by the user.

Why do we need Algorithms?

The following are the steps required to add two numbers entered by the user:

Step 1: Start

Step 2: Declare three variables a, b, and sum.

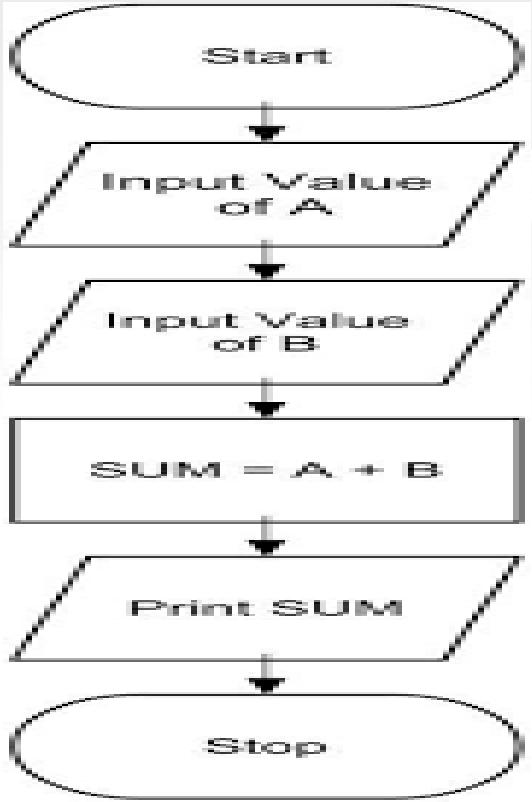
Step 3: Enter the values of a and b.

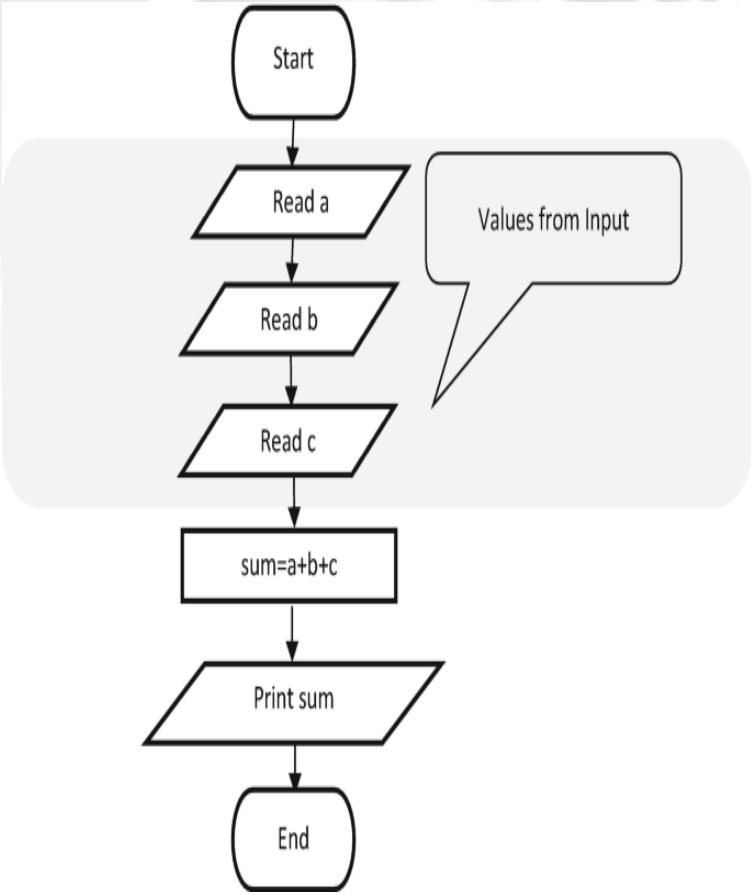
Step 4: Add the values of a and b and store the result in the sum variable, i.e., sum=a+b.

Step 5: Print sum

Step 6: Stop

Why do we need Algorithms?





Factors of an Algorithm

- **Modularity:** If any problem is given and we can break that problem into small-small modules or small-small steps, which is a basic definition of an algorithm, it means that this feature has been perfectly designed for the algorithm.
- **Correctness:** The correctness of an algorithm is defined as when the given inputs produce the desired output, which means that the algorithm has been designed algorithm. The analysis of an algorithm has been done correctly.
- **Maintainability:** Here, maintainability means that the algorithm should be designed in a very simple structured way so that when we redefine the algorithm, no major change will be done in the algorithm.

Factors of an Algorithm

- **Functionality:** It considers various logical steps to solve the real-world problem.
- **Robustness:** Robustness means that how an algorithm can clearly define our problem.
- **User-friendly:** If the algorithm is not user-friendly, then the designer will not be able to explain it to the programmer.
- **Simplicity:** If the algorithm is simple then it is easy to understand.
- **Extensibility:** If any other algorithm designer or programmer wants to use your algorithm then it should be extensible.

Importance of Algorithms

1.Theoretical importance: When any real-world problem is given to us and we break the problem into small-small modules. To break down the problem, we should know all the theoretical aspects.

2.Practical importance: As we know that theory cannot be completed without the practical implementation. So, the importance of algorithm can be considered as both theoretical and practical.

Algorithm Analysis

The algorithm can be analyzed in two levels, i.e., first is before creating the algorithm, and second is after creating the algorithm. The following are the two analysis of an algorithm:

- **Priori Analysis:** Here, priori analysis is the theoretical analysis of an algorithm which is done before implementing the algorithm. Various factors can be considered before implementing the algorithm like processor speed, which has no effect on the implementation part.
- **Posterior Analysis:** Here, posterior analysis is a practical analysis of an algorithm. The practical analysis is achieved by implementing the algorithm using any programming language. This analysis basically evaluate that how much running time and space taken by the algorithm.

Algorithm Complexity

The performance of the algorithm can be measured in two factors:

- **Time complexity:** The time complexity of an algorithm is the amount of time required to complete the execution.

Space complexity: An algorithm's space complexity is the amount of space required to solve a problem and produce an output.

What are different types of algorithms?

- **Search engine algorithm.** This algorithm takes search strings of keywords and operators as input, searches its associated database for relevant webpages and returns results.
- **Encryption algorithm.** This computing algorithm transforms data according to specified actions to protect it. A symmetric key algorithm, such as the Data Encryption Standard, for example, uses the same key to encrypt and decrypt data.

Greedy algorithm. This algorithm solves optimization problems by finding the locally optimal solution, hoping it is the optimal solution at the global level. However, it does not guarantee the most optimal solution.

What are different types of algorithms?

Recursive algorithm. This algorithm calls itself repeatedly until it solves a problem. Recursive algorithms call themselves with a smaller value every time a recursive function is invoked.

Backtracking algorithm. This algorithm finds a solution to a given problem in incremental approaches and solves it one piece at a time.

Divide-and-conquer algorithm. This common algorithm is divided into two parts. One part divides a problem into smaller sub-problems. The second part solves these problems and then combines them together to produce a solution.

What are different types of algorithms?

Dynamic programming algorithm. This algorithm solves problems by dividing them into sub-problems. The results are then stored to be applied for future corresponding problems.

Brute-force algorithm. This algorithm iterates all possible solutions to a problem blindly, searching for one or more solutions to a function.

Sorting algorithm. Sorting algorithms are used to rearrange data structure based on a comparison operator, which is used to decide a new order for data.

What are different types of algorithms?

- **Hashing algorithm.** This algorithm takes data and converts it into a uniform message with a hashing
- **Randomized algorithm.** This algorithm reduces running times and time-based complexities. It uses random elements as part of its logic.

Advantages and disadvantages of Algorithms:

Advantages of Algorithms:

•It is easy to understand.

•An algorithm is a step-wise representation of a solution to a given problem.

•In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

Disadvantages of Algorithms:

•Writing an algorithm takes a long time so it is time-consuming.

•Understanding complex logic through algorithms can be very difficult.

•Branching and Looping statements are difficult to show in Algorithms

How to Design an Algorithm?

In order to write an algorithm, the following things are needed as a pre-requisite:

1. The **problem** that is to be solved by this algorithm i.e. clear problem definition.
2. The **constraints** of the problem must be considered while solving the problem.
3. The **input** to be taken to solve the problem.
4. The **output** to be expected when the problem is solved.
5. The **solution** to this problem, is within the given constraints.

Then the algorithm is written with the help of the above parameters such that it solves the problem.

Example: Consider the example to add three numbers and print the sum.

How to Design an Algorithm?

•**Step 1: Fulfilling the pre-requisites**

As discussed above, in order to write an algorithm, its pre-requisites must be fulfilled.

- **The problem that is to be solved by this algorithm**: Add 3 numbers and print their sum.
- **The constraints of the problem that must be considered while solving the problem**: The numbers must contain only digits and no other characters.
- **The input to be taken to solve the problem:** The three numbers to be added.
- **The output to be expected when the problem is solved:** The sum of the three numbers taken as the input i.e. a single integer value.
- **The solution to this problem, in the given constraints:** The solution consists of adding the 3 numbers. It can be done with the help of '+' operator, or bit-wise, or any other method.

How to Design an Algorithm?

- **Step 2: Designing the algorithm**

Now let's design the algorithm with the help of the above pre-requisites:

Algorithm to add 3 numbers and print their sum:

- START
- Declare 3 integer variables num1, num2 and num3.
- Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
- Declare an integer variable sum to store the resultant sum of the 3 numbers.
- Add the 3 numbers and store the result in the variable sum.
- Print the value of the variable sum
- END

- **Step 3: Testing the algorithm by implementing it.**

How to express an Algorithm?

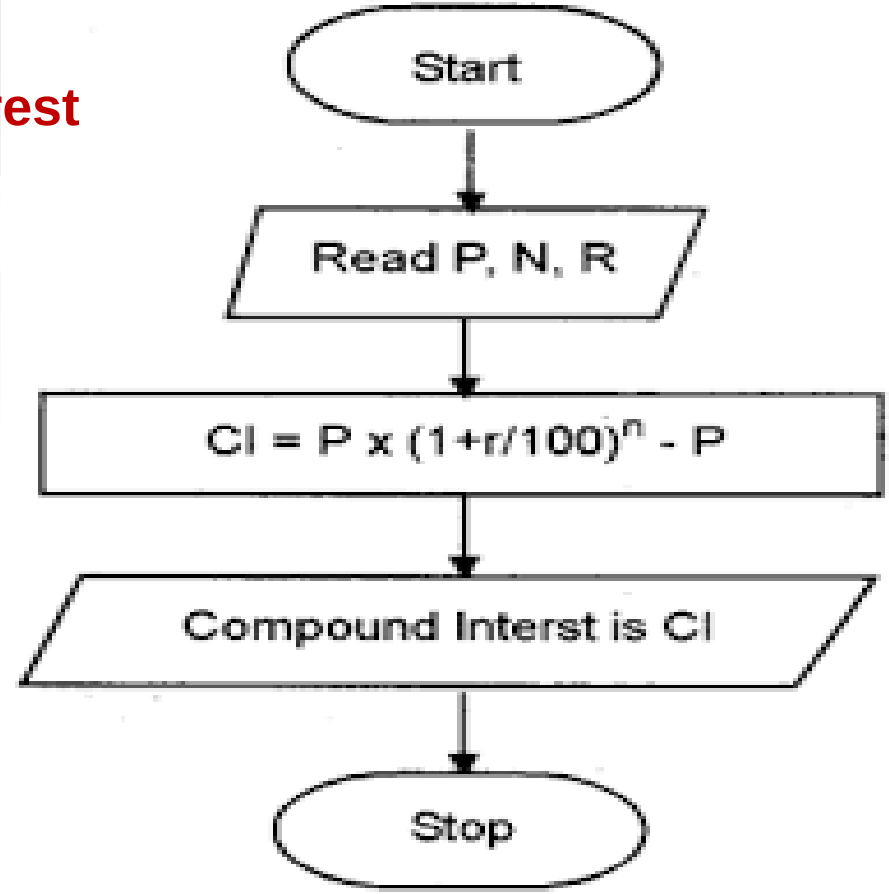
1.Natural Language :- Here we express the Algorithm in natural English language. It is too hard to understand the algorithm from it.

2.Flow Chat :- Here we express the Algorithm by making graphical/pictorial representation of it. It is easier to understand than Natural Language.

3.Pseudo Code :- Here we express the Algorithm in the form of annotations and informative text written in plain English which is very much similar to the real code but as it has no syntax like any of the programming language, it can't be compiled or interpreted by the computer. It is the best way to express an algorithm because it can be understood by even a layman with some school level programming knowledge.

Algorithms: Different Examples

Calculation of compound interest



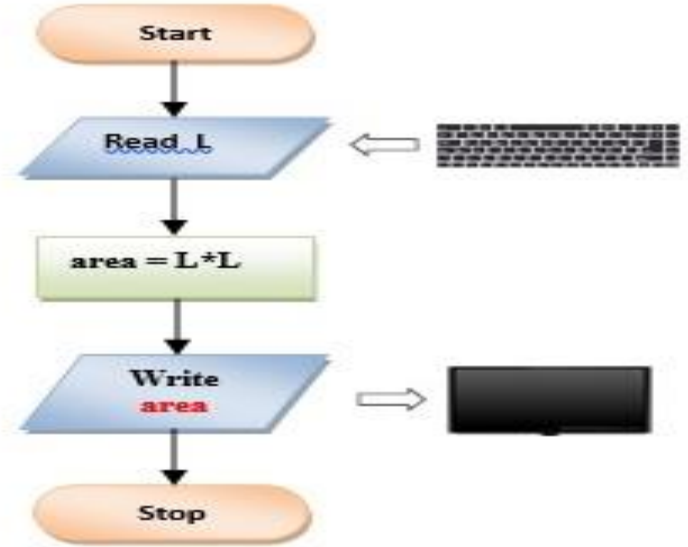
Algorithms: Different Examples

Finding Area of the square

Algorithm

1. Start
2. Read length, L
3. area = L*L
4. Print or display area
5. Stop

Flowchart



Algorithms: Different Examples

Finding Area of the rectangle

Algorithm

1. Start
2. Read side length, a
3. Read side length b
4. area = a*b
5. Print or display area
6. Stop

Flowchart

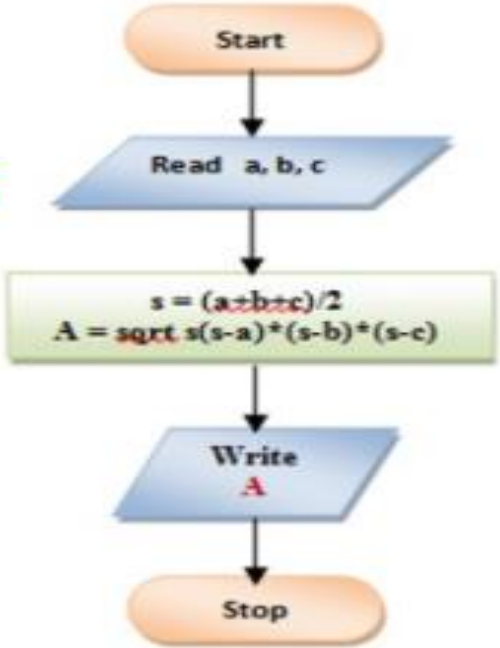
Algorithms: Different Examples

Area of a triangle where three sides are given

Algorithm

1. Start
2. Read a, b, c
3. s = (a+b+c)/2
4. A=sqrt (s *(s-a)*(s-b)*(s-c))
5. Print or display A
6. Stop

Flowchart



Algorithms: Different Examples

Find the area & perimeter of a square

Algorithm

1. Start
2. Read length L
3. Area A = L*L
4. Perimeter P = 4*L
5. Print or display A,P
6. Stop

Flowchart

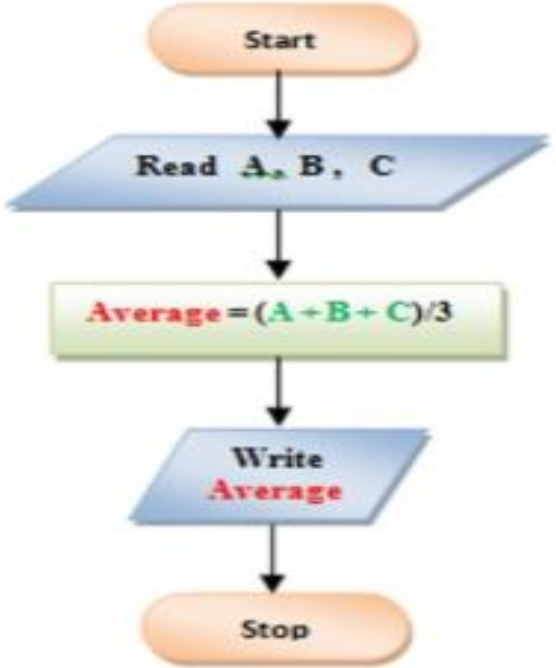
Algorithms: Different Examples

Calculating the average for 3 numbers

Algorithm

1. Start
2. Read 3 numbers A, B, C
3. Calculate the average by the equation:
   Average = (A + B + C)/3
4. Print average
5. Stop

Flowchart



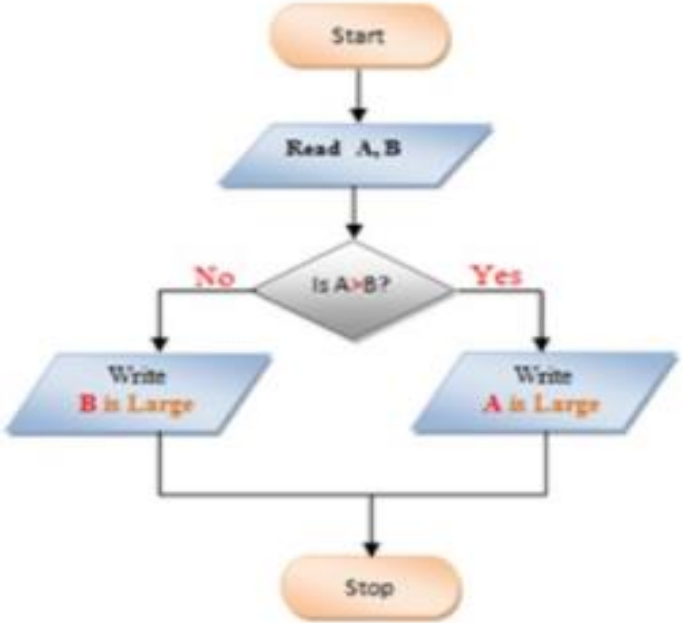
Algorithms: Different Examples

Greatest of two numbers

Algorithm

1. Start
2. Read A,B
3. If A > B then
   Print A is large
   else
   Print B is large
4. Stop

Flowchart



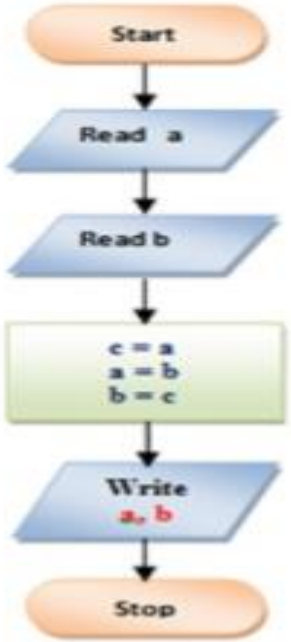
Algorithms: Different Examples

Interchange the value of two variables

Algorithm

1. Start
2. Read two values into two variables a, b
3. Declare third variable, c
   - c = a
   - a = b
   - b = c
4. Print or display a, b
5. Stop

Flowchart



Algorithms: Different Examples

Convert temperature from Fahrenheit to Celsius

Algorithm

1. Start
2. Initialize F = 0, C = 0
3. Read F
4. C = (F-32) * 5/9
5. Write C
6. Stop

Flowchart

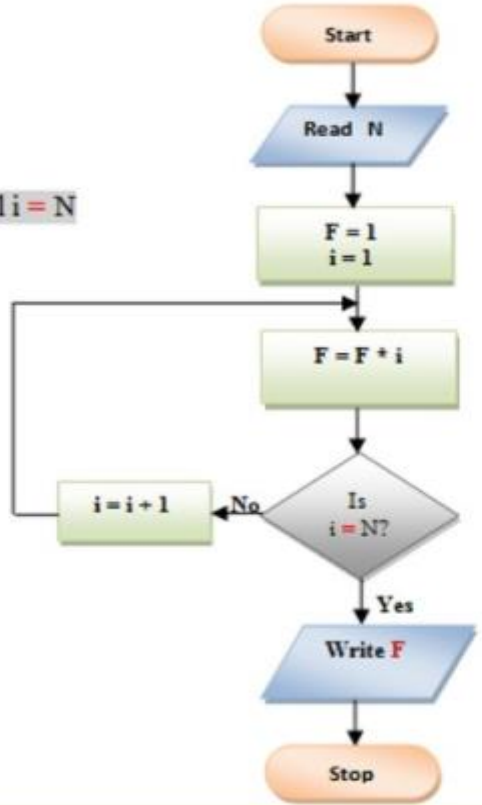
Algorithms: Different Examples

Draw a flowchart for computing factorial N, where N! = 1 * 2 * 3 * ..... N

Algorithm

1. Start
2. Read N
3. Initialize F =1, i = 1
4. F = F * i;
5. Increment i by 1
6. Repeat step 4 & 5 until i = N
7. Print F
8. Stop

Flowchart



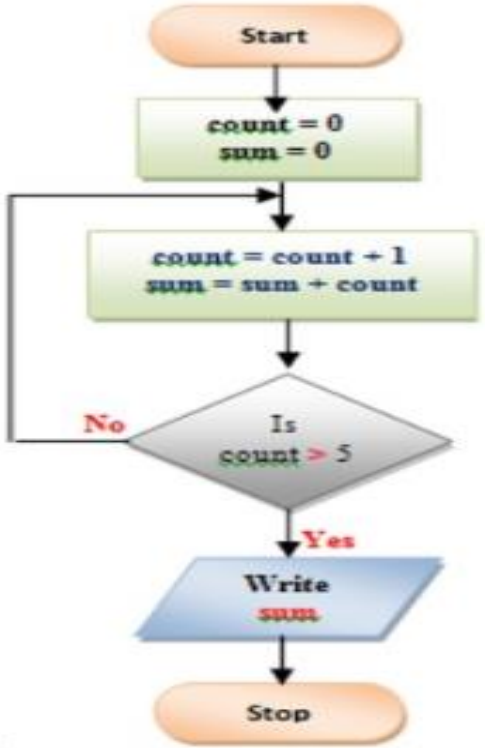
Algorithms: Different Examples

Find the Sum of First 5 Natural Numbers

Algorithm

1. Start
2. Initialize count = 0, sum = 0
3. count = count + 1
4. sum = sum + count
5. Repeat steps 3,4 until count > 5
6. Print sum
7. Stop

Flowchart



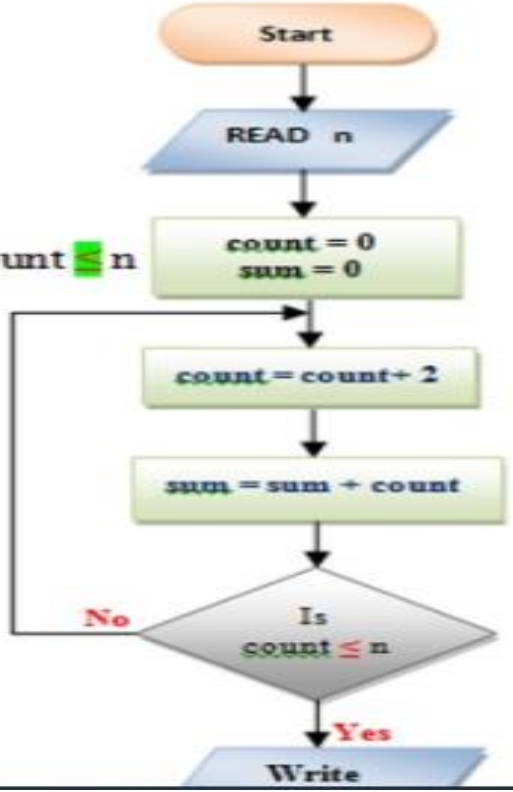
Algorithms: Different Examples

To find the sum of all even numbers up to 'n'

Algorithm

1. Start
2. Read n
3. count=0
4. sum=0
5. count = count + 2
6. sum = sum + count
7. Repeat steps 5 & 6 until count ≤ n
8. Print sum
9. Stop

Flowchart



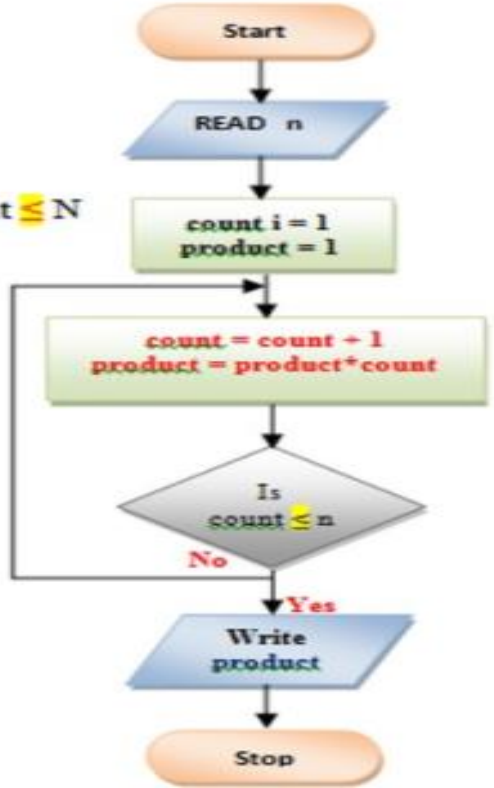
Algorithms: Different Examples

To find Product of numbers upto N

Algorithm

1. Start
2. Read n
3. count i = 1
4. product = 1
5. product=count*product
6. count = count + 1
7. Repeat steps 5,6 until count ≤ N
8. Print product
9. Stop

Flowchart



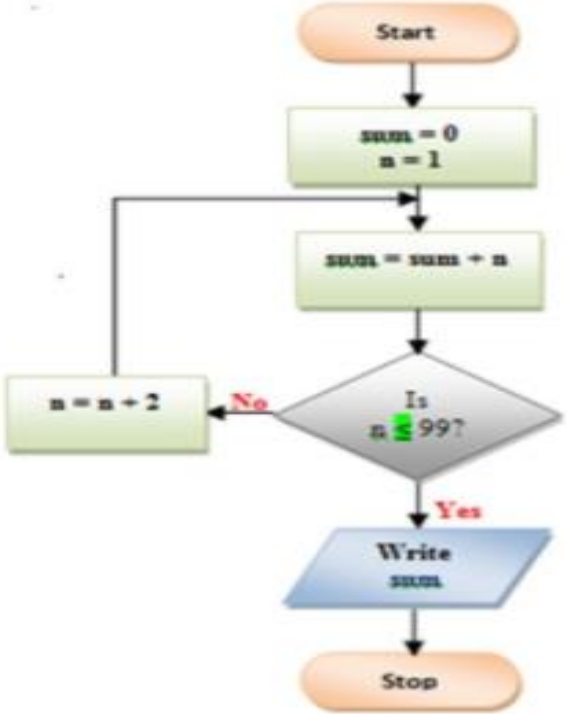
Algorithms: Different Examples

Sum of first 50 odd numbers

Algorithm

1. Start
2. sum=0, n = 1
3. sum=sum + n
4. n = n + 2
5. Repeat steps 4 and 5 until n ≤ 99
6. Print sum
7. Stop

Flowchart



شكراً لحسن المتابعة