Introduction to Algorithms:

Algorithm:

An algorithm is a finite set of well-defined instructions that takes some set of values as *input* and produces some set of values as *output* to solve a problem. An algorithm is thus a sequence of computational steps that transform the input into the output.

Or

A step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.

Key properties of Algorithms include:

Basic Criteria of an Algorithm:

Input: Algorithms take one or more inputs, which are the initial data or values upon which the algorithm operates to produce an output.

Output: Algorithms generate an output based on the provided input and the sequence of steps prescribed by the algorithm. The output represents the solution or result of the problem the algorithm addresses.

Deterministic: Each step in the algorithm must be clear and deterministic, meaning that given the same input, the algorithm will always produce the same output and follow the same set of steps.

Finiteness: An algorithm must have a finite number of steps. It should eventually terminate after executing a finite number of operations, producing the desired result.

Effectiveness: Algorithms must be effective, meaning they must be capable of being carried out in practice using a reasonable amount of time and resources. The steps of the algorithm should be feasible and within the limits of computational capabilities.

Distinct Area of an Algorithm:

1. Design of an Algorithm: This part is used to design a new algorithm for the specific given problem. The different algorithms may be designed for a single problem, but during the design of an algorithm, we must have to follow the basic criteria.

2. Validate of an Algorithm: Validate an algorithm means we have to check that the designed algorithm produces the correct output or not for all legal sets of inputs.

3. Analysis of an Algorithm: In the analysis part we calculate that if for a single problem, there is more than one algorithm designed then we have to analysis that which algorithm takes minimum time and minimum space to produce the output.

4. Testing: It is the process to make a program or algorithm error-free.

Algorithm design tools – flowcharts and pseudo code, notations

What is a Flowchart?

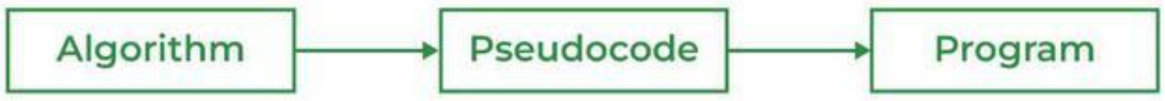
- A flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem.

- It makes use of symbols that are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”.

- Example: Draw a flowchart to input two numbers from the user and display the largest of two numbers

What is PseudoCode?

- A Pseudocode is defined as a step-by-step description of an algorithm. Pseudocode does not use any programming language in its representation instead it uses the simple English language text as it is intended for human understanding rather than machine reading.



(Pseudocode is an intermediate state between algorithm and program)

What is Notation?

- Notation is a way to describe the running time or space complexity of an algorithm based on the input size. It is commonly used in complexity analysis to describe how an algorithm performs as the size of the input grows. The three most commonly used notations are Big O, Omega, and Theta.

Algorithm header, purpose, conditions and selection, loops, procedures and sub-algorithms

Header:

- In algorithm, a header typically refers to the beginning part of an algorithm file where important information about the file is provided. This can include things like author information, file descriptions, and external dependencies.

Purpose:

- The purpose of an algorithm is the reason it was created or designed. It defines what problem the algorithm is meant to solve or what task it is supposed to accomplish.

Conditions and Selection:

- Conditions and selection refer to the ability of a program to make decisions based on certain conditions. This is typically achieved through conditional statements such as **if statements, switch statements, or ternary operators**. These statements allow the program to choose different paths of execution based on the evaluation of certain conditions.

Loops:

- Loops are programming constructs that allow you to repeat a block of code multiple times. There are different types of loops, including:

    - **For loop:** Executes a block of code a fixed number of times.
    - **While loop:** Executes a block of code as long as a specified condition is true.
    - **Do-While loop:** Similar to a while loop, but it always executes the block of code at least once, even if the condition is initially false.

Procedures and Sub-algorithms:

- Procedures (also known as functions or methods) and sub-algorithms are reusable blocks of code that perform a specific task.

- They allow you to break down your program into smaller, more manageable pieces, making your code more modular, easier to understand, and easier to maintain.

- Procedures can take input parameters and return output values, while sub-algorithms are smaller algorithms that are typically part of a larger algorithm.

Algorithm: SumEvenNumbers

Header:
Title: SumEvenNumbers
Author: [Your Name]
Date: [Current Date]

Purpose:
This algorithm calculates the sum of all even numbers up to a given positive integer n.

Conditions and Selection:
Loop through numbers from 1 to n.
If a number is even, add it to the sum.

Loops:
For loop to iterate through numbers from 1 to n.

Procedures and Sub-algorithms:
None needed for this simple algorithm.

Example:
Input: n = 10
Output: "The sum of even numbers up to 10 is 30."

Write the algorithm in pseudocode:

Algorithm SumEvenNumbers

```
  Procedure SumEvenNumbers(n)
    sum = 0
    for i from 1 to n do
      if i % 2 == 0 then
        sum = sum + i
      end if
    end for
    Output "The sum of even numbers up to " + n + " is " + sum + "."
  End Procedure

End Algorithm
```

Program development: Analysis, Design, Coding, Testing and Verification

Analysis:

Purpose: The goal of the analysis stage is to understand the problem that the program is supposed to solve. This involves gathering requirements from stakeholders, defining the scope of the project, and identifying the functionalities that the program needs to support

Actions:

- Gathering requirements from clients, end-users, and other stakeholders.
- Analyzing existing systems or processes to identify areas for improvement.
- Defining use cases, user stories, or functional specifications.
- Identifying potential risks and constraints.

Design:

Purpose: The design stage focuses on creating a blueprint or plan for how the program will be implemented. This includes defining the overall architecture, data structures, algorithms, and user interface.

Actions:

- **Architectural design:** Defining the overall structure of the program, including modules, components, and their interactions.
- **Detailed design:** Specifying data structures, algorithms, and data flows.
- **User interface design:** Designing the layout, navigation, and interaction patterns of the program's user interface.
- **Database design:** Designing the structure of the database and its relationships.

Coding:

Purpose: The coding stage involves translating the design specifications into actual code. Developers write code according to the design, following coding standards and best practices.

Actions:

- Writing source code in the chosen programming language(s).
- Implementing algorithms, data structures, and business logic.
- Writing documentation, comments, and annotations to explain the code.

Testing:

Purpose: The testing stage is crucial for identifying and fixing defects or errors in the program. It ensures that the software meets the specified requirements and functions correctly.

Actions:

- **Unit testing:** Testing individual components or modules in isolation.
- **Integration testing**: Testing the interaction between different components/modules.
- **System testing:** Testing the entire system as a whole.
- **Regression testing:** Testing to ensure that new changes do not introduce new defects.
- **User acceptance testing (UAT):** Testing by end-users to validate that the software meets their needs.

Verification:

Purpose: Verification involves evaluating whether the program meets the specified requirements and adheres to the design specifications.

Actions:

- Reviewing code and design documents to ensure compliance with requirements.
- Conducting walkthroughs, inspections, or code reviews.
- Performing static analysis to identify potential issues in the code.
- Comparing the actual behavior of the software against expected results.