

Ch1. Problem Solving using Flowchart and Algorithm

1.1 Introduction

A flowchart is a graphical representation of an algorithm or a process. It visually illustrates the sequence of steps, decisions, and operations involved in solving a problem or executing a task. Flowcharts use standardized symbols to represent different types of actions or steps (e.g., ovals for start/end, rectangles for processes, diamonds for decisions) and connecting arrows to indicate the flow of control and information. Flowcharts serve as a visual aid for understanding, designing, documenting, and analyzing algorithms and processes, making complex logic more accessible and easier to follow, especially for beginners in programming. They are a powerful tool for planning and visualizing program logic before writing actual code.

An algorithm is a finite set of well-defined, unambiguous instructions or a step-by-step procedure for solving a computational problem or achieving a specific task. Algorithms are fundamental to computer science and programming, providing the logical blueprint for how a program will process data and produce results. Key properties of algorithms include being finite (terminating after a finite number of steps), definite (each step is precisely defined), having inputs, producing outputs related to those inputs, and being effective (each step can be performed in a finite amount of time).

1.1.1 Steps for problem-solving

Problem-solving using algorithms and flowcharts involves a systematic approach to define, analyze, and solve a problem.

1. Define the Problem:

- Clearly state the problem to be solved.
- Identify the inputs required and the desired outputs.

2. Develop the Algorithm:

- **Break down the problem:** Divide the main problem into smaller, manageable steps.
- **Sequence the steps:** Arrange the steps in a logical order of execution.
- **Include control structures:**
 - **Sequence:** Instructions executed one after another.
 - **Selection (Decision):** Use conditional statements (e.g., IF-THEN-ELSE) to execute different paths based on a condition.
 - **Iteration (Looping):** Repeat a set of instructions until a specific condition is met (e.g., FOR, WHILE loops).
- **Write the algorithm:** Express the steps in a clear, unambiguous language (e.g., pseudocode or structured English).

3. Create the Flowchart:

- **Start/Stop:** Use oval symbols to mark the beginning and end of the process.
- **Input/Output:** Use parallelogram symbols for data input or output.

- **Process:** Use rectangle symbols for calculations or processing steps.
- **Decision:** Use diamond symbols for conditional tests, with multiple paths originating from them (e.g., "Yes" or "No").
- **Flow Lines:** Use arrows to connect the symbols and indicate the flow of control.
- **Connectors:** Use small circles to connect different parts of a complex flowchart.

4. Test and Refine:

- **Trace the algorithm/flowchart:**

Manually follow the steps with sample inputs to verify the logic and identify any errors.

- **Refine as needed:**

Modify the algorithm or flowchart if errors are found or if there are more efficient ways to achieve the desired outcome.

Example (Algorithm to find the larger of two numbers):

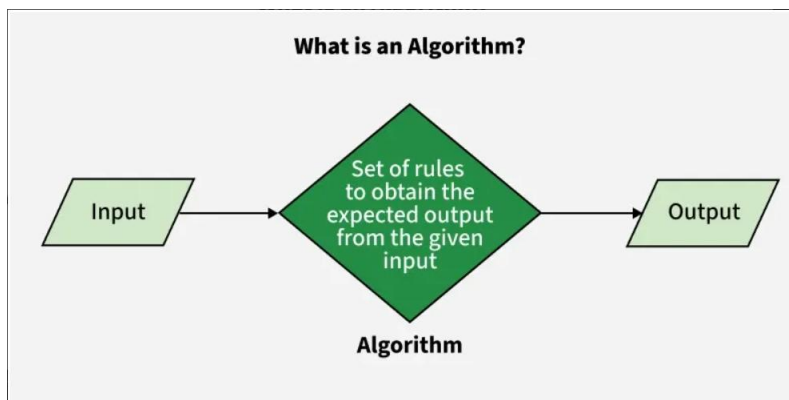
Code

Algorithm: Find_Larger_Number

1. START
2. INPUT number1, number2
3. IF number1 > number2 THEN
 PRINT number1 is larger
4. ELSE
 PRINT number2 is larger
5. END

1.1.2 Algorithm

The word **Algorithm** means "A set of finite rules or instructions to be followed in calculations or other problem-solving operations"



Use of Algorithms

- ➔ Algorithms are necessary for solving complex problems efficiently and effectively.
- ➔ They help to automate processes and make them more reliable, faster, and easier to perform.
- ➔ Algorithms also enable computers to perform tasks that would be difficult or impossible for humans to do manually.
- ➔ They are used in various fields such as mathematics, computer science, engineering, finance, and many others to optimize processes, analyze data, make predictions, and provide solutions to problems.

Characteristics of algorithm

- **Clear and Unambiguous:** The algorithm should be unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs:** If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- **Feasible:** The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions t
- **Clear and Unambiguous:** The algorithm should be unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs:** If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- **Feasible:** The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.
- **Input:** An algorithm has zero or more inputs. Each that contains a fundamental operator must accept zero or more inputs.
- **Output:** An algorithm produces at least one output. Every instruction that contains a fundamental operator must accept zero or more inputs.
- **Definiteness:** All instructions in an algorithm must be unambiguous, precise, and easy to interpret. By referring to any of the instructions in an algorithm one can clearly understand

what is to be done. Every fundamental operator in instruction must be defined without any ambiguity.

- **Finiteness:** An algorithm must terminate after a finite number of steps in all test cases. Every instruction which contains a fundamental operator must be terminated within a finite amount of time. Infinite loops or recursive functions without base conditions do not possess finiteness.
- **Effectiveness:** An algorithm must be developed by using very basic, simple, and feasible operations so that one can trace it out by using just paper and pencil.

Advantages of Algorithms:

- It is easy to understand.
- An algorithm is a step-wise representation of a solution to a given problem.
- In an 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(**imp**).

How to Design an Algorithm?

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** is to be expected when the problem is solved.
5. The **solution** to this problem is within the given constraints.

Algorithms are crucial because they provide structured, step-by-step instructions to solve problems, automate tasks, and make decisions efficiently in technology and daily life. They enable computers to process vast amounts of data, power applications like search engines and AI, optimize complex processes, and ensure consistent, reliable outcomes by breaking down challenges into manageable steps.

Why Algorithms Are Important

- **Problem-Solving:**

Algorithms provide a fixed, systematic method to solve problems, breaking complex challenges into smaller, understandable steps.

- **Automation:**

They automate tasks, making them faster, more reliable, and more efficient than manual processes, especially in computer systems.

- **Decision-Making:**

Algorithms form the basis for intelligent decision-making in systems like artificial intelligence and machine learning, helping computers learn from data and make predictions.

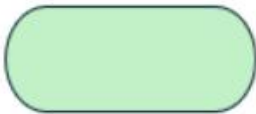
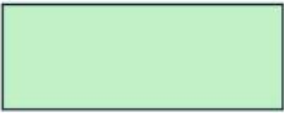

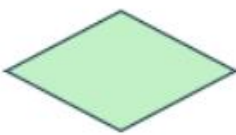

- **Efficiency:**

A core goal of algorithms is to be efficient, meaning they solve problems quickly and use resources (like time and memory) as effectively as possible.

- **Foundation of Technology:**

Algorithms are the bedrock of modern technology, powering everything from the search results you see to the recommendations on your streaming services and the navigation in your car.

1.2 Symbolic representation of a flowchart

Symbol	Name	Function
	Oval	Represents the start or end of a process
	Rectangle	Denotes a process or operation step
	Arrow	Indicates the flow between steps
	Diamond	Signifies a point requiring a yes/no
	Parallelogram	Used for input or output operations

1.2.1 Importance / Advantages of using a Flowchart

- Flowcharts are a better way of communicating the logic of the system.
- Flowcharts act as a guide for blueprint during program designed.
- Flowcharts help in debugging process.
- With the help of flowcharts programs can be easily analyzed.
- They provide better documentation.
- Flowcharts serve as a good proper documentation.

1.2.2 Disadvantages / limitations of using a Flowchart

- It is difficult to draw flowcharts for large and complex programs.
- There is no standard to determine the amount of detail.
- It is very difficult to modify the Flowchart.
- Making a flowchart is costly.
- If changes are done in software, then the flowchart must be redrawn

1.2.3 Flow of control

Control Flow Statements Type	Control Flow Statement	Description
Conditional Statements	if-else	Executes a block of code if a specified condition is true, and another block if the condition is false.
	switch-case	Evaluates a variable or expression and executes code based on matching cases.
Looping Statements	for	Executes a block of code a specified number of times, typically iterating over a range of values.
	while	Executes a block of code as long as a specified condition is true.

Control Flow Statements Type	Control Flow Statement	Description
	do-while	Executes a block of code once and then repeats the execution as long as a specified condition is true.
Jump Statements	break	Terminates the loop or switch statement and transfers control to the statement immediately following the loop or switch.
	continue	Skips the current iteration of a loop and continues with the next iteration.
	return	Exits a function and returns a value to the caller.
	goto	Transfers control to a labeled statement within the same function. (Note: goto is generally discouraged due to its potential for creating unreadable and error-prone code.)

1.3 Problem solving using 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 the **intermediate state between an idea and its implementation(code)** in a high-level language.

Problem solving using pseudocode involves outlining the logical steps of an algorithm or program in a high-level, human-readable format, independent of any specific programming language syntax. This approach aids in structuring solutions before actual coding begins.

Steps for Problem Solving with Pseudocode:

- **Understand the Problem:**

Clearly define the problem to be solved, including inputs, expected outputs, and any constraints.

- **Break Down the Problem:**

Divide complex problems into smaller, more manageable sub-problems or modules.

- **Outline the Logic:**

Write out the steps of the solution using clear, concise, English-like statements. This includes control structures like IF-THEN-ELSE, WHILE, FOR, and basic operations like READ, PRINT, COMPUTE.

- **Refine and Review:**

Evaluate the pseudocode for clarity, correctness, and efficiency. Ensure all necessary steps are included and the logic flows correctly. This stage helps in identifying potential issues early.

- **Translate to Code:**

Once the pseudocode is finalized, translate each step into the syntax of the chosen programming language.

Difference between Algorithm and Pseudocode.

Algorithm	Pseudocode
An Algorithm is used to provide a solution to a particular problem in form of a well-defined step-based form.	A Pseudocode is a step-by-step description of an algorithm in code-like structure using plain English text.
An algorithm only uses simple English words	Pseudocode also uses reserved keywords like if-else, for, while, etc.
These are a sequence of steps of a solution to a problem	These are fake codes as the word pseudo means fake, using code like structure and plain English text
There are no rules to writing algorithms	There are certain rules for writing pseudocode
Algorithms can be considered pseudocode	Pseudocode cannot be considered an algorithm
It is difficult to understand and interpret	It is easy to understand and interpret