DESIGN AND ANALYSIS OF ALGORITHM
UNIT 1

INTRODUCTION

WHAT IS AN ALGORITHM?

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

Or

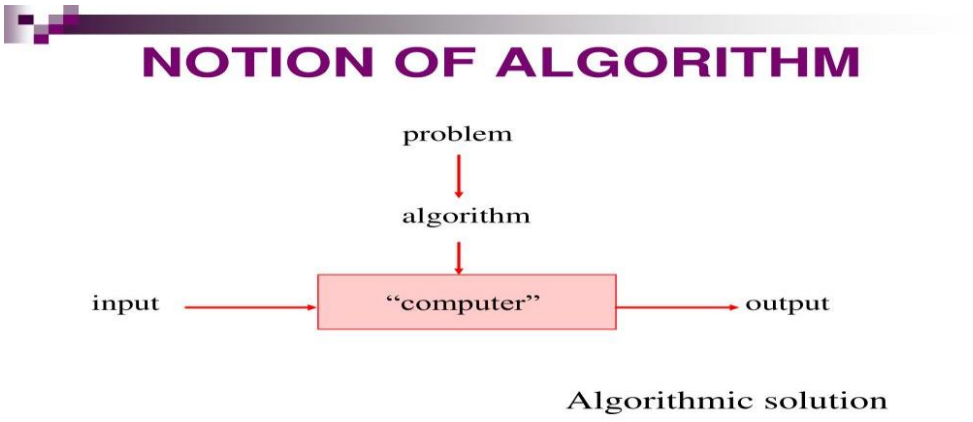
" A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations".

Or

An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. An algorithm is an efficient method that can be expressed within finite amount of time and space.

Or

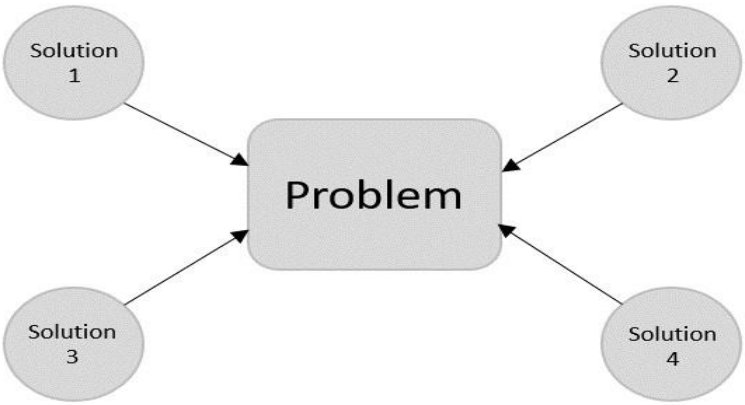
An algorithm is the best way to represent the solution of a particular problem in a very simple and efficient way. If we have an algorithm for a specific problem, then we can implement it in any programming language, meaning that the algorithm is independent from any programming languages.



Algorithm Design

The important aspects of algorithm design include creating an efficient algorithm to solve a problem in an efficient way using minimum time and space.

To solve a problem, different approaches can be followed. Some of them can be efficient with respect to time consumption, whereas other approaches may be memory efficient. However, one has to keep in mind that both time consumption and memory usage cannot be optimized simultaneously. If we require an algorithm to run in lesser time, we have to invest in more memory and if we require an algorithm to run with lesser memory, we need to have more time.



How to Design an Algorithm?

To write an algorithm, the following things are needed as a pre-requisite:

- The problem that is to be solved by this algorithm i.e. clear problem definition.
- The constraints of the problem must be considered while solving the problem.
- The input to be taken to solve the problem.
- The output is to be expected when the problem is solved.
- The solution to this problem is within the given constraints.

Example

Let's try to learn algorithm-writing by using an example.

Problem − Design an algorithm to add two numbers and display the result.

Step 1 − START

Step 2 − declare three integers a, b & c

Step 3 − define values of a & b

Step 4 − add values of a & b

Step 5 − store output of step 4 to c

Step 6 − print c

Step 7 − STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

Step 1 − START ADD

Step 2 − get values of a & b

Step 3 − c ← a + b

Step 4 − display c

Step 5 − STOP

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Characteristics of Algorithms

Not all procedures can be called an algorithm. An algorithm should have the following characteristics −

- **Unambiguous −** Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input −** An algorithm should have 0 or more well-defined inputs.
- **Output −** An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness −** Algorithms must terminate after a finite number of steps.
- **Feasibility −** Should be feasible with the available resources.

Pseudocode

Pseudocode gives a high-level description of an algorithm without the ambiguity associated with plain text but also without the need to know the syntax of a particular programming language.

The running time can be estimated in a more general manner by using Pseudocode to represent the algorithm as a set of fundamental operations which can then be counted.

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).

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING

A sequence of steps involved in designing and analyzing an algorithm is shown in the figure below:

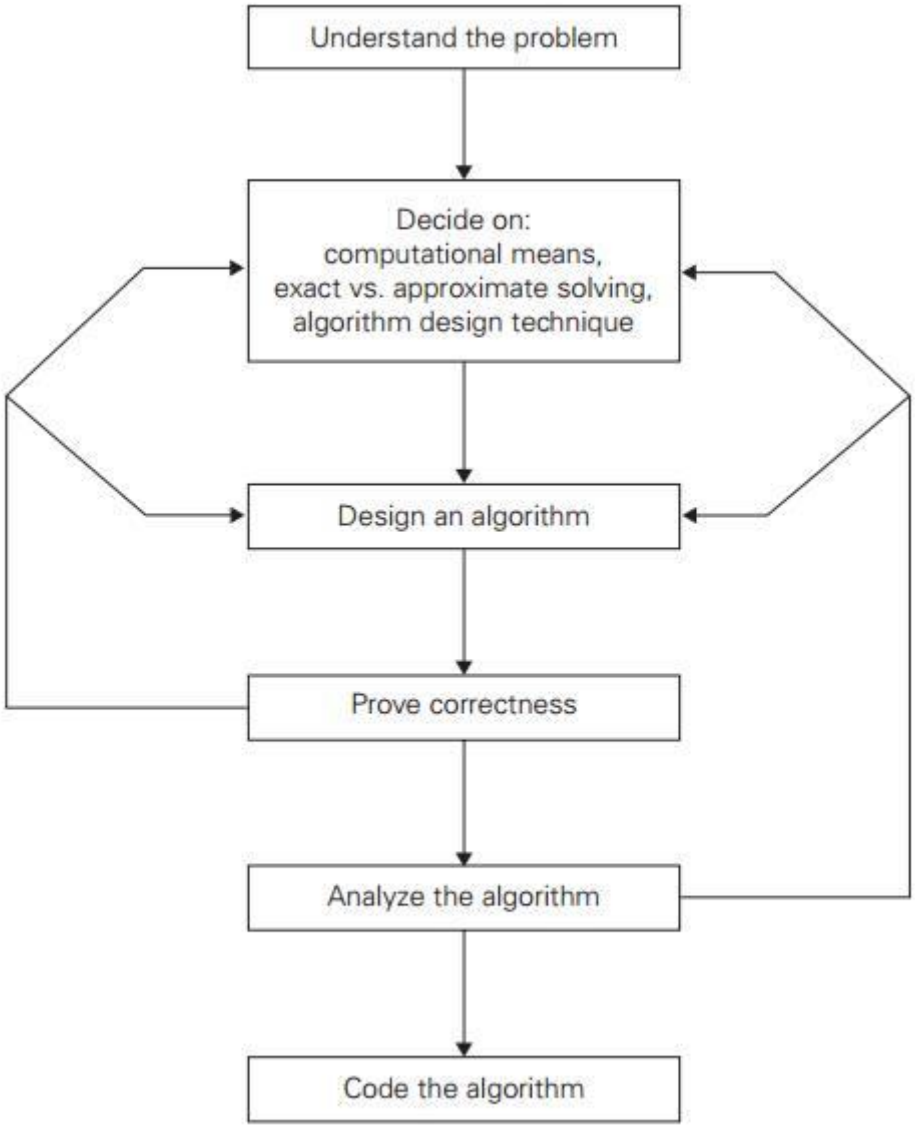


FIGURE 1.2 Algorithm design and analysis process.

DESIGN AND ANALYSIS OF ALGORITHM
UNIT 1

UNDERSTANDING THE PROBLEM

- This is the first step in designing of algorithm.
- Read the problem's description carefully to understand the problem statement completely.
- Ask questions for clarifying the doubts about the problem.
- Identify the problem types and use existing algorithm to find solution.
- Input (instance) to the problem and range of the input get fixed

DECISION MAKING

The Decision making is done on the following:

A computational problem is a problem that can be solved step-by-step with a computer. These problems usually have a well-defined input, constraints, and conditions that the output must satisfied. Here are some types of computational problems: A decision problem is one where the answer is yes or no.

An exact algorithm is a computational method that guarantees a mathematically precise solution to a given problem, while an **approximate algorithm** provides an approximate solution that is close to the optimal solution, but not necessarily exact.

An algorithm design technique means a unique approach or mathematical method for creating algorithms and solving problems. While multiple algorithms can solve a problem, not all algorithms can solve it efficiently.

DESIGN AN ALGORITHM

The algorithms which follow the divide & conquer techniques involve three steps: Divide the original problem into a set of subproblems. Solve every subproblem individually, recursively. Combine the solution of the subproblems (top level) into a solution of the whole original problem.

PROVE CORRECTNESS

Correctness: An algorithm's correctness is defined as when the given inputs produce the desired output, indicating that the algorithm was designed correctly. An algorithm's analysis has been completed correctly.

An algorithm must be correct because we rely upon it for our desired output.

There are two types of correctness: partial correctness and total correctness

Partial correctness

An algorithm is **partially correct** if it receives valid input and then terminates.

Total correctness

An algorithm is **totally correct** if it receives valid input, terminates, and always returns the correct output.

ANALYZING AN ALGORITHM

In the analysis of the algorithm, it generally focused on CPU (time) usage, Memory usage, Disk usage, and Network usage. All are important, but the most concern is about the CPU time. Be careful to differentiate between:

Performance: How much time/memory/disk/etc. is used when a program is run. This depends on the machine, compiler, etc. as well as the code we write.

Complexity: How do the resource requirements of a program or algorithm scale, i.e. what happens as the size of the problem being solved by the code gets larger.

CODING AN ALGORITHM

The coding / implementation of an algorithm is done by a suitable programming language like C, C++, JAVA.

The transition from an algorithm to a program can be done either incorrectly or very inefficiently. Implementing an algorithm correctly is necessary. The Algorithm power should not reduce by inefficient implementation.

Standard tricks like computing a loop's invariant (an expression that does not change its value) outside the loop, collecting common sub-expressions, replacing expensive operations by cheap ones, selection of programming language and so on should be known to the programmer.

FUNDAMENTALS OF THE ANALSIS OF ALGORITHM EFFICIENCY

The efficiency of an algorithm is defined as the number of computational resources used by the algorithm. An algorithm must be analysed to determine its resource usage. The efficiency of an algorithm can be measured based on the usage of different resources.
An algorithm which takes fewer resources and computes results in a minimum time for a problem then that algorithm is known as efficient.
The efficiency of the algorithm is measured based on the usage of different resources and minimum running time.

1. Space efficiency
2. Time efficiency

Space efficiency:

The space complexity of an algorithm describes the amount of memory required during the program execution as the function input. Generally, the space needed.
Space required by an algorithm equals the sum of the following two components.

- A fixed part independent of the particular problem includes instruction space, space for constants, simple variables, and fixed-size structure variables.
- A variable part includes structured variables whose size depends on the problem being solved.

Space complexity S(p) of any algorithm a is

$$S(p) = C + Sp,$$

where 'C' is the fixed part, and 'Sp' is the variable part of the algorithm, which depends on an instance characteristic i.

Examples:

Algorithm: sum(a,b)

//input: 2 numbers a and b

//output : sum of a and b

Return a+r

Solution:

Assume constant values as 1

Now as we don't have any supporting variables so Sp=0

S(p)= C+Sp

C=a+b=1+1=2 and Sp=0

S(p)=2+Sp

Therefore, S(p)=2

DESIGN AND ANALYSIS OF ALGORITHM
UNIT 1

Time complexity

The total time needed to complete the execution of an algorithm is called time complexity. The time complexity depends on the "basic operations" – most important operation of the algorithm, i.e the operations contributes the most of the total running time.

$$T(x)= Cop * Cn$$

Where,

T: time complexity

Cop: time taken for one execution of basic operation

Cn: function which express how many times the basic operation is executed

The basic operation depends on,

- Worst case efficiency
- Best case efficiency
- Average case efficiency

Worst case efficiency

- Maximum number of times the basic operation gets executed for input n.
- we calculate the upper bound on the running time and we define an algorithms worst case time complexity by big O notation which determines the set of functions grows slower or same rate as expression.
- Example : linear search – o(1), o- orderoff

best case efficiency

- Manimum number of times the basic operation gets executed for input n.
- we calculate the lower bound on the running time and we define an algorithms best case time complexity by "big omega" notation which determines the set of functions grows slower or same rate as expression.
- Example : linear search – o(n), o- orderoff

Average case efficiency

- The number of comparison is minimum or maximum then average case comes into existence.
- we define an algorithms best case time complexity by "big theta" notation which determines the set of functions grows slower or same rate as expression.