Algorithm Created by github.com/farzadafi

define

- 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 "

use case

- Algorithms play a crucial role in various fields and have many applications.
- Some of the key areas where algorithms are used include
  - Computer Science
    - Algorithms form the basis of computer programming and are used to solve problems ranging from simple sorting and searching to complex tasks such as artificial intelligence and machine learning
  - Mathematics
  - ...
- Algorithms can be simple and complex depending on what you want to achieve.
- Every time you operate your phone, computer, laptop, or calculator you are using Algorithms.
- The Algorithm designed are language-independent, i.e. they are just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

What is the need for 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 performance, make accurate predictions, and provide solutions to problems.

Characteristics of an 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 that can be implemented in any language, and yet the output will be the same, as expected.
- 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.

Types of Algorithms

- Brute Force Algorithm
  - It is the simplest approach to a problem. A brute force algorithm is the first approach that comes to finding when we see a problem.
- Recursive Algorithm
  - A recursive algorithm is based on recursion. In this case, a problem is broken into several sub-parts and called the same function again and again.
- Backtracking Algorithm
- Backtracking Algorithm
- Searching Algorithm
- Sorting Algorithm
- Hashing Algorithm
- Divide and Conquer Algorithm
- ...

Advantages

- 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

- 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).

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

Step 1: Fulfilling the pre-requisites

- 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: We can solve this problem by adding the 3 numbers, it can be done with the help of the '+' operator, or bit-wise, or any other method.

Step 2: Designing the algorithm

- 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.

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int num1, num2, num3;
    int sum;
    cout << "Enter the 1st number: ";
    cin >> num1;
    cout << " " << num1 << endl;
    cout << "Enter the 2nd number: ";
    cin >> num2;
    cout << " " << num2 << endl;
    cout << "Enter the 3rd number: ";
    cin >> num3;
    cout << " " << num3;
    sum = num1 + num2 + num3;
    cout << "\nSum of the 3 numbers is: "
         << sum;
    return 0;
}
```

Java

```java
import java.util.*;
class API {
    public static void main(String[] args)
    {
        int sum = 0;

        Scanner sc
            = new Scanner(System.in); // Scanner definition

        System.out.println("Enter the 1st number: ");
        int num1 = sc.nextInt();

        System.out.println(" " + num1);
        System.out.println("Enter the 2nd number: ");
        int num2 = sc.nextInt();
        System.out.println(" " + num2);
        System.out.println("Enter the 3rd number: ");
        int num3 = sc.nextInt();
        System.out.println(" " + num3);


        sum = num1 + num2 + num3;
        System.out.println("Sum of the 3 numbers is: "
                           + sum);
    }
}
```

One problem, many solutions :)

- The solution to an algorithm can be or cannot be more than one.
- The problem can be solved sometimes with a single algorithm, sometimes it depends on the mind of the programmer that want to solve the problem
- It means that while implementing the algorithm, there can be more than one method to implement it.

analyze an Algorithm

- For a standard algorithm to be good, it must be efficient.
- Hence the efficiency of an algorithm must be checked and maintained. It can be in two stages:
  - Priori Analysis:
    - "Priori" means "before".
    - Hence Priori analysis means checking the algorithm before its implementation.
    - In this, the algorithm is checked when it is written in the form of theoretical steps.
    - This Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.
  - Posterior Analysis:
    - "Posterior" means "after".
    - Hence Posterior analysis means checking the algorithm after its implementation.
    - In this, the algorithm is checked by implementing it in any programming language and executing it.
    - This analysis helps to get the actual and real analysis report on correctness, space required, time consumed, etc.

Algorithm complexity

- An algorithm is defined as complex based on the amount of Space and Time it consumes.
- Hence the Complexity of an algorithm refers to the measure of the time that it will need to execute and get the expected output, and the Space it will need to store all the data (input, temporary data, and output).
- Hence these two factors define the efficiency of an algorithm.
- Therefore the complexity of an algorithm can be divided into two types:
  - Space Complexity
    - The space complexity of an algorithm refers to the amount of memory required by the algorithm to store the variables and get the result.
    - This can be for inputs, temporary operations, or outputs.
    - calculate Space Complexity
      - The space complexity of an algorithm is calculated by determining the following 2 components:
        - Fixed Part
          - This refers to the space that is required by the algorithm. For example, input variables, output variables, program size, etc.
        - Variable Part
          - This refers to the space that can be different based on the implementation of the algorithm. For example, temporary variables, dynamic memory allocation, recursion stack space, etc.
  - Time Complexity
    - The time complexity of an algorithm refers to the amount of time required by the algorithm to execute and get the result.
    - This can be for normal operations, conditional if-else statements, loop statements, etc.
    - Calculate, Time Complexity
      - The time complexity of an algorithm is also calculated by determining the following 2 components:
        - Constant time part:
          - Any instruction that is executed just once comes in this part. For example, input, output, if-else, switch, arithmetic operations, etc.
        - Variable Time Part
          - Any instruction that is executed more than once, say n times, comes in this part. For example, loops, recursion, etc.

express an Algorithm

- Natural Language
  - Here we express the Algorithm in the natural English language. It is too hard to understand the algorithm from it.
- Flow Chart:
  - Here we express the Algorithm by making a graphical/pictorial representation of it.
  - It is easier to understand than Natural Language.
- example

- 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 way we write code in different programming languages, except that 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 knowledge.
- example

```
if "1"
   print response
        "I am case 1"

if "2"
   print response
        "I am case 2"
```