Algorithm definition:
An algorithm is a well-defined sequential computational technique that accepts a value or a collection of values as input and produces the output(s) needed to solve a problem.

Or we can say that an algorithm is said to be accurate if and only if it stops with the proper output for each input instance.

Example:
Consider a box where no one can see what’s happening inside, we say a black box.

We give input to the box and it gives us the output we need but the procedure that we might need to know behind the conversion of input to desired output is an ALGORITHM.

An algorithm is independent of the language used. It tells the programmer the logic used to solve the problem. So, it is a logical step-by-step procedure that acts as a blueprint to programmers.

Why do we use algorithms?
Consider two kids, Aman and Rohan, solving the Rubik’s Cube. Aman knows how to solve it in a definite number of steps. On the other hand, Rohan knows that he will do it but is not aware of the procedure. Aman solves the cube within 2 minutes whereas Rohan is still stuck and by the end of the day, he somehow managed to solve it (might have cheated as the procedure is necessary).

So the time required to solve with a procedure/algorithm is much more effective than that without any procedure. Hence the need for an algorithm is a must.

Types of Algorithms:
Sorting algorithms: Bubble Sort, insertion sort, and many more. These algorithms are used to sort the data in a particular format.
Searching algorithms: Linear search, binary search, etc. These algorithms are used in finding a value or record that the user demands.
Graph Algorithms: It is used to find solutions to problems like finding the shortest path between cities, and real-life problems like traveling salesman problems.

Algorithm Analysis:
Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Types of Algorithm Analysis:

1. Best case
2. Worst case
3. Average case

Best case: Define the input for which algorithm takes less time or minimum time. Example: In the linear search when search data is present at the first location of large data then the best case occurs.

Worst Case: Define the input for which algorithm takes a long time or maximum time. Example: In the linear search when search data is not present at all then the worst case occurs.

Average case: In the average case take all random inputs and calculate the computation time for all inputs.
And then we divide it by the total number of inputs.

Average case = all random case time / total no of case

Examples with their complexity analysis:

```
// C++ implementation of the approach
#include <bits/stdc++.h>
using namespace std;

// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
   int i;
   for (i = 0; i < n; i++) {
      if (arr[i] == x)
         return i;
   }
   return -1;
}

// Driver's Code
int main()
{
   int arr[] = { 1, 10, 30, 15 };
   int x = 30;
   int n = sizeof(arr) / sizeof(arr[0]);
```

```
    // Function call
    cout << x << " is present at index "
         << search(arr, n, x);

    return 0;
}
```

Time Complexity Analysis: (In Big-O notation)

Best Case: O(1), This will take place if the element to be searched is on the first index of the given list. So, the number of comparisons, in this case, is 1.

Average Case: O(n), This will take place if the element to be searched is on the middle index of the given list.

Worst Case: O(n), This will take place if:
The element to be searched is on the last index
The element to be searched is not present on the list

Practice Problem:
In this example, we will take an array of length (n) and deals with the following cases :

If (n) is even then our output will be 0
If (n) is odd then our output will be the sum of the elements of the array.