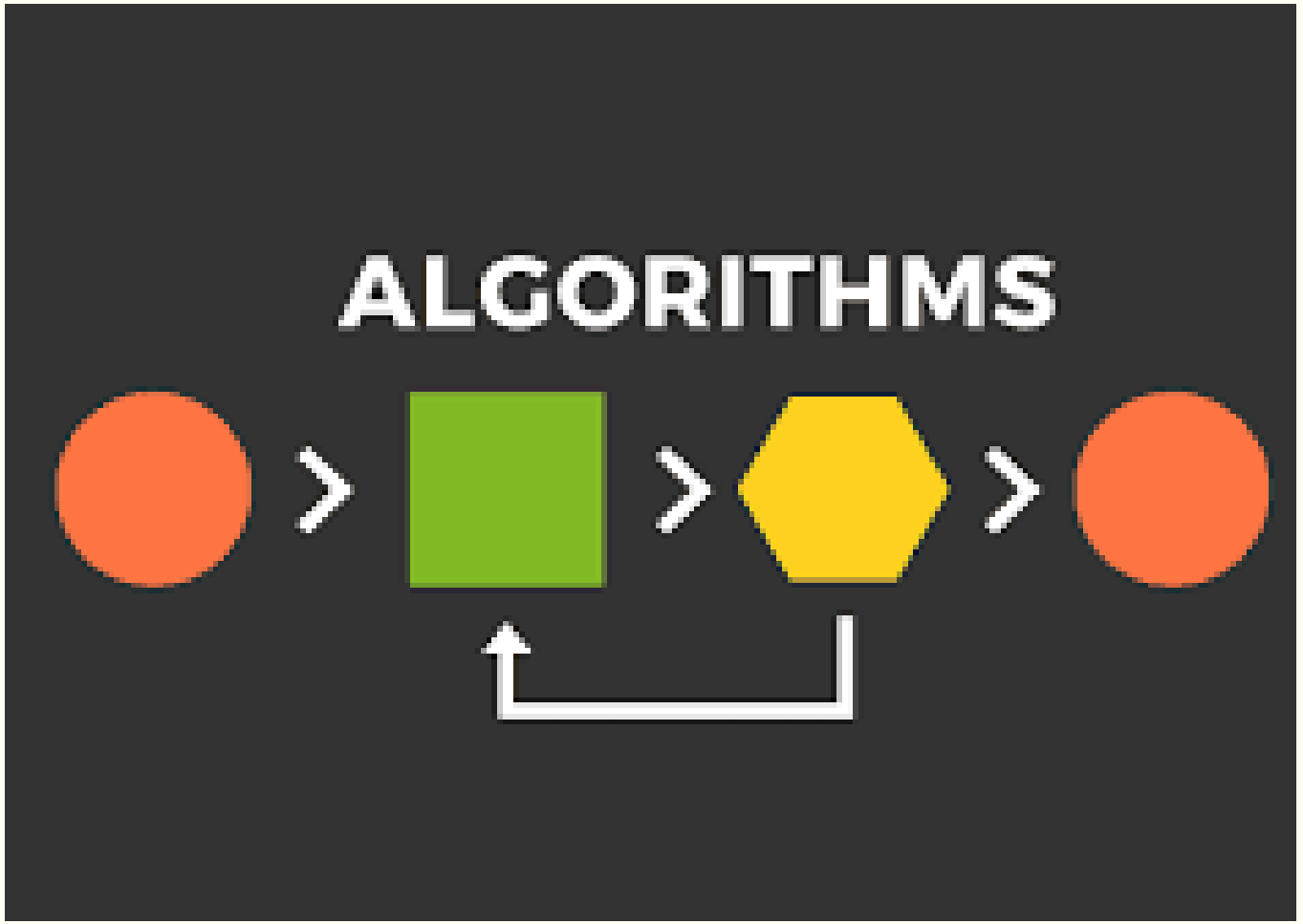
Algorithm Analysis



Algorithm Definitions

- An Algorithm is a finite set of instructions that,if followed, accomplishes a particular task.

Characteristics of Algorithm:

1) Input: An algorithm should have finite no of input

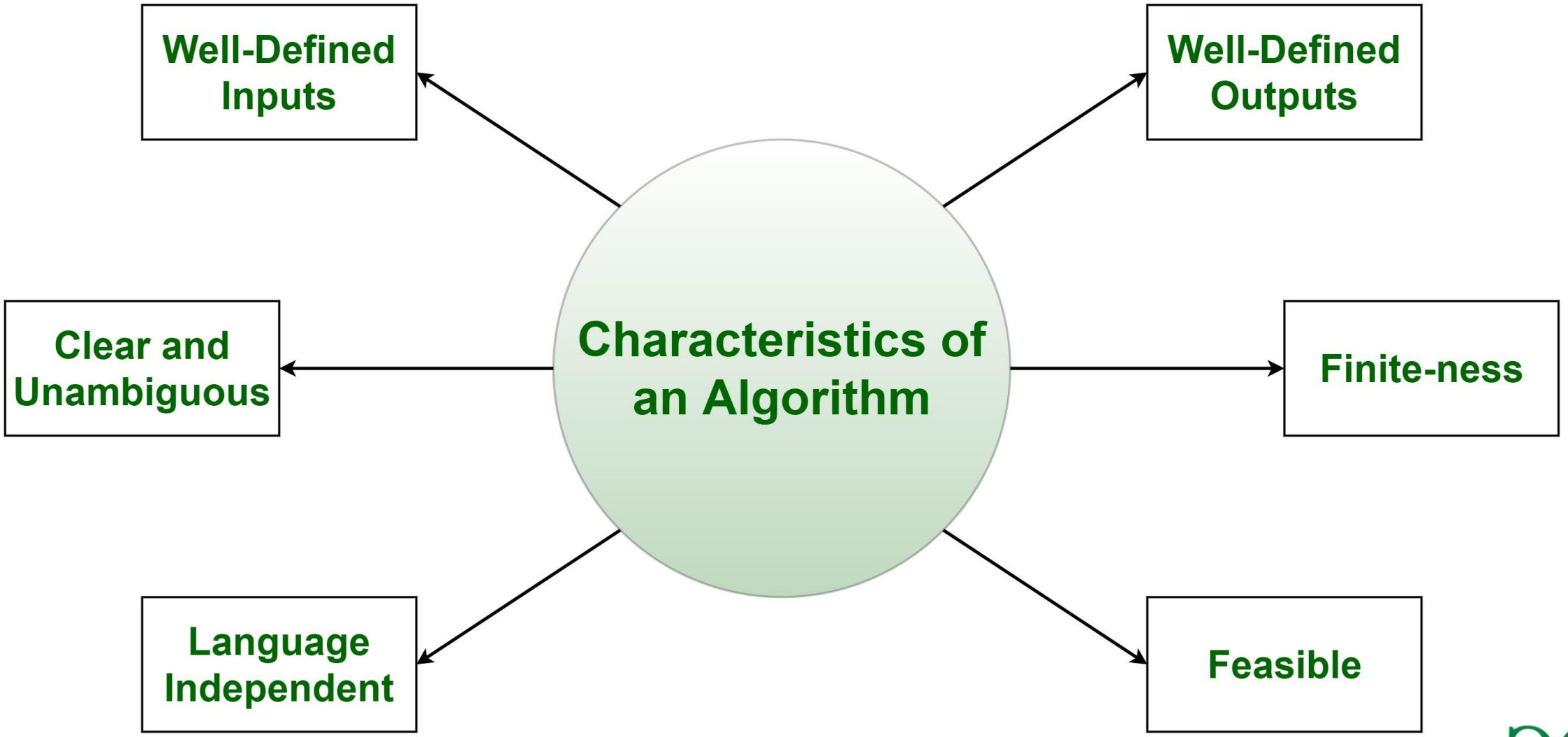
2) Output: An algorithm must produce at least one output

3) Definiteness: Each instruction should be clear & ambiguous

4) Finiteness: An algorithm must have finite no of instructions.
i.e it should terminate

5) Effectiveness: Every instruction of an algorithm must be feasible

Characteristics of an Algorithm



PERFORMANCE ANALYSIS

What is Performance Analysis of an algorithm?

- ✓ Suppose if we want to go from city "A" to city "B", there can be many ways of doing this. We can go by flight, by bus, by train and also by bicycle. Depending on the availability and convenience, we choose the one which suits us.

- ✓ Similarly, in computer science, there are multiple algorithms to solve a problem. When we have more than one algorithm to solve a problem, we need to select the best one. Performance analysis helps us to select the best algorithm from multiple algorithms to solve a problem.
- ✓ When there are multiple alternative algorithms to solve a problem, we analyze them and pick the one which is best suitable for our requirements.

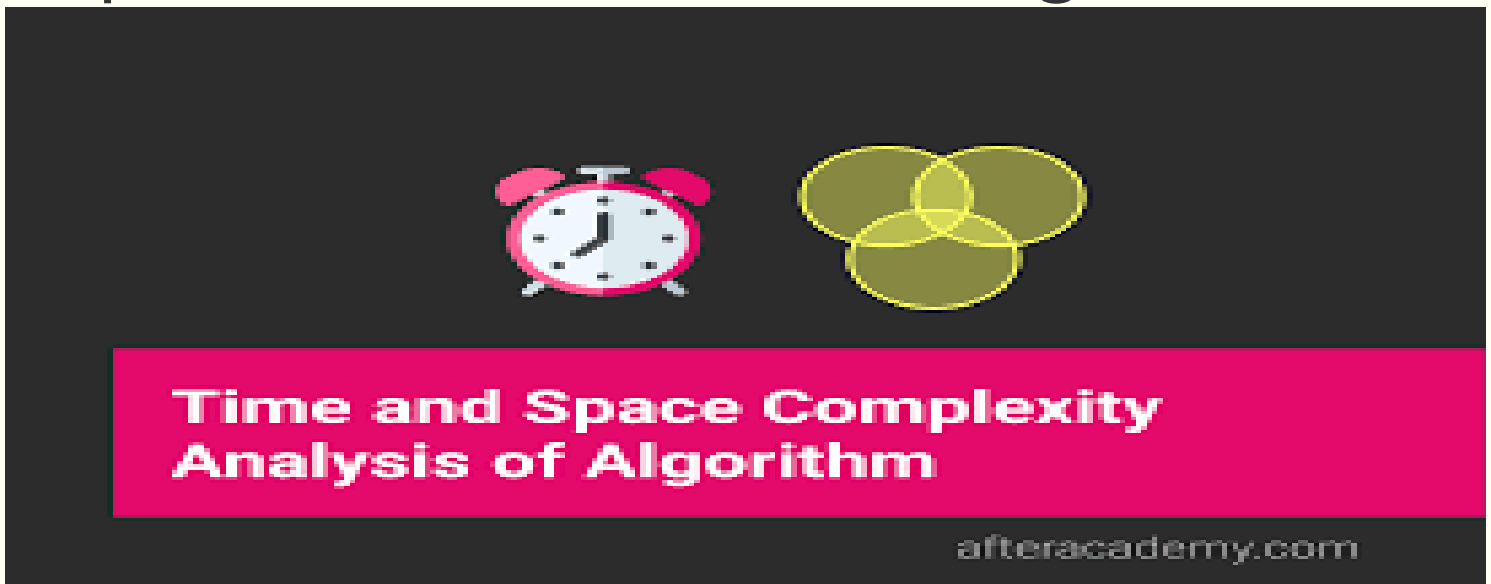
Generally, the performance of an algorithm depends on the following elements...

- Whether that algorithm is providing the exact solution for the problem?
- Whether it is easy to understand?
- Whether it is easy to implement?
- How much space (memory) it requires to solve the problem?
- How much time it takes to solve the problem? Etc.,

Performance analysis of an algorithm is performed by using the following measures...

- Space required to complete the task of that algorithm (**Space Complexity**). It includes program space and data space

- Time required to complete the task of that algorithm (**Time Complexity**)



What is Space complexity?

For any algorithm, memory is required for the following purposes...

1. **To store constant values.**
2. **To store program instructions.**
3. **To store variable values.**
4. **And for few other things like function calls, jumping statements etc,.**

Space complexity of an algorithm can be defined as follows...

Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm.

Consider the following example

```
int square(int a)
{
     return a * a;

 }
```

It requires 2 bytes of memory to store variable **'a'** and another 2 bytes of memory is used for **return value**.

Totally it requires 4 bytes of memory to complete its execution. And this 4 bytes of memory is fixed for any input value of 'a'. This space complexity is said to be *Constant Space Complexity*.

Consider the following example

```
int sum(int A[ ], int n)
{ int sum = 0, i;
 for(i = 0; i < n; i++)
sum = sum + A[i];
return sum; }
```

'n*2' bytes of memory to store array variable **'a[ ]'**
2 bytes of memory for integer parameter **'n'**
4 bytes of memory for local integer variables **'sum'** and **'i'** (2 bytes each)
2 bytes of memory for **return value**.

totally it requires '2n+8' bytes of memory to complete its execution. Here, the total amount of memory required depends on the value of 'n'. As 'n' value increases the space required also increases proportionately. This type of space complexity is said to be *Linear Space Complexity*.

What is Time complexity?

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution.

- To calculate exact Time complexity of program is very difficult task.
- So a rough estimate can possible with help of **Active operation in program.**
- The total number of active operations is defined as its **frequency count**
- After calculating the frequency count the Time complexity is expressed using an asymptotic notation.

Example

```
a=a+b;
```

This statements executes 1 time thus its frequency count is =1

```
for(i=1;i<=n;i++)
a=a*b;
```

This statements executes n time thus its frequency count is =n

```
for(i=1;i<=m;++i)
for(j=1;j<=n;j++)
a=a*b
```

This statements executes m*n time thus its frequency count is $m \sim n = n^2$

Asymptotic Notations

What is Asymptotic Notation?

Whenever we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm. But when we calculate the complexity of an algorithm it does not provide the exact amount of resource required. So instead of taking the exact amount of resource,

we represent that complexity in a general form (Notation) which produces the basic nature of that algorithm. We use that general form (Notation) for analysis process.

Asymptotic notation of an algorithm is a mathematical representation of its complexity.

Majorly, we use THREE types of Asymptotic Notations

1.Big - Oh (O)

2.Big - Omega (Ω)

3. Theta (Θ)

Big - Oh Notation (O)

- Big - Oh notation is used to define the **upper bound** of an algorithm in terms of Time Complexity.
- It always indicates **the maximum time required by an algorithm** for all input values.
- It describes the **worst case of an algorithm** time complexity.

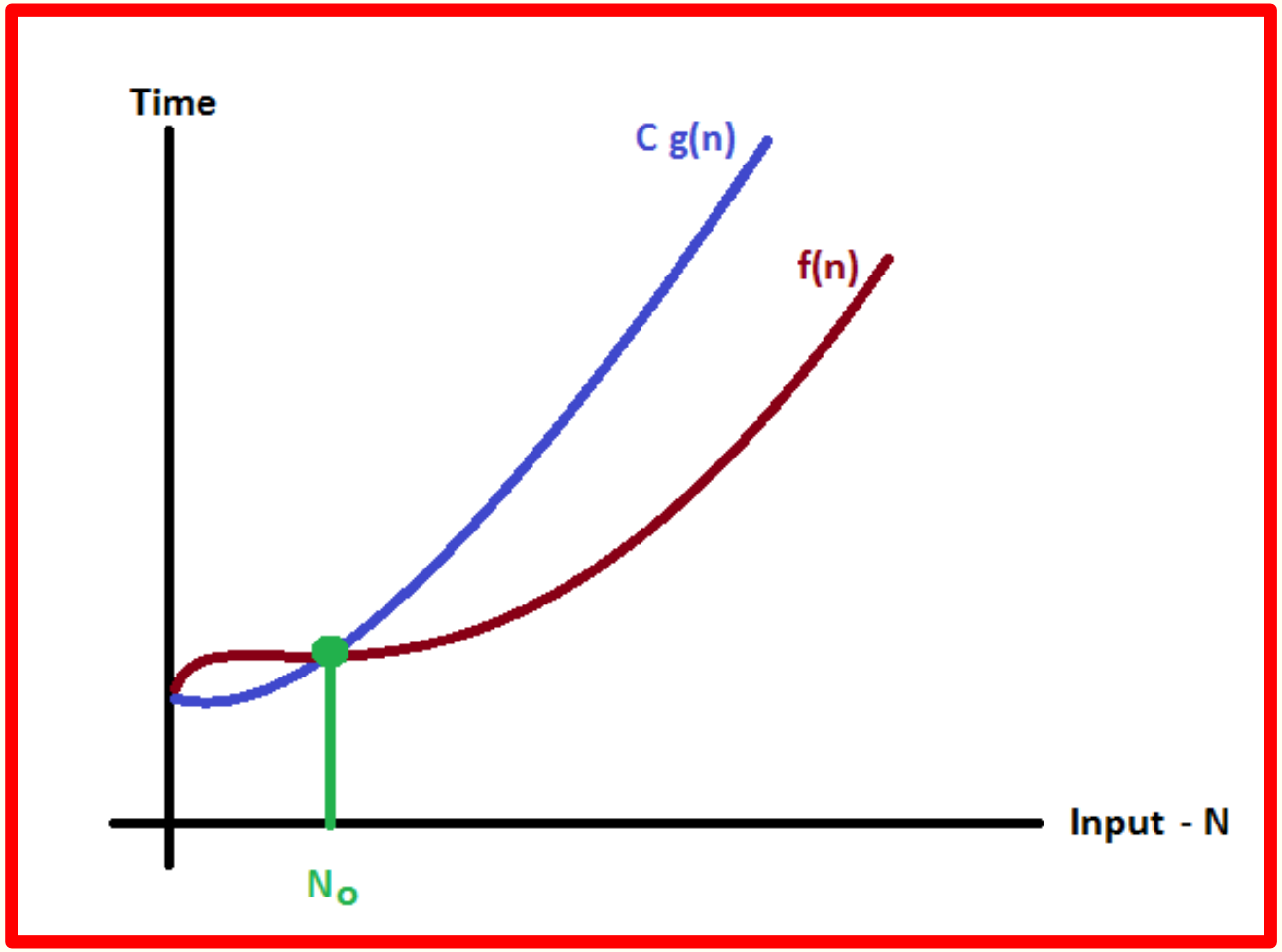
Big - Oh Notation can be defined as follows...

Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term.

If $f(n) <= C\ g(n)$ for all $n >= n_0$, $C > 0$ and $n_0 >= 1$. Then we can represent f(n) as O(g(n)).

$$f(n) = O(g(n))$$

Consider the following graph drawn for the values of f(n) and C g(n) for input (n) value on X-Axis and time required is on Y-Axis



In above graph after a particular input value n_0, always C g(n) is greater than f(n) which indicates the algorithm's upper bound.

Constant Time: O(1)

An algorithm is said to run in constant time if it requires the same amount of time regardless of the input size.

A=A +B

Above we have a single statement. Its Time Complexity will be Constant. The running time of the statement will not change in relation to N

Linear Time: O(n)

An algorithm is said to run in linear time if its time execution is directly proportional to the input size, i.e. time grows linearly as input size increases

```
for(i=1;i<=n;i++)
a=a*b;
```

The time complexity for the above algorithm will be Linear. The running time of the loop is directly proportional to N

Quadratic Time: $O(n^2)$

An algorithm is said to run in quadratic time if its time execution is proportional to the square of the input size.

```
for(i=1;i<=n;++i)
for(j=1;j<=n;j++)
a=a*b
```

This time, the time complexity for the above code will be Quadratic. The running time of the two loops is proportional to the square of N.

Big – Omega Notation (Ω)

- Big - Omega notation is used to define the **lower bound** of an algorithm in terms of Time Complexity.
- That means Big-Omega notation always indicates the minimum time required by an algorithm for all input values.
- Big-Omega notation describes the best case of an algorithm time complexity.

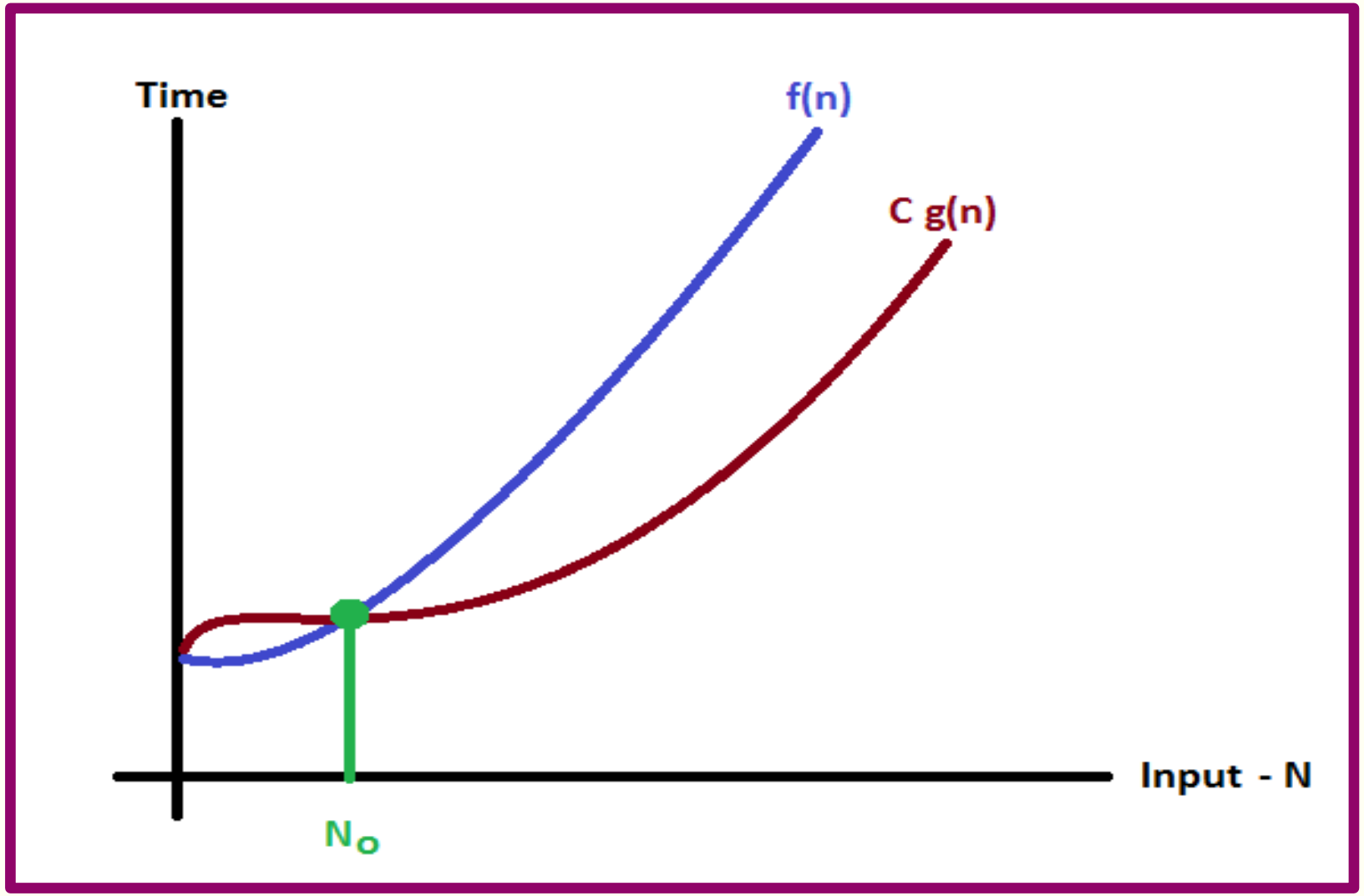
Big - Omega Notation can be defined as follows...

$$f(n) = \Omega(g(n))$$

Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term.

If $f(n) >= C\ g(n)$ for all $n >= n_0$, $C > 0$ and $n_0 >= 1$.

Then we can represent f(n) as Ω(g(n)).



In above graph after a particular input value n_0, always C g(n) is less than f(n) which indicates the algorithm's lower bound.

Big - Theta Notation (Θ)

- Big - Theta notation is used to define the **average bound** of an algorithm in terms of Time Complexity.
- Big - Theta notation always indicates the average time required by an algorithm for all input values.
- This notation describes the average case of an algorithm time complexity.

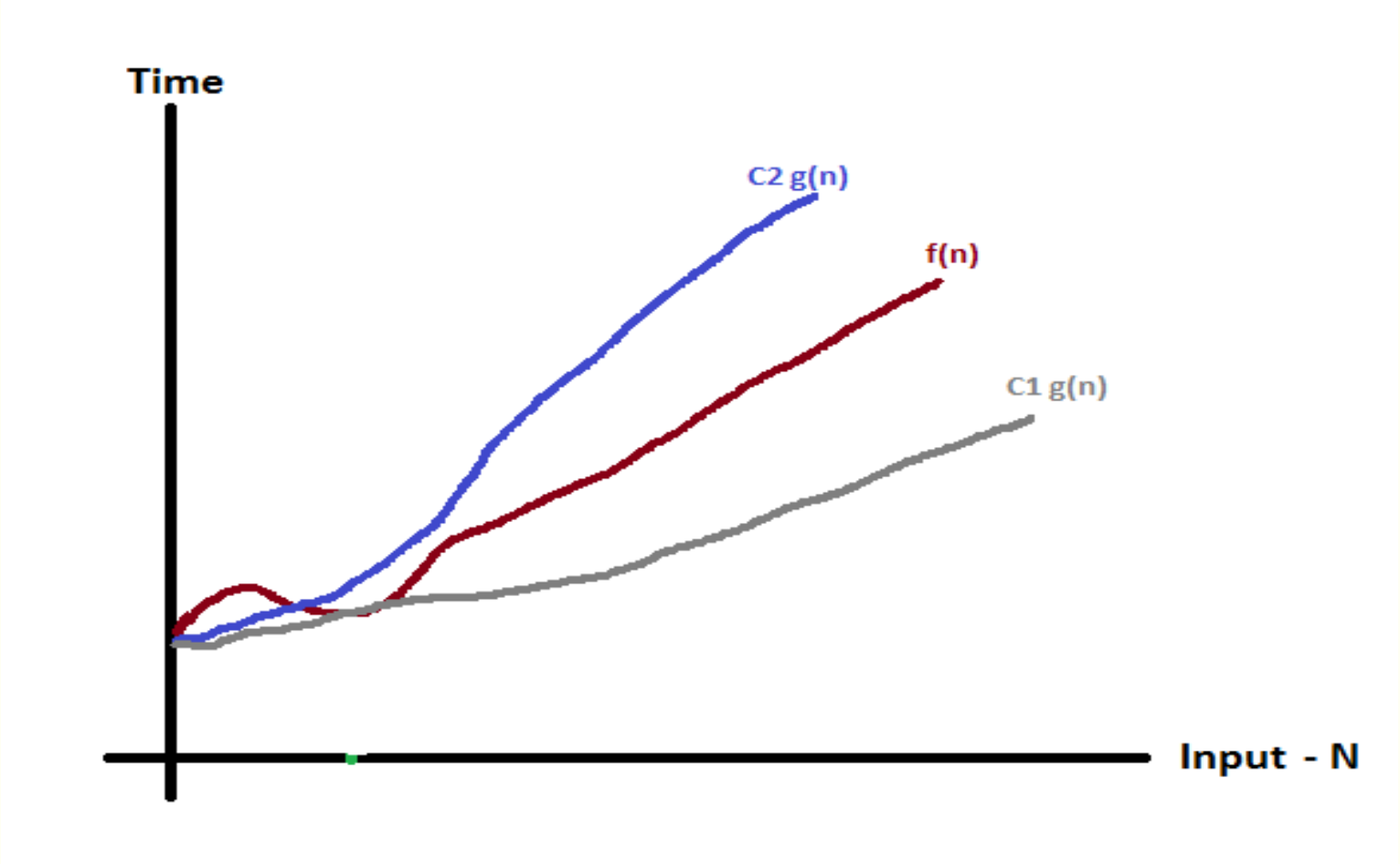
Big - Theta Notation can be defined as follows...

$$f(n) = \Theta(g(n))$$

Consider function f(n) as time complexity of an algorithm and g(n) is the most significant term.

If $C_1\ g(n) <= f(n) <= C_2\ g(n)$ for all $n >= n_0$, $C_1 > 0$, $C_2 > 0$ and $n_0 >= 1$.

Then we can represent f(n) as Θ(g(n)).



In above graph after a particular input value n_0, always C_1 g(n) is less than f(n) and C_2 g(n) is greater than f(n) which indicates the algorithm's average bound.

Questions ?????

1. What is Characteristic of good algorithm?
2. What is Space Complexity?
3. Define Time complexity?
4. Define Big O notation?
5. Define Omega Notation?
6. Define Theta Notation?
7. Which notations is used to define to denote lower bound?
8. What term is used to describe an O(n) algorithm?

REFERENCES

1. DS AND ALGORITHM BY POONAM PONDE , VISION
2. WEBSITE WWW.GOOGLE.COM

Developed by Alka Mhetre

Thank you