Asymptotic Notations

Dr J P Patra
Associate Professor
Computer Science & Engineering
UTD, CSVTU, Bhilai

Outline:

- What is Asymptotic Notation?
- Why Asymptotic Notation is important ?
- Types of Asymptotic Notations
- Properties of Asymptotic Notations
- Advantages and Disadvantages of Asymptotic Notations

What is Asymptotic Notation?

- Asymptotic notation is used to categorize algorithms into complexity classes, such as polynomial time (P), exponential time (EXP), or logarithmic space (L).

Or

- Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

Why Asymptotic Notation is important?

Algorithm Analysis: Asymptotic notation is crucial for analyzing algorithms and understanding their efficiency. This analysis is essential for selecting the most efficient algorithm for a given problem.

Factors which influence Analysis

- Number of steps present in an Algorithm
- Different Operators used in an Algorithm
- Processing Speed of the Machine
- Input sequence provided by the User

- **Algorithm Comparison:** Asymptotic notation provides a standardized way to compare the performance of different algorithms without being influenced by specific hardware or software details.

- **Predicting Performance:** Asymptotic notation allows us to predict how an algorithm will perform as input sizes become very large.

- **Algorithm Design:** Asymptotic analysis helps algorithm designers make informed decisions about trade-offs between efficiency and functionality.

Types of Asymptotic Notations

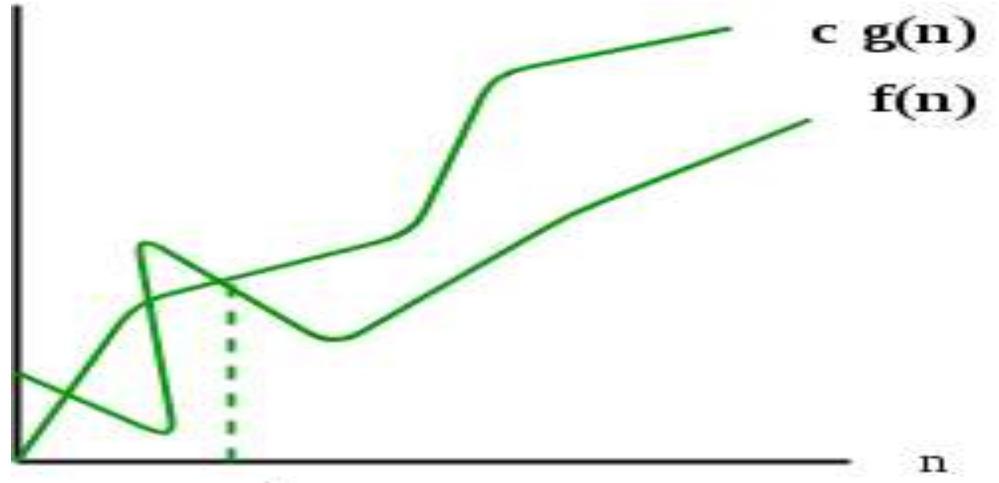
- **Big O Notation (O)**
- **Big Omega Notation (Ω)**
- **Big Theta Notation (Θ)**
- **Little o Notation (o)**
- **Little Omega Notation (ω)**

Big O Notation (O)

Big O notation represents the upper bound of the running time of an algorithm. Thus, it provides the worst case complexity of an algorithm.

Definition :

The function f(n) is Big O of g(n) written as f(n)= O(g(n)), if there exist two positive constants C and n_0 such that f(n) ≤ C.g(n) , for all n ≥ n_0



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

Example: **Prove that f(n)=$2n^2+3n+5 = O(n^2)$**

or

Calculate the Upper Limit of the above Function.

Solution: As the above problem deals with Big O Notation we must have to satisfy the condition $f(n) \leq cg(n)$ for $n \geq n_0$

So, we write $2n^2+3n+5 \leq 2n^2+3n+n$ (When $n \geq 5$, i.e $n_0 = 5$)

$\Rightarrow 2n^2+3n+5 \leq 2n^2+4n$

Further we can write this as

$\Rightarrow 2n^2+4n \le 2n^2+ n^2$ (When $n^2 \ge 4n$, i.e $n \ge 4$, i.e $n_0 = 4$)

$\Rightarrow 2n^2+4n \le 3n^2$

So, we can see that the value of $f(n)= 2n^2+3n+5$ will always be less than or equal to $g(n)= n^2$ when we start the value of n with 4 or 5.

Hence $2n^2+3n+5 = O(n^2)$ is true at n=4 or n=5.

For this problem

$2n^2+3n+5 = O(n^3)$ is true

$2n^2+3n+5 = O(n^4)$ is true

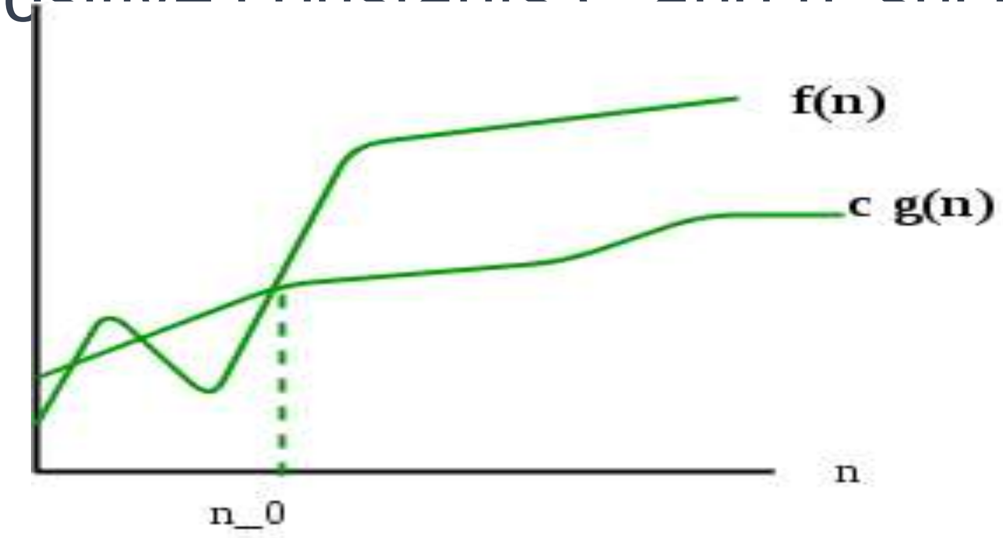
and $2n^2+3n+5 = O(2^n)$ is also true.

Big Omega Notation (Ω)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

Definition :

The function f(n) is Big Omega of g(n) written as f(n)= Ω (g(n)), if there exist two positive constants C and n such that f(n) ≥ C.g(n) ,for all n ≥ n_0



f(n) = Omega(g(n))

Example:

(i) $f(n)=2n^2+3n+5 = \Omega(n^2)$ is True

(ii) $f(n)=2n^2+3n+5 = \Omega(n)$ is True

(iii) $f(n)=2n^2+3n+5 = \Omega(\sqrt{n})$ is True

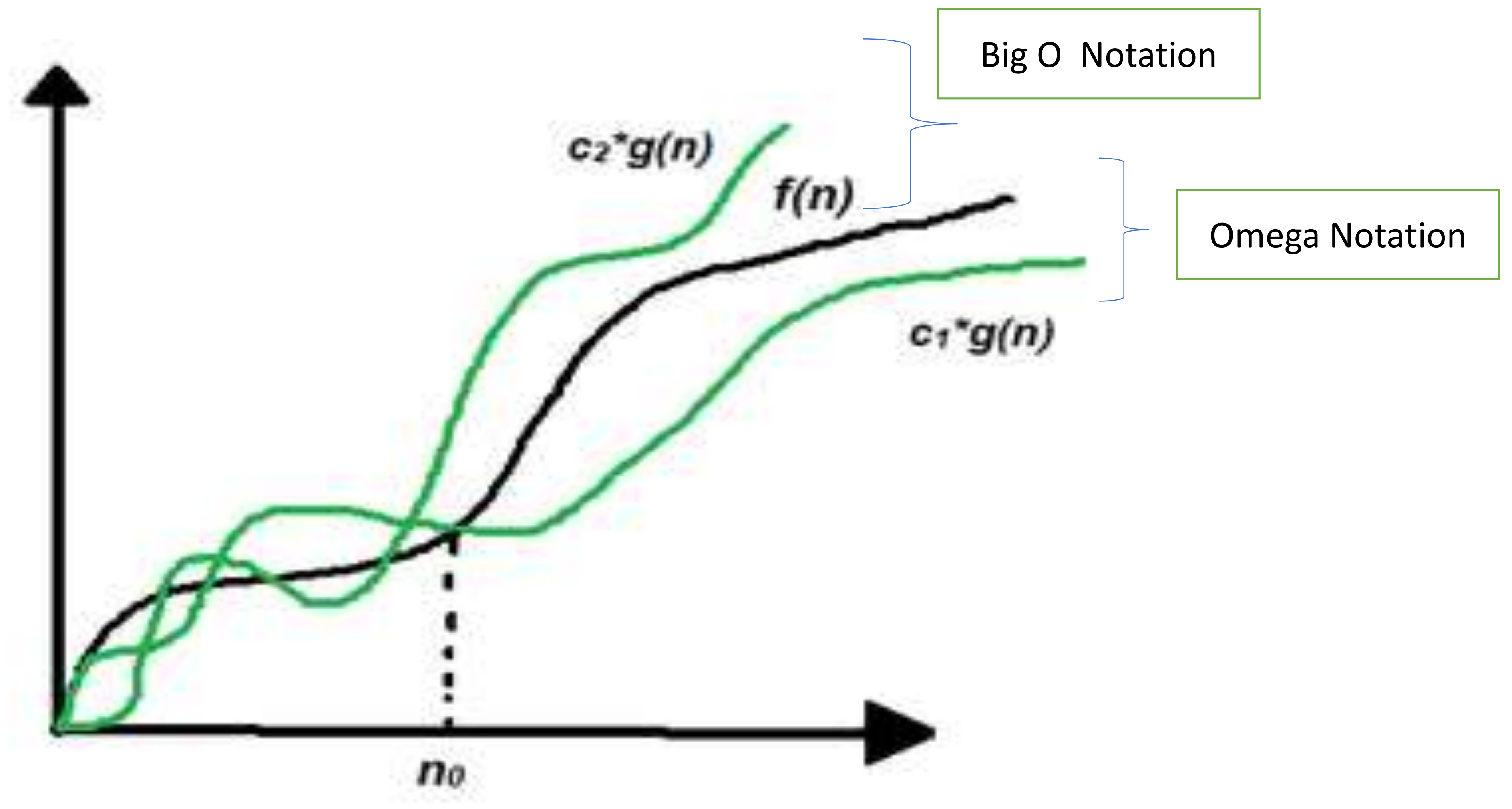
(iv) $f(n)=2n^2+3n+5 = \Omega(\log n)$ is True

Big Theta Notation (Θ)

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

Definition :

The function f(n) is Big Theta of g(n) written as $f(n) = \Theta(g(n))$, if there exist three positive constants C_1, C_2 and n_0 such that $C_1 * g(n) \leq f(n) \leq C_2 * g(n)$, for all $n \geq n_0$


Big O Notation
Omega Notation
$c_2 * g(n)$
$f(n)$
$c_1 * g(n)$
n_0

Example:

(i) $f(n)=2n+5 = \Theta(n)$ is True

then

(ii) $f(n)=2n+5 = \Theta(n^2)$ will be False

also

(iii) $f(n)=2n+5 = \Theta(\sqrt{n})$ will be False

Properties of Asymptotic Notations

General Properties:

If f(n) is O(g(n)) then a*f(n) is also O(g(n)), where a is a constant.

Example:

$f(n) = 2n^2+5$ is $O(n^2)$

then, $7*f(n) = 7(2n^2+5) = 14n^2+35$ is also $O(n^2)$.

Note: Similarly, this property satisfies both Θ and Ω notation.

Transitive Properties:

If f(n) is O(g(n)) and g(n) is O(h(n)) then f(n) = O(h(n)).

Example:

If f(n) = n, g(n) = n^2 and h(n)=n^3
Here n is O(n^2) and n^2 is O(n^3) then, n is O(n^3)

Similarly, this property satisfies both Θ and Ω notation.

Symmetric Properties:

If $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$.

Example:

If $f(n) = n^2$ and $g(n) = n^2$

then, $f(n) = \Theta(n^2)$ and $g(n) = \Theta(n^2)$

This property only satisfies for Θ notation.

Transpose Symmetric Properties:

If f(n) is O(g(n)) then g(n) is Ω (f(n)).

Example:

If(n) = n , g(n) = n^2

then n is O(n^2) and n^2 is Ω (n)

This property only satisfies O and Ω notations.

Advantages of Asymptotic Notations

- Asymptotic analysis provides a high-level understanding of how an algorithm performs with respect to input size.
- It is a useful tool for comparing the efficiency of different algorithms and selecting the best one for a specific problem.
- It helps in predicting how an algorithm will perform on larger input sizes, which is essential for real-world applications.
- Asymptotic analysis is relatively easy to perform and requires only basic mathematical skills.

Disadvantages of Asymptotic Notations

- Asymptotic analysis does not provide an accurate running time or space usage of an algorithm.
- It assumes that the input size is the only factor that affects an algorithm’s performance, which is not always the case in practice.
- Asymptotic analysis can sometimes be misleading, as two algorithms with the same asymptotic complexity may have different actual running times or space usage.
- It is not always straightforward to determine the best asymptotic complexity for an algorithm, as there may be trade-offs between time and space complexity.

Thank you!