# Asymptotic Analysis

KARNAVATI UNIVERSITY | UIT

In Asymptotic Analysis, we evaluate the **performance of an algorithm** in terms of input size (we don't measure the actual running time).

We calculate, order of growth of time taken (or space) by an algorithm in terms of input size.

For example linear search grows linearly and Binary Search grows logarithmically in terms of input size.

The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine-specific constants and don't require algorithms to be implemented and time taken by programs to be compared.

# Asymptotic Notations

KARNAVATI UNIVERSITY | UIT

- Asymptotic Notations are mathematical tools used to analyze the performance of algorithms by understanding how their efficiency changes as the input size grows.
- These notations provide a concise way to express the behavior of an algorithm's time or space complexity as the input size approaches infinity.
- Rather than comparing algorithms directly, asymptotic analysis focuses on understanding the relative growth rates of algorithms' complexities.
- Asymptotic analysis allows for the comparison of algorithms' space and time complexities by examining their performance characteristics as the input size varies.

# Types of Asymptotic Notations

KARNAVATI UNIVERSITY | UIT

1. **Big-O Notation (O-notation)**
2. **Omega Notation (Ω-notation)**
3. **Theta Notation (Θ-notation)**

# Types of Asymptotic Notations

KARNAVATI UNIVERSITY | UIT

1. **Big-O Notation (O-notation)**

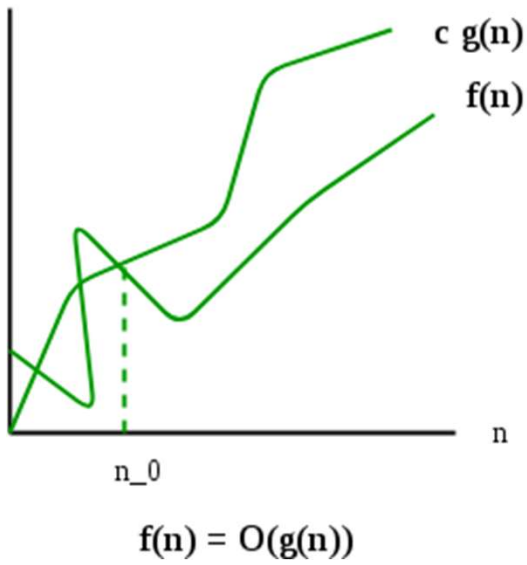
- Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.
- It is the most widely used notation for Asymptotic analysis.
- It specifies the upper bound of a function.
- The maximum time required by an algorithm or the worst-case time complexity.
- It returns the highest possible output value(big-O) for a given input.
- Big-O(Worst Case) It is defined as the condition that allows an algorithm to complete statement execution in the longest amount of time possible.

# Big-O Notation (O-notation)

KARNAVATI UNIVERSITY | UIT

- If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist a positive constant C and n0 such that, $0 \leq f(n) \leq cg(n)$ for all $n \geq n0$
- **It returns the highest possible output value (big-O)for a given input.**

**The execution time serves as an upper bound on the algorithm's time complexity.**




**Mathematical Representation of Big-O Notation:**

$O(g(n)) = \{ f(n)$: there exist positive constants c and n0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n0 \}$

# Omega Notation (Ω-notation)

KARNAVATI UNIVERSITY | UIT

Omega notation represents the lower bound of the running time of an algorithm.

Thus, it provides the best case complexity of an algorithm.

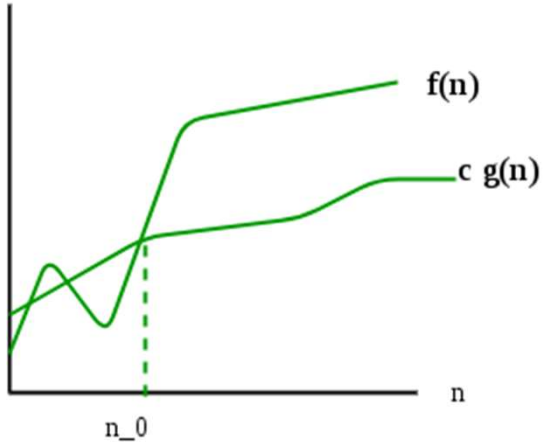
The execution time serves as a lower bound on the algorithm's time complexity.

It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.

# Omega Notation (Ω-notation)

KARNAVATI UNIVERSITY | UIT

- Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c*g(n) ≤ f(n) for all n ≥ n0
- **Mathematical Representation of Omega notation :**
- *Ω(g(n)) = { f(n): there exist positive constants c and n0 such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }*




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

# Theta Notation (Θ-notation)

KARNAVATI UNIVERSITY | UIT

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.

# Theta Notation (Θ-notation)

KARNAVATI UNIVERSITY | UIT

- Let g and f be the function from the set of natural numbers to itself. The function f is said to be Θ(g), if there are constants c1, c2 > 0 and a natural number n0 such that c1* g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0
- **Mathematical Representation of Theta Notation :**
- *Θ (g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0}*
- ***Note:** Θ(g) is a set*