Shri Vile Parle Kelavani Mandal's

Institute Of Technology, Dhule

Department of Information Technology

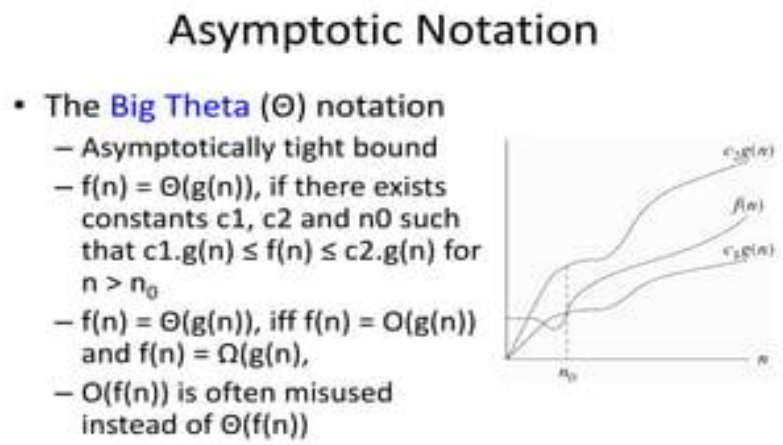
Design and Analysis of Algorithm lab

Sr. No.	Student Name	Roll No	PRN	Class
01	Sarthak Ravindra Bhamare	06	2254491246006	Second year
02	Parth Hemant Borse	07	2254491246007	Second year
03	Arpita Vinayak Chandratre	08	2254491246008	Second year
04	Bhairavi Pradip Chaudhari	09	2254491246009	Second year
05	Kalpesh Chaudhari	10	2254491246010	Second year

1] Explain asymptotic notations: O,Ω and θ.

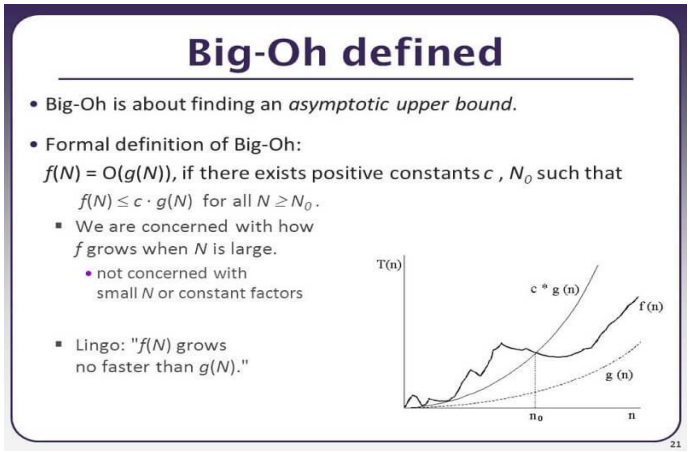
Solution:-

Asymptotic notation is a way to describe the behavior of functions as their input size approaches infinity. It's commonly used in computer science and mathematics to analyze the efficiency of algorithms and to express how the runtime or space requirements of an algorithm grow as the size of the input grows.

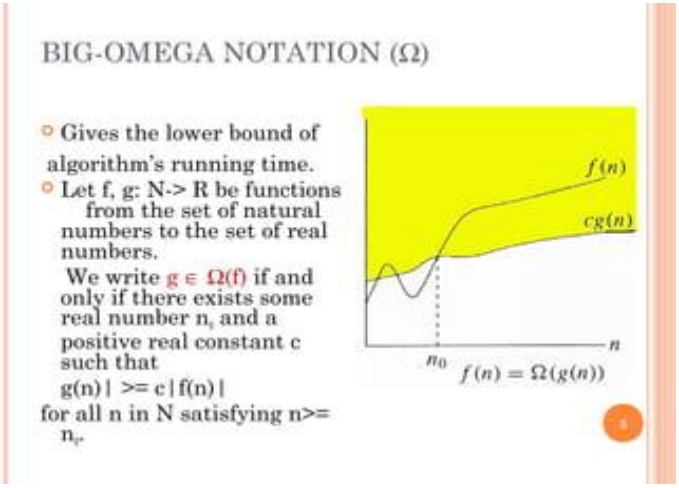


There are three main types of asymptotic notation:

Big O notation (O): This notation represents the upper bound of an algorithm's growth rate. It describes the worst-case scenario of an algorithm's time complexity. For example, if an algorithm has a time complexity of O(n^2), it means that the algorithm's runtime grows no faster than the square of the input size (n), when n is sufficiently large.



Omega notation (Ω): This represents the lower bound of an algorithm's growth rate. It describes the best-case scenario of an algorithm's time complexity. For example, if an algorithm has a time complexity of Ω(n), it means that the algorithm's runtime grows at least as fast as the input size (n), when n is sufficiently large.



Theta notation (θ): This represents both the upper and lower bounds of an algorithm's growth rate. It describes the average-case scenario of an algorithm's time complexity. For example, if an algorithm has a time complexity of θ(n), it means that the algorithm's runtime grows at the same rate as the input size (n), when n is sufficiently large.

These notations help us compare the efficiency of different algorithms and make predictions about how they will perform as the size of the input increases.

Q 3] Solve Following Recurrence Relations Using Master 'S Method.

Some theory :

- **Recurrence:** recurrence relation is a mathematical expression that define Sequence in term of that previous term.
- It is often used to model the time complexity of recursive algorithm.

Master's theorem:

- Using master method we can find recurrence relation time complexity.
- Substitution method is applicable for all problems case but master theorem is applicable for specific problem.
- Substitution method is slower, but master method is very faster.

T(n) = aT(n/b) + f(n)

a≥1 & b>1

then only problem solution finds using master theorem.

Solution of master theorem is T(n) = $n^{\log_b a}$.[u(n)]

Value of u(n) depends upon h(n).

h(n) = f(n)/ $n^{\log_b a}$

based on h(n) value, u(n) value find using following table.

If f(n)	U(n)
If n^r, r>0	O(n^r)
n^r,r<0	O(1)
$(\log n)i$ i≥0	$(\log_2 n)\, i + 1$ /i+1

Example 1] T(n) = 8T(n/2) + n^2

Solution : a=8, b=2 , f(n) = n^2

Solution of master theorem is

T(n) = $n^{\log_b a}$. [u(n)]

= $n^{\log_2 8}$. [u(n)]

T(n)= n^3 . [u(n)] ----1

H(n) = f(n) /$n^{\log_b a}$

= n^2 / $n^{\log_2 8}$

= n^2 /n^3

= 1 / n

H(n) = n^{-1}

[u(n)] = O(1), r<0

Equation 1 become

T(n) = n^3.1

T(n)= O(n^3)

Example 2] T(n) = 9T(n/3) + n^3

Solution: a= 9, b= 3, f(n)= n^3

Master theorem solution is

T(n) = $n^{\log_b a}$. [u(n)]

= $n^{\log_3 9}$. $[u(n)]$

T(n) = n^2 [u(n)] -----1

H(n)= f(n) /$n^{\log_b a}$\

H(n)= n^3 / n^2

H(n)= n

U(n)= O(n)

Equation 1 become

T(n)= n^2 . n

T(n)= n^3

T(n)= O (n^3)

Q 2) Solve following recurrence relation using substitution method. $T(1) = 1 \quad T(n) = 4T(n / 3) + n^2$.

Some Theory :

- **Recurrence:** recurrence relation is a mathematical expression that define Sequence in term of that previous term.
- It is often used to model the time complexity of recursive algorithm.

Substitution Method:

The substitution method is a technique used to solve recurrence relations by "guessing" a solution and then using mathematical induction to prove its correctness. Here's how it works.

1. Guess a Solution Form: Based on the form of the recurrence relation, make an educated guess about the form of the solution. This guess usually involves some constants or functions that need to be determined.

2. Prove by Induction: Once you've guessed the form of the solution, you need to prove its correctness using mathematical induction. This involves two steps:

- **Base Case:** Prove that the solution holds for the initial conditions of the recurrence relation (typically for the smallest values of n).

- **Inductive Step:** Assume that the solution holds for some arbitrary value of n (the induction hypothesis) and then use this assumption to prove that it also holds for the next value of n.

3. Solve for Constants: After proving the correctness of the guessed solution, solve for any constants or functions involved in the solution using the initial conditions provided in the recurrence relation.

4. Write Down the Final Solution: Once you have determined the constants or functions, write down the closed-form expression for the sequence.

Solution of Substitution Method $T(n) = 4T(n /3) + n^2$:-

$$T(n) = 4T(n /3) + n^2 \quad \text{----------1)}$$

$$T(n/3) = 4T(n /3^2) + (n/3)^2 \quad \text{----------2)}$$

$$T(n/3^2) = 4T(n /3^3) + (n/3^2)^2 \quad \text{----------3)}$$

$$T(n/3^3) = 4T(n/3^4) + (n/3^3)^2 \quad \text{----------4)}$$

Eq^n 2) put in eq^n 1)

$$T(n) = 4T(\underline{n/3}) + n^2$$

$$T(n) = 4\,[\,4T(n/3^2) + (n/3)^2\,] + n^2$$

$$T(n) = 4^2T(n/3^2) + 4(n/3)^2 + n^2$$

Put in eq^n 3)

$$T(n) = 4^2\,[4T(n/3^3) + (n/3^2)^2\,] + 4(n/3)^2 + n^2$$

$$T(n) = 4^3T(n/3^3) + 4^2(n/3^2)^2 + 4(n/3)^2 + n^2$$

Put in eq^n 4)

$$T(n) = 4^3\,[4T(n/3^4) + (n/3^3)^2\,] + 4^2(n/3^2)^2 + 4(n/3)^2 + n^2$$

$$T(n) = 4^4T(n/3^4) + 4^3(n/3^3)^2 + 4^2(n/3^2)^2 + 4(n/3)^2 + n^2$$