NAME: KAVIN.A

REG NO: 192321003

ASSIGNMENT-2

SUBJECT: DESIGN AND ANALYSIS OF ALGORITHM

SUBJECTCODE: CSA0088

1.] Explain Asymtopic Notation with details

* The goal of analysis of an algorithm is to compare algorithm in running time & also memory mangement.

* Expressing the complexity in terms of its relationship to know function, this type analysis called asymptotic analysis

* Asymptotic Notation the mathematical way of representing the time complexity.

* The Notation we use to describe the asymptotic running time of an algorithm are defined in terms of function whose domains are set of natural numbers.

There are three types of Notations.

* Big oh (O) Notation
* Big omega (Ω) Notation
* Theta (Θ) notation

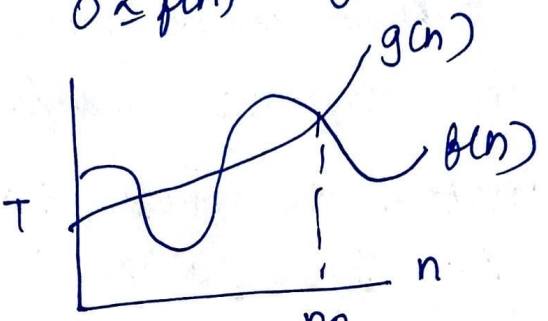
Big oh (O) notation:

Asymptotic less than. This notation mainly represent upper bound of algorithm run time. Big Oh notation is useful to calculate the maximum amount of time execution.

Formula:

$f(n) <= cg(n)$

$O(g(n)) = \{ f(n)$: there exist positive constants $0 \leq f(n) < cg(n)$ for all $n \geq n_0 \}$



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

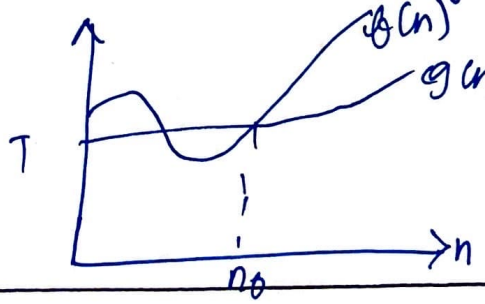
omega (Ω) notation:

Asymptotic notation "greater than" it represent lower bound of algorithm run time by using omega notation we can calculate minimum amount of time

Formula:

$f(n) = \gamma c\, g(n)$

$\Omega(g(n)) = \{ f(n)$: exist positive constant $0 \leq (g(n) < f : n > n_0 \}$



Theta (Θ) notation:

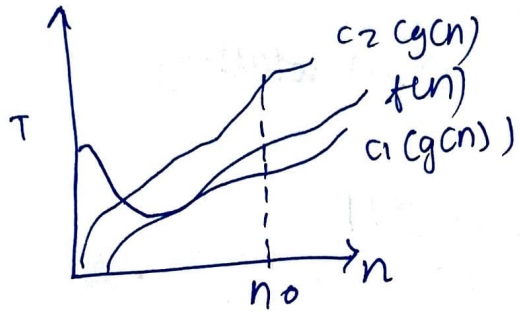
Asymptotic "Equality", it represent average bound of algorithm running time, By using theta notation we can calculate average amount of time. So it is called time complexity.

Formula:

$c_1 g(n) \leq f(n) = c_2 g(n)$

$\Theta(g(n)) = f(n)$ there exist positive constant c_1, c_2

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$



2.) Master theorem problem:

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

$T(n) = 3T(n/4) + n$ using Master theorm

$a = 3, b = 4$

$4(n) = n$

$\Rightarrow n \log_4 3$

$f(n) < n \log_4 3$

$= O(n)$

Hence proved

3) $f(n) = 4n^3 + 2n^2 + n$

$g(n) = n^3$

PT $f(n)$ is $O(g(n))$

$n = 0$:

$f(0) = 4(0)^3 + 2(0)^2 + 0$

$g(n) = 0^3$

$0 = 0$

$O(n)$ notation //

$n = 1$

$f(1) = 4(1)^3 + 2(1)^2 + 1$

$g(n) = 1^3$

$7 < 1$ $O(n)$ notation //

$n = 2$

$f(2) = 4(2)^2 + 2(2)^2 + 1$

$g(2) = 2^3$

$33 < 3$

$O(n)$ notation //

$n = 3$

$f(3) = 4(3)^3 + 2(3)^2 + 1$

$g(3) = 3^3$

$163 < 27$ $O(n)$ notation //

(i) Apply the solution method to solve the recurrsive $T(n) = 2T(n/2) + n$ iteration method.

$T(n) = 2T(n/2) + n$

$T(n) = n/2$

$T(n) = 2(2T(n/4) + n/2) + n \rightarrow ①$

$T(n) = 2(2T(n/8) + n/4) + n \rightarrow ②$

$T(n) = 2(2T(n/16) + n/8) + n \rightarrow ③$

$n = 4T(n/4 + 2n) + 1$

$n = 8T(n/8) + 3n + 1$

$n = 8(n/12 + 4n) + 1$

k times

$T(n) = 2^k T(n/2^k) + kn$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

$T(n) = 2^{\log_2 n} + n(\log_2 n) + \log_2 n$

$n(1)$

$= nT(1) + (1)\log 2^1$

$= O(n \log n)$

linear

proved //

5) $h(n) = 5n^4 + 3n^3 + n$ P.T $h(n)$ is $n(n^4)$ Explain the significance of constant factors in your proof.

$n = 0$

$h(0) = 5(0)^4 + 3(0)^3 + 0$

$= 0$

$n(0) \neq 4$

$= 0^c$

$n = 1$

$h(1) = 5(1)^4 + 3(1)^3 + 1$

$= 9$

$h(1) = 9 > 4$

$n = 1$

$n(1) = n^4$

$= 4$

$= 4$

condition satisfy

$n = 2$

$h(2) = 5(2)^4 + 3(2)^3 + 2$

$= 105$

$n(2) = 2^4$

$= 16$

$n(2) = 105 > 16$

condition satisfy

$n = 3$

$n(3) = 3^4$

$= 81$

condition satisfy

$n = 3$

$h(3) = 5(3)^4 + 3(3)^3 + 3$

$= 459$

$h(3) = 459 > 81$

Explain Master theorem in detail.

The Master theorem is a tool used to solve recurrence relation that arise in the analysis of divide & conquer algorithm

Theorem provides a systematic way of solving recurrence relations

It's used to determine running time of algorithm in terms of asymptotic notations

$$T(n) = a(T)(n/b) + f(n)$$

$$f(n) = \theta(n^k \log^p n)$$

case 1:
If $\log_b a > k$ then $\theta(n \log^a b)$
$= T(n) = \theta(n \log b^a)$

case 2:
If $\log_b a = k$ then $t(n) = \theta(n \log b^x, \log n)$

case 3:
$f(n) > n \log_b a$ then $T(n) = \theta(F(n))$

where,
n = size of input
a = no. of subproblem in the recursion
n/b = size of each problem
$f(n)$ = cost of work done outside the recursive call.