ASSIGNMENT-2

3/08/2024

M. Satvi Reddy
192311038
CSA-0688 - DAA

1. Explain the master theorem with details.

Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

where $a \geq 1$ and $b > 1$ are constants, and $f(n)$ is an asymptotically positive function. The theorem provides a straightforward way to analyze the time complexity of divide and conquer algorithms, where the problem is divided into a subproblems, each of size $\frac{n}{b}$, and combined in $f(n)$ time.

The ~~masther~~ master theorem applies to recurrences of this form and categorizes their solutions into three cases based on the growth rate of $f(n)$ relative to $n^{\log_b a}$.

Here's a detailed explanation of the theorem and its cases:

case 1: $f(n) < \log_b a$

$T(n) = \theta(n^{\log_b a})$

case 2: $f(n) = n \log_b a$

$T(n) = \theta(n \log_b a \log^k n)$

case 3: $f(n) > n \log_b a$

$T(n) = \theta(f(n))$

4. h/n

6 n

Given the recurrence relation $T(n) = 3T(n/4) + n$, analyze its time complexity using the Master theorem.

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

$a = 3 \quad b = 4 \quad f(n) = n$

$n^{\log_b a} = n^{\log_4 3}$

$f(n) < n^{\log_b a}$

$\Rightarrow T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_4 3})$

For the functions $f(n) = 4n^3 + 2n^2 + n$ and $g(n) = n^3$. Provide a detailed analysis prove that $f(n)$ is $\theta(g(n))$.

$\Rightarrow n = 0$

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

$g(0) = 0^3$

$0 = 0$

$o(n)$

$n = 3$

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

$g(3) = 3^3$

$\Rightarrow \Omega(n)$

$n = 1$

$f(1) = 4(1)^3 + 2(1)^2 + 1 = 7$

$g(n) = 1^3$

$7 > 1$

$\Omega(n)$

$n = 2$

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

$g(2) = 2^3$

$\Rightarrow \Omega(n)$

4. $h(n) = 5n^4 + 3n^3 + n$. prove $h(n)$ is $O(n^4)$

$n = 0$

$\Rightarrow h(0) = 5(0)^4 + 2(0)^3 + 0 = 0$

$h(0) = 0^4 = 0$

$n = 1$

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

$n = 1$

$h(1) = n^4 = 4$

$n = 2$

$h(2) = 5(2)^4 + 3(2) + 2 = 105$

$h(2) = 2^4 = 16$

$105 > 16$

$n = 3$

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

$= 489$

$h(3) = 81$

$489 > 81$

5. Asymtoti notation.

$\Rightarrow$ Mathematical tool used to represent time complexity.

1) Big o(o) notation

2) Big omega(Ω) notation

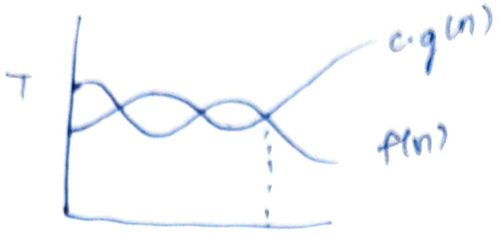
3) Theta (θ) notation

Big Oh(O)

- upper bond of algorithm.

- used to calculator max amount of time.

$f(n) = (g(n))$.

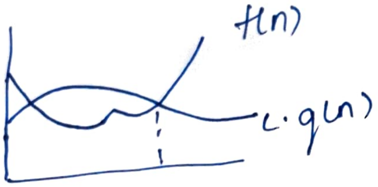


$f(n) = a(g(n))$

Omega (Ω)

- lower bound of algorithm
- used to calculate min amount of time.

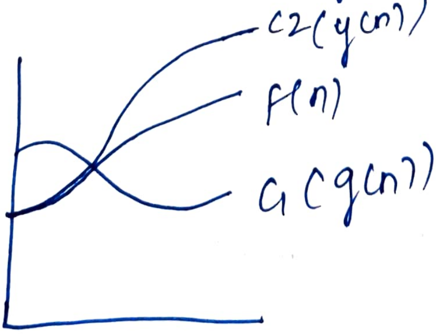
$f(n) >= c.g(n)$



theta (θ)

- average level of algorithm.
- calculate average amount of the

$T\, g(n) <= f(n) <= c_2\, g(n)$



Substitution method to solve.

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

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

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

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

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

$$\vdots$$

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

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

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

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

$$n/2k = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

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

$$= n T(1) + (1) \log_2$$

$$= O(n \log n)$$

hence proved.