A. C. Patil College of Engineering Kharghar Navi-Mumbai Maharashtra

Name : Priyush Bhimrao Khobragade

Roll NO: 52
PRN NO :211112018

Subject : Analysis of Algorithms (AOA)
Assignment-01

[Analysis of Algorithm (AOA)]

Assignment No-01

Q. a) Prove that (i) if $f(n) = n^2 + 50n$ then $f(n) = O(n)^2$

1) → Ans :- By Big oh defination

~~f(n) is off~~ $O(n^2)$ · if $f(n) \leq cn^2$ for some $n \geq n_0$. Let us check this condition:

if $n^2 + 50n \leq cn^2$ then $1 + \frac{50}{n} \leq c$. Therfore, Big oh defination holds for $n \geq n_0 = 1$ & $c \geq 51$. larger value of n_0 result in smaller factors & but in any case above statement is valid.

b) If $f(n) = 15n^3 + n^2 + 4$ then $f(n) \neq O(n^2)$.

Ans - By Big-oh defination,

$f(n)$ is $O(n^2)$ if $f(n) \leq cn^2$ for some $n \geq n_0$. let us check this statement condition:

$15n + 1 + \frac{4}{n^2} \leq c$ ∴ the big-oh condition can't hold (the left side of latter inequality is growing infinetely so that there is no such constant factor c).

Q. 3) $T(n) = 3T(n-T)$ for $n>1$ $T(1) = 4$

(c) $T(n) = 3T(n-1)$ — (i)

$T(n-1) = 3T((n-1)-1)$

$T(n-2) = 3T(n-2)$ — (ii)

$T(n-2) = 3T(n-3)$ — (iii)

putting eqn (ii) in (i)

$T(n) = 3(3T(n-2))$ — (iv)

putting eqn in (iv)

$T(n) = 3((3(3T(n-3)))$

$T(n) = 3^3 T(n-3)$

$= 3^{n-1} T(n-(n-1))$

$= 3^{n-1} T(1)$

$\therefore T(1) = 4$

$\therefore T(n) = 3^{n-1} \cdot 4$

$\boxed{T(n) \approx O(3^n)}$

Q.2)

a) $T(n) = 2T(n-1) + 1$ for $n > 1$ $T(1) = 1$

$T(n) = 2T(n-1) + 1$ — ①

$T(n-1) = 2T(n-2) + 1$ — ②

$T(n-2) = 2T(n-3) + 1$ — ③

put ② & put ③

$T(n) = 2[2T(n-2) + 1] + 1$

Q-5

$T(n) = 2 \cdot 2[T(n-2) + 1 + 1$

$T(n) = 2^2 T(n-2) + + 1$ — Ⓐ

put (ii) in Ⓐ

$T(n) = 2^3 T(n-3) + 1 + + +$

$T(n) = 2^i T(n-i) + i$

How, we need to substitute is $(n-i)$ here $n > 1$; $T(1) = 1$ $T(n) = 1$

$n - i = 1 \Rightarrow i = (n-1)$

$\Rightarrow (2^{n-1}) \cdot T(1) + (n-1)$. $\because (T(1) = 1)$

$\therefore (2^{n-1}) + (n-1)$

$O(2^n)$.

Q.2) Algorithm Fibbonacci (n)

if (n > 1)

return n

else

$F_1 = 0$

$F_2 = 2$

for $i = 2$ to n do

$F = f_1 + f_2$

$f_1 = F_2$

$F_2 = F$

return 1

Soln

a Basic operation is executed n time.

Q.5) Find order of growth for solution of following recurrence relation using master Method.

a) $T(n) = 4T(n/2) + n$ $\qquad T(n) = 1$ —

Soln $\quad T(n) = 4T\left[\frac{n}{b}\right] + n \qquad (n^k \times \log^p n)$

where, $4 > 1$, $b > 1$, $k \geq 0$ and p is real no.

Case 1: if $a > b^k$ then $T(n) = \theta(n^{\log_b^a})$

$T(n) = 4T\left[\frac{n}{2}\right] + n$

Comparing this eqn with given eqn we get,

$a = 4,\ b = 2,\ k = 1$ and $p = 0$

$4 > 2^1$ Hence case 1 is true. So,

$T(n) = \Theta(n \log_b^a) = \Theta(n \log_b^a)$

$= \cancel{\Theta(n^2)}$

b) $T(n) = 4T\left[\frac{n}{2}\right] + n^2 \qquad T(1) = 1$

Sol $\rightarrow$ Reccurrence relation $4T\left[\frac{n}{2}\right] + n^2$

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

$a = 4$ & $b = 2$

$\therefore a \geq 1$ & $b > 1$

$n \log_b^a = n \log_2^4 = n^2$

~~$f(n) = \Theta(n \log_b^a)$~~

By using master therom

$T(n) = \Theta(n \log_b^a p . \log(n)) = \Theta(n^2 \log n)$

c) $T(n) = 4T\left[\frac{n}{2}\right] + n^3 \qquad T(1) = 1$

Soln $\rightarrow$ Camparing with $T(n) = aT\left(\frac{n}{b}\right) + fn$

$a = 4, b = 2; f(n) = n^3$

Sol^n is $T(n) = n \log_6^9 \; [h(n)]$

$$L(n) = \frac{f(n)}{n \log_6^9} = \frac{n^3}{n \log^3} = n$$

So, relation in $h(n)$ & $u(n)$ is

n^r & $r > 0$ is (n^r) so,

$$T(n) = n \log_2^h \; [u(n)]$$
$$= n^2 . o[(n)]$$

$$T(n) = \Theta(n^3)$$

Q.6) Define Asymptotic Notation Big-O, Omega & Theta.

- Asymptotic Notation :-

it is mathmatical related Notations used to derive the running time of an algorithm, when input lines towards particular value of limiting value.

But when input array is in reverse condition algorithm taking maximum time to sort element i.e worst case.

when input array is sorted number not in reverse order order, it takes average time. These duration are denoted using asymptotic

Types of Asymptotic

- Big-O notation
- omega notation
- Theta notation

* **Big-O Notation** :- " It is represents upper bound of running time of an algorithm. Thus it gives worst case computing of an algorithm."

O(g(n)) f(n) where exist positive constant c such that 0 ≤ f(n) ≤ g(n) for all n. no.

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

function f(n) belong to set O(g(n))
C is positive constant
for all value of n, running time of an algorithm doesn't cross by O(g(n)).

* **Omega Notation (Ω)** :- " It is represent lower bound of algorithm, thus it gives best case computing of an algorithm.

Ω (g(n)) = f(n) where exist positive constant 'c'. n' such that 0 ≤ g(n) ≤ f(n) for all n ≥ n.
f(n) belong to set Ω(g(n)), c positive constant such that above c g(n) for sufficient large n.

* **Theta Notation (θ)** - " Encloses function above & below. since it represents upper & lower bound of running time of an algorithm. it is used for analyzing average case computing of an algorithm.
θ(g(n)) = f(n) where exist positive constant c_1, c_2 & no such that 0 ≤ (g(n) ≤ f(n) ≤ c2.g(n) for n ≥ no.