Name : P. Pradeep

Roll No : 2016114.5

ALGORITHMS (ASSIGNMENT)

1.

(a) Given

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

$(c \geq 0)$

$\Rightarrow$ there exists a constant c such that values of n

$0 \leq f(n) \leq c\, g(n)$ $\forall$ very large ~~numbers~~ values of n

since both $0 \leq f(n)$ and $0 \leq g(n)$

squaring on both sides

$$\Rightarrow (f(n))^2 \leq c^2 (g(n))^2$$

$$\Rightarrow (f(n))^2 \leq k\,(g(n))^2$$

$$\Rightarrow f(n)^2 = O(g(n)^2)$$

(b) To proove or disproove

$$f(n) + g(n) = \Theta(\underbrace{\min\{f(n), g(n)\}}_{x})$$

$$\Rightarrow k_2\, x \leq f(n) + g(n) \leq k_1\, x$$

let $f(n) = n$

$g(n) = n^2$

$\min(f(n), g(n)) = n$

$$\Rightarrow k_2 n \leq n + n^2 \leq k_1 n$$

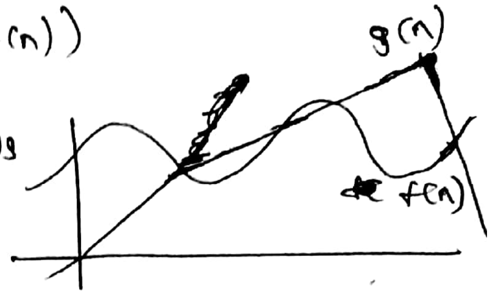
which is false because $n^2 \not\leq kn$

So it is false.

c)

$f(n) \neq \Omega(g(n))$ implies $g(n) = o(f(n))$

$\Rightarrow kg(n) \nless f(n)$ $\Rightarrow$ consider a graph like this



~~N f(n) = o(g(n)) but it does not imply g(n) = o(f(n))~~

The above graph shows that we cannot say $g(n) = o(f(n))$

d) $\min\{f(n), g(n)\} \in O(f(n)+g(n))$

let case -1

$f(n) \leq g(n)$

then

$\min\{f(n), g(n)\} = f(n)$,

we know that $f(n) \leq f(n) + g(n)$

and

$g(n) \leq f(n) + g(n)$

$\Rightarrow \min(f(n), g(n)) \leq f(n) + g(n)$

$\Rightarrow \min(f(n), g(n)) \in O(f(n)+g(n))$

2. To prove $\log(n!) = \Theta(n \log n)$ and $n! = o(n^n)$.

a) To prove $n! = o(n^n)$

$n! = n \cdot n-1 \cdots 1$

we know that $n, n-1, \cdots 1 \leq n$

$\Rightarrow n \cdot n-1 \cdots 1 \leq n \cdot n \cdots n$ times

Scanned by CamScanner

$\Rightarrow \quad n! \le n^n$

$\Rightarrow \quad n! \le k n^n \quad$ k be any constant

$\Rightarrow \quad n! = O(n^n)$

Now to prove $\log(n!) = \Theta(n \log n)$

since $n!$ and n^n are +ve apply log on both sides

$n! \le k n^n$

$\log(n!) \le \log(k n^n)$

$\Rightarrow \quad \log(n!) \le \log k + \log n^n$

$\Rightarrow \quad \log(n!) \le n \log n$

$\Rightarrow \quad \log(n!) = O(n \log n) \quad \text{--- (1)}$

$\cancel{\log(n!) \le \log 1 + \log 2 + \ldots + \log n}$

$n! \ge n/2 \times n/2 \times \ldots \; n/2$ times

$\Rightarrow \quad n! \ge (n/2)^{n/2}$

apply log on both sides

$\log(n!) \ge n/2 \log(n/2)$

$\Rightarrow \log(n!) \ge k n \log(n)$

$\Rightarrow \quad \log(n!) = \Theta(n \log n)$

3. Insertion sort is a stable algorithm

Ex:- 6 $\boxed{4}$ 4 5 step 1

4 6 $\boxed{4}$ 5 step 2

4 4 6 $\boxed{5}$ step 3

4 4 5 6 step 4

merge sort is also a stable algoritm if we give more preference to left array than right half array while merging.

i.e keep '≤'.

Ex:

6 4 4 5

6 4 4 5

$\boxed{4\ 6}$ $\boxed{4\ 5}$

$4 \leq 4$

$\Rightarrow$ $\boxed{4\ \ 4\ 5\ 6}$ $\rightarrow$ final sorted array.

4. To prove $(n+a)^b = \theta(n^b)$

$$(n+a)^b = {}^bC_0 n^b a^0 + \dots + {}^bC_b a^b \quad [b \geq 1, a \geq 0]$$

$$\Rightarrow (n+a)^b = C_0 n^b + C_1 n^{b-1} + \dots + C_n \geq C_0 n^b \quad - (1)$$

we also know that,

$c_0 n^b + c_1 n^{b-1} + \cdots + c_n \leq n^b (c_0 + \cdots + c_n)$

$c_0 n^b + c_1 n^{b-1} + \cdots + c_n \leq n^b k$ -- (2)

from (1) and (2)

$c_0 n^b + c_1 n^{b-1} + \cdots + c_n = \Theta(n^b)$ for $b > 0$, $a > 0$

5.

$T(n) = 2T(\sqrt{n}) + 1$, $T(1) = 1$

T. Prove $T(n) = \Theta(\log n)$

let $n = 2^k \Rightarrow k = \log_2 n$

$T(2^k) = 2T(2^{k/2}) + 1$

let $S(k) = T(2^k)$

$S(k) = 2S(k/2) + 1$

By applying master theorem,

$\log_b^a = 1 > 0$

$\Rightarrow S(k) = \Theta(k) = T(2^k) = T(n)$

$\Rightarrow T(n) = \Theta(k) = \Theta(\log n)$.

6.

a) True

b) False, because $T(n) = \Omega(f(n)) \Rightarrow cf(n) \leq T(n)$ for some constant c, $n \geq n_0$.

c) True, given $T(n) \leq c_1 f(n)$ $n \geq n_1$

$c_2 f(n) \leq T(n)$ $n \geq n_2$

then from above two equations,

$c_2 f(n) \le T(n) \le c_1 f(n)$ $\quad n \ge \max(n_1, n_2)$

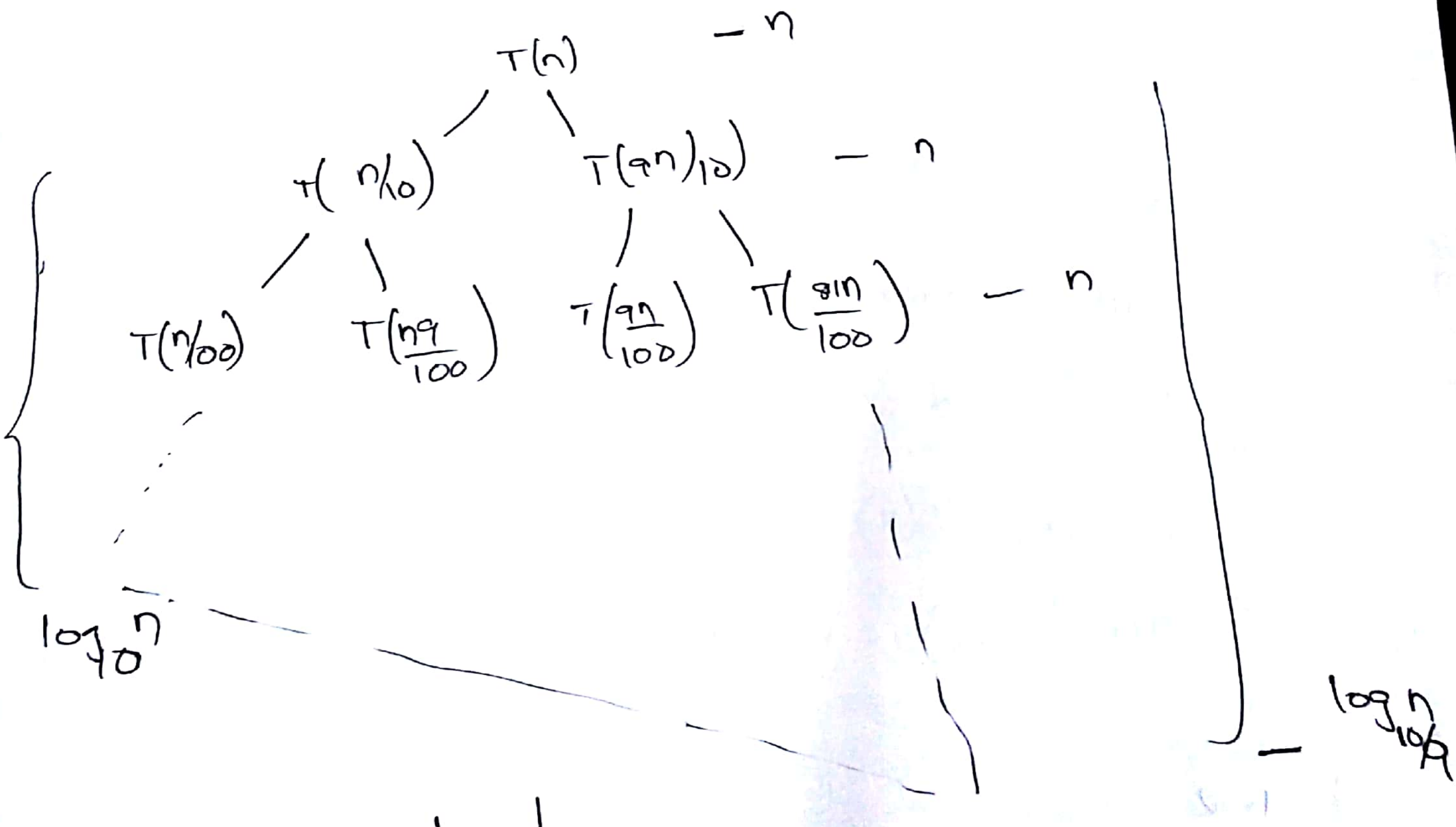
$\Rightarrow T(n) = \Theta(f(n))$

d) False,

because we cannot say $T(n) < cf(n)$ even though $T(n) \le cf(n)$.

7.

$T(n) = T(n/10) + T(9n/10) + n$



for lower band

$T(n) \ge n \log_{10} n$

$\Rightarrow k \log n \le T(n)$

$\Rightarrow T(n) = \Omega(\log n \cdot n)$ --(1)

for upper band

$$T(n) \le n \log_{10/9} n$$

$$\Rightarrow T(n) \le k\, n \log_{10/9} n$$

$$\Rightarrow T(n) = O(n\log n) \quad \text{--- (2)}$$

from (1) and (2)

$\Rightarrow$

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

8. median can be found in $O(\log n)$

let $A_1[1 \text{---} n]$ - m_1

$A_2[1 \text{--} n]$ - m_2

Compare medians of both arrays A_1 and A_2:

if $(m_1 < m_2)$

then median will lie $A_2[0, \text{--} m_2]$ (or)

(or) it will be b/w $A_1[m_1 \text{---} n]$ & last element.

else if $(m_2 < m_1)$.

the median will lie $A_1[0, \text{---} m_1]$ (or)

$A_2[m_2, \text{--} n]$

else

return $\frac{m_1+m_2}{2}$.

v median.

9. ~~$n^{1.004} = O(n\log n)$~~ complexity = $O(\log n)$.

10.

In general merge sort always takes time complexity $O(n \log n)$

But worst case time complexity of insertion sort is $O(n^2)$.

So In general merge sort is better.

a) If the array is almost sorted then "Insertion sort" is better.

Proof

In insertion sort, In every iteration the pointer element will be compared to the one beside it, so if the array is almost sorted then it will be $O(n)$.

9. To show (or) disprove $n^{1.0004} = o(n \log n)$

The above claim is false because we can proove

$$n^{1.0004} = \Omega(n \log n)$$

$$\Rightarrow n^{1.0004} \geq c_1 n \log n$$

$$\Leftarrow n^{1/250} \geq c_1 \log n$$

lets us take $c_1 = 1$, then we have to find some value of n such that $\forall\ n \geq n_0$ the inequality holds

$$\Rightarrow n^{1/250} \geq \log n$$

let $n = e^x \Rightarrow x = \log n$

$$\Rightarrow e^{x/250} \geq x$$

$$1 + \frac{x/250}{1!} + \frac{(x/250)^2}{2!} + - \geq x \quad [\because e^x \text{ expansion}]$$

Since the sum has ∞ terms and x is a very large number let us ~~check~~ consider the inequality ~~for~~ by taking any single term for convinience,

$$\left(\frac{x}{250}\right)^2 \times \frac{1}{2} \geq x$$

$$\Rightarrow x \geq 2(250)^2$$

$$\Rightarrow n \geq e^{2(250)^2} \quad \text{ie} \quad n_0 = e^{2(250)^2}$$

Hence for $\forall\ n \geq n_0$ we have $n^{1.0004} \geq n \log n \Rightarrow n^{1.0004} = \Omega(n \log n)$

10.

b) No, we cannot improve complexity of merge sort no matter if the array is sorted or not because,

while merging two different half arrays even though ~~sorted~~ in sorted array we know that we can join those arrays directly we need to copy those elements into other array which takes o(n).

So time complexity will be always o(nlogn).