SER 501 : Advanced Data Structures & Algorithms

Name: Sai Swaroop Reddy V.

Assignment 1.

Instructor: Ajay Bansal.

Due date: 5th Oct, 11:59PM

Problem ①:

$$T(n) = \begin{cases} 2 & \text{if } n=2 \\ 2T(n/2)+n & n=2^k,\ k>1 \end{cases}$$

is $T(n) = n\lg n$.

Base case:

For $n=2$, $T(2) = 2\lg 2$

$= 2$ (same as base case)

Induction hypothesis:

Assume the statement is true for

some $n = 2^k$

ie $T(2^k) = 2^k \log_2 2^k$

$= k \times 2^k$

Induction step:

We need to prove, the statement is true for $n = 2^{k+1}$

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

from induction hypothesis

$$= 2(k \times 2^k) + 2^{k+1}$$

$$= (2k+2)2^k$$

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

comparing this with $n\log_2 n$

$$T(2^{k+1}) = 2^{k+1}\log_2 2^{k+1}$$

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

$\therefore$ As the base and inductive step both hold true, we have proven that $T(n) = n\log_2 n$ for $n = 2^k$.

Problem 2:

(A)

① $n^3 + 3^n$

Between n^3 and 3^n, the exponential grows faster.

$\therefore n^3 + 3^n = O(3^n)$

② $3n\log(5n)$

Using properties of logarithms, $\log(5n)$ is essentially $\log(n)$ in terms of growth.

$\therefore 3n\log(5n) = O(n\log n)$

③ $100 \times 2^n + 3^n$

Between 2^n and 3^n, 3^n grows faster.

$\therefore 100 \times 2^n + 3^n = O(3^n)$.

(4) $80n\log n + 5n^3 + \sqrt{n}$

Among $80n\log n$, $5n^3$ and $\sqrt{n}$, n^3 grows faster.

$$\therefore 80n\log n + 5n^3 + \sqrt{n} = O(n^3)$$

(5) $1^3 + 2^3 \dots n^3$

Sum of cubes of first 'n' natural numbers $= \left(\frac{n(n+1)}{2}\right)^2$

$$\therefore 1^3 + 2^3 + \dots n^3 = O(n^4)$$

(B) $f(n) = 1 + 2 + 4 + \dots + 2^n$

This is a geometric series with $r = 2$

Sum of 'n' terms in geometric series

$$S_n = \frac{a(1-r^{n+1})}{1-r} = \frac{a(r^{n+1}-1)}{r-1}$$

Here a = first term = 1

$$S_n = \frac{2^{n+1}-1}{2-1} = 2^{n+1}-1$$

$$f(n) = 2^{n+1}-1$$

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

Justification by mathematical induction.

<u>Base case:</u>

When $n=1$

$f(1) = 1$

which is less than $2^1 = 2$

<u>Induction hypothesis:</u>

Assume that the statement is true for some arbitrary $n=k$

i.e $f(k) \leq c \times 2^k$ for some constant 'c'.

Inductive step:

We need to prove that

$$f(k+1) \leq c \times 2^{k+1}$$

Given:

$$f(k+1) = f(k) + 2^{k+1}$$

$$\leq c \times 2^{k} + 2^{k+1}$$

The above holds true for the right choice of c, specifically if $c \geq 2$ the inductive step holds true.

$\therefore$ By mathematical induction, $f(n) = O(2^n)$ is a valid upper bound for $f(n) = 1 + 2 + 4 \cdots + 2^n$.

Problem 3:

① The number of bits required to represent a positive integer 'n' in binary form is essentially the highest power of 2 that can be represented by n. This is equivalent to $\log_2 n + 1$ bits

Example: $n = 8$

$\Rightarrow n = 2^3 \Rightarrow 4$ bits $= 1000$

for a positive integer, it can be represented by $\lfloor \log_2 n \rfloor + 1$ ($\lfloor x \rfloor$ = floor function)

Example: $n = 7 = 3$ bits $= 111$

$\Rightarrow \lfloor \log_2 7 \rfloor + 1 \Rightarrow \lfloor 2.807 \rfloor + 1$

$\Rightarrow 2 + 1$

$= 3$

$\therefore$ In big O notation, it is $O(\log n)$

② The given code prints "hello" and then 'n' reduces by half in each iteration.

<u>Example</u>: if $n = 16$

$16 > 1$, prints "hello", then $n = 8$

$\rightarrow$ $8 > 1$, $n/2 = 4$

$\rightarrow$ $4 > 1$, $n/2 = 2$

$\rightarrow$ $2 > 1$, $n/2 = 1$

So, the number of times it prints is basically the number of bits required to print n minus 1.

$\therefore$ $O(\log n)$ [from ①]

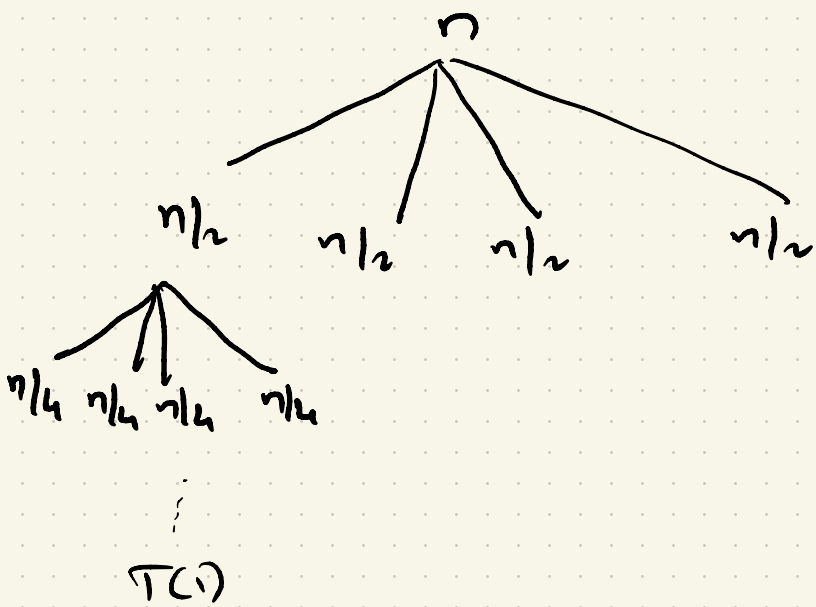
→ for the above example

$n = 16$ no of bits $= 10000 = 5$

$\log_2 16 = \log_2 2^4 = 4 \rightarrow 5-1$

<u>Problem 4</u>:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & n = 2^k,\ k \geq 1 \end{cases}$$



n

$4n/2 = 2n$

$4\left(4\left(\frac{n}{4}\right)\right) = 4n$

No. of leaves $= n^{\log_b a}$

$= n^{\log_2 4} = n^2.$

Comparing $f(n)$ and $n^{\log_b a}$

n and n^2

$T(n) = O(n^2)$

<u>Substitution method.</u>

Let's assume $T(n) \leq cn^2$ for some constant 'c' and for all 'n'.

Base: for $n=1$

$T(1) = c$

$= \Theta(1)$

Now for

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

$$\leq 4c\left(\frac{n}{2}\right)^2 + n$$

$$\leq cn^2 + n$$

The constant 'c' must be chosen such that the extra 'n' is absorbed. This can be done by appropriately choosing a larger value for c.

$\therefore$ By both methods $T(n) = O(n^2)$

<u>Problem</u> 5:

If we let $T(n)$ denote the time to sort an array of size n. using recursive insertion sort.

- The first step

  sorting the sub array $A[1 \ldots n-1]$ will take $T(n-1)$ time

- Second step is to insert $A[n]$ into $A[1 \ldots n-1]$ will take at most $n-1$ comparisons in the worst case

Thus, the recurrence is

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

Base case: $T(1) = 0$

(as it takes no time to sort array of size 1).

- Solving the recurrence:

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

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

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

$$\vdots$$

$$= T(1) + n(n) - (1+2+3 \ldots n)$$

$$= T(1) + n^2 - \frac{n(n+1)}{2}$$

$\therefore \quad O(n^2)$

Problem 6:

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

$a = 2 \quad b = 2 \quad f(n) = n^4$

$\log_b a = \log_2 2 = 1$

comparing $n^{\log_b a}$ and $f(n)$

$\Rightarrow n^1$ and n^4.

$\therefore f(n) = \Omega(n^{\log_b a + \varepsilon})$

$= \Omega(n^{1+\varepsilon})$ with $\varepsilon = 3$

Regularity conditions:

$af(n/b) \leq cf(n)$

$2f(n/2) \leq cn^4$ for $c<1$

$T(n) = \Theta(n^4)$

(b) $T(n) = T(7n/10) + n$

Comparing $n^{\log_b a}$ and $f(n)$

$\Rightarrow n^{\log_{10/7} 1}$ and n

$\Rightarrow n^0$ and n

$\therefore f(n) = \Omega(n^{\varepsilon})$ for $\varepsilon = 1$

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

(c) $T(n) = 16T(n/4) + n^2$

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

$\Rightarrow n^{\log_4 16}$ and n^2

$\Rightarrow n^2$ and n^2

$\therefore T(n) = \Theta(n^{\log_b a} \log(n))$

$= \Theta(n^2 \log(n))$

(d) $T(n) = 2T(n/4) + \sqrt{n}$

$n^{\log_b a}$ and $n^{1/2}$

$n^{\log_4 2}$ and $n^{1/2}$

$n^{1/2}$ and $n^{1/2}$

$\therefore T(n) = \Theta(n^{1/2} \log(n))$

(e) $T(n) = \sqrt{2}\,T(n/2) + \log n$

Master theorem doesn't apply.

(f) $T(n) = 64T(n/8) - n^2 \log(n)$

Master theorem doesnot apply due to negative term

(g) $T(n) = 2T(n/4) + n^{0.51}$

$n^{\log_4 2}$ and $n^{0.51}$

$f(n) = \Omega(n^{\log_4 2 + \varepsilon})$ for $\varepsilon = 0.01$

$T(n) = \Theta(n^{0.5})$

(h) $T(n) = 16T(n/4) + n!$

Master theorem doesn't apply due to factorial term

(i) $T(n) = 0.5T(n/2) + 1/n$

Master theorem doesn't apply due to decreasing term $1/n$

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

Master theorem doesn't apply due to exponential term 2^n