Design and analysis of algorithms.

Sudarsharan K.T.
192324058
CSA-0669.

1) Solve the following recurrence relation

a) $x(n) = x(n-1)+5$ for $n>1$ $x(1)=0$

at $n=1$; $x(1) = 0$ (given)

at $n=2$; $x(2) = x(2-1)+5$

$= x(1)+5$

$= 0+5 = 5$

$x(2) = 5$.

at $n=3$, $x(3) = x(3-1)+5$

$= x(2)+5$

$= 5+5 = 10$

$x(3) = 10$

at $x=4$; $x(4) = x(4-1)+5$

$= x(3)+5$

$= 10+5$.

$x(4) = 15$

$\therefore x(n)$ increases by 5 for each increment of 5 difference $(d) = 5$.

$x(n) = x(1)+(n-1).d$.

Where . $x(1)=0$, $d=5$.

$x(n) = 0+(n-1)5$

$x(n) = 5(n-1)$

Ans : $x(n) = 5(n-1)$.

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

$n=1 : x(1) = 4$.

$n=2 : x(2) = 3x(2-1)$

$= 3x(1)$

$= 3 \times 4 = 12$.

$x(0) = 12$.

$n=3 : x(3) = 3x(3-1)$

$= 3x(2)$

$= 3 \times 12 = 36$

$n=4 ; x(4) = 3x(4-1)$

$= 3x(3)$

$= 3 \times 36$.

$x(4) = 108$.

∴ $x(n)$ obtained by multiplying the pvwious form by 3.

Ratio = 3.

$x(n) = x(1) \cdot v^{n-1}$

here, $x(1) = 4, v = 3$.

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

ans : $x(n) = 4 \times 3^{n-1}$.

Algorithm:

$x(n) = \begin{cases} x(n/2) + n & \text{for } n > 1 \\ x(1) & \end{cases}$

$x(n) = x(n/2) + c \longrightarrow (1)$

$x(n/2) = x(n/4) + c \longrightarrow (2)$

$x(n/4) = x(n/8) + c \longrightarrow (3)$

sub (2) in (1).

$x(n) = x(n/4) + c + c.$

$x(n) = x(n/4) + 2c \longrightarrow (4)$

$= x(n/2^2) + 2c$

sub (3) in (4).

$x(n) = x(n/8) + c + 2c.$

$x(n) = x(n/2^2) + 3c.$

$x(n) = x(n/2^k) + kc.$

$n = 2k ; x(1) = 1.$

$x(n) = x + kc.$

$x(n) = 1 + kc$

$x(n) = 1 + \log n . c$

Time complixity $= O(\log n)$

d). $x(n) = x(n/3) + 1$ for $n > 1$ $x(1) = 1$

(solve for $n = 3k$)

$x(n) = x(n/3) + 1 \longrightarrow (1)$

$x(n/3) = x(n/9) + 1 \longrightarrow (2)$

$x(n/9) = x(n/27) + 1 \longrightarrow (3)$

sub ② in ①,

$x(n) = x(n/27) + 3$ —— ⑤.

$= x(n/3^2) + 3$.

$x(n) = x(n/3^v) + k$.

$x(n) = x(n/3^k) + k$.

$= x(n/n) + k$.

$= x(1) + k$.

$= 1 + k$.

$x(n) = \log n$.

∴ Time complexity $= O(\log n)$.

2) Evaluate following recurrences completely.

i) $T(n) = T(n/2) + 1$ where $n = 2k$ for all $k \geq 0$

$T(n) = T(n/2) + 1 \quad n = 2k$.

sub $n = 2^t$

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

$n = k - 1$

$T(2^{k-1}) = T\left(\frac{2^{k-1}}{2}\right) + 1 = T(2^{k-2}) + 1$

$n = k - 2$

$T(2^{k-2}) = T\left(\frac{2^{k-2}}{2}\right) + 1 = T(2^{k-3}) + 1$

$T(2^1) + T(2^0) + 1$

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

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

ine.

$2^0 = 1$, $T(2^0) = T(1)$

$T(2^k) = 1 + k$.

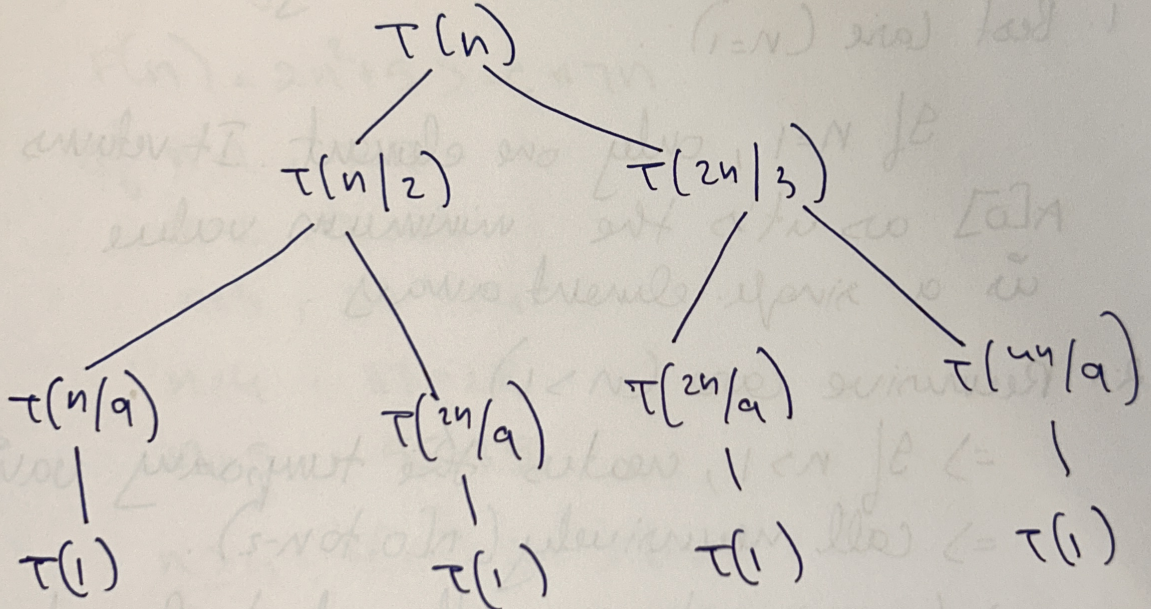
$T(n) = 1 + \log_2 n$

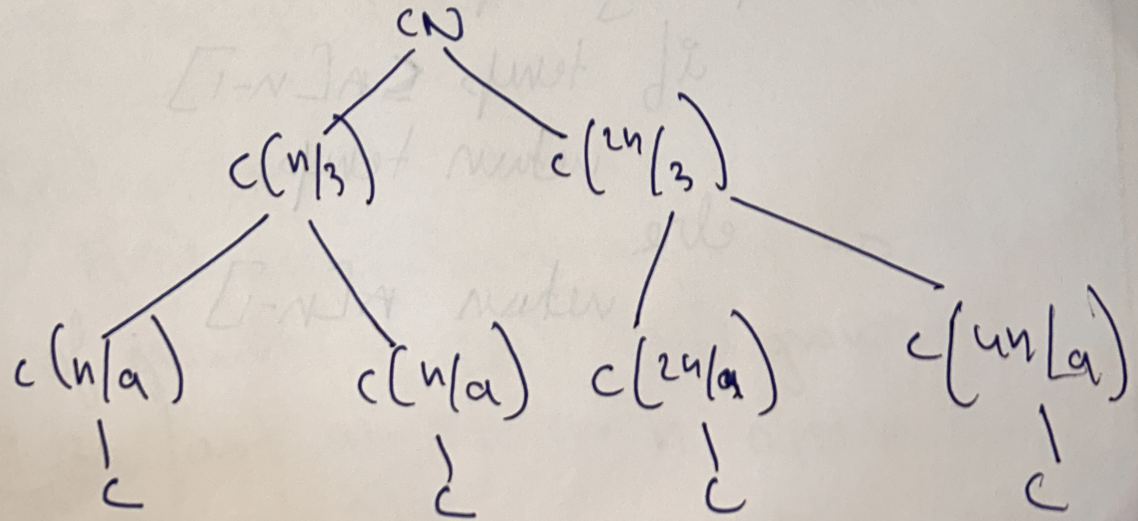
Time complexity $= O(\log n)$

ii) $T(n) = T(n/3) = T(2n/3) + cn$

we use recursion tree method.

$T(n) = T(n/3) + T(2n/3) + cn$.





3) Mini(A[0 ... n-1])

if(n-1) return A[0].

use temp = mini(A(0 ... n-2))

if temp <= A(n-1) return temp.

Return A(n-1).

a) What does this algorithm compute?

This algorithm computes the minimum value in an away A.

1. Best case (n=1).

If n=1, only one element. It returns the A[0] as it's the minimum value is a single element away.

2. Recursive case (n>1).

=> If n>1, creates the temporary variable

=> call recursively (A[0 to n-2).

=> Comparing temp with last element

if temp ≤ A[n-1]

return temp

else

return A[n-1]

$T(1) = C_1$

recursive

case = $T(n) = T(n-1) + C_2$.

final solution.

$T(n) = C_2 * n^2 + (C_1 - C_2)$

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

4) i) $F(n) = 2n^2 + 5$ and $g(n) = 7n$, Use the $\Omega(g(n))$ notation.

$F(n) = 2n^2 + 5 >= C * 7n$.

if $n = 1$, $7 = 7$

$n = 2$, $13 = 14$

$n = 3$, $23 = 14$.

$n = 4$, $37 = 28$

$n = 5$, $55 = 35$

$n \geq 4$. $F(n) = 2n^2 > 7n$

$F(n)$ is always greater than or equal to $c * g(n)$

$F(n) = \Omega(g(n))$

$\therefore$ $F(n)$ is at least as fast as the order of growth of $g(n)$. $F(n)$ grows at least as fast as Fn as n approaches Positive infinity.