Design And Analysis of Algorithm

Analytical Questions

Assignment-I

1) Solve the following recurrance relation.

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

Ans: at $n=1$; $x(1) = 0$ (Given).

at $n=2$;

$x(2) = x(2-1) + 5$

$= x(1) + 5$

$= 0 + 5$

$x(2) = 5$

at $n=3$;

$x(3) = x(3-1) + 5$

$= x(2) + 5$

$= 5 + 5$

$= 10$

at $n=4$;

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

$= x(3) + 5$

$= 10 + 5$

$= 15$

$\therefore$ $x(n)$ increaces by 5 for each increament

of 5 difference $(d) = 5$

$x(n) = x(1) + (n-1) \cdot d$ { formula for nth term to find general form of $x(n)$

there, $x(1) = 0$, $d = 5$

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

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

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

Ans: $n=1$; $x(1) = 4$

$n=2$;

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

$= 3x(1)$

$= 3(4)$

$= 12$

$n=3$;

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

$= 3x(2)$

$= 3(12)$

$= 36$

$n=4$;

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

$= 3x(3)$

$= 3(36)$

$= 108$

$\therefore$ $x(n)$ obtained by multiplying the previous ~~factor~~ term by 3

Ratio = 3

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

there; $x(1) = 4$, $r = 3$

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

c) $x(n) = x(n/2) + n$ for $n > 1$ $x(1) = 1$ (solve for $n = 2^k$)

$n = 2^k$

$n=1$; $x(1) = 1$

$n=2$; $x(2) = x(1) + 2 = 3$

$n=4$; $x(4) = x(2) + 4 = 7$

$n=8$; $x(8) = x(4) + 8 = 15$

$n=16$; $x(16) = x(8) + 16 = 31$

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

$\therefore$ $2^k = n$.

$x(n) = x(2^k)$

$= 2^{(\log_2 n)+1} - 1$

$= 2 \cdot 2^{\log_2 n} - 1$

$x(n) = 2n - 1$ //.

d) $x(n) = x(n/3) + 1$ for $n > 1$ $x(1) = 1$ (solve for $n = 3^k$)

Ans: $x(1) = 1$

$n = 3$; $x(3) = x(1) + 1 = 2$

$n = 9$; $x(9) = x(3) + 1 = 3$

$n = 27$; $x(27) = x(9) + 1 = 4$

$x(n) = 1 + \log_3 n$ hold true for $n = 3^k$

$x(n) = 1 + \log_3 n$ //.

2] Evaluate the following recurrences completely

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

take $n = 2^k$ i.e $k = \log n$

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

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

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

$= T(2^{k-3}) + 1 + 2$

$= T(2^{k-3}) + 3$

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

$= T(2^0) + k$

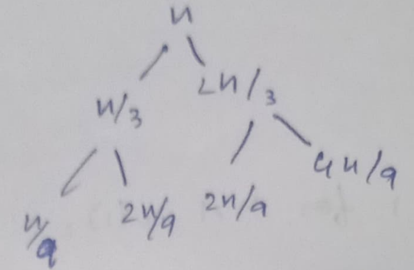
$= T(1) + k$

If $T(1) = 1 \Rightarrow T(2^k) = 1 + k$

i.e $T(n) = \log n + 1$

$\therefore$ Thus $T(n) = \Theta(\log n)$.

ii) $T(n) = T(n/3) + T(2n/3) + cn$, where 'c' is a constant and 'n' is the input size



$T(n) =$ " $+n$ " $=$ sum of all numbers in this tree

length $= \log_3 n$

$T(n) \geq n \log_3 n$

$(\because T$ is $\Omega(n \log n)$

depth $= \log_{3/2} n$

$T(n) \leq n \log_{3/2} n$

T is $\Theta(n \log n)$.

3) Consider the following recussion algorithm

Min1 [A[0.....n-1])

if n=1 return A[0]

else temp = Min1 (A[0.. . n-2])

if temp <= A[n-1] return temp

else

return A[n-1]

a) what does this algorithm compute?

This algorithm computes the minimum value in an array A.

i) Best case (n=1):

if n=1, Only one element. It returns the A[0] as it's the min value in a single element array.

ii) Recursive case ($n>1$):

⇒ if $n>1$, creates ~~the~~ temporary variable (temp)

⇒ Call recursively (A[0 to n-2]) = first $n-1$ element

⇒ Comparing temp with last element (A[n-1])

if temp ≤ A[n-1]
    return temp
else
    return A[n-1]

b] setup a recurrance relation for the algorithm basic operation count and solve it.

Base case : $T(1) = C_1$ [C_1 is constant → return single element]

recursive care : $T(n) = T(n-1) + C_2$ [C_2 → constant representing the basic operations for comparison and assignment.

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

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

ii) Recursive case $(n>1)$:

⇒ if $n>1$, creates the temporary variable (temp)

⇒ call recursively ($A[0$ to $n-2]$) = first $n-1$ element

⇒ comparing temp with last element ($A[n-1]$)

if temp $\le A[n-1]$
    return temp
else
    return $A[n-1]$

b] setup a recurrance relation for the algorithm basic operation count and solve it.

Base case: $T(1) = c_1$ [c_1 is constant → return single element]

recursive case: $T(n) = T(n-1) + c_2$ [c_2 → constant representing the basic operations for comparision and assignment.

$\therefore T(n) = c_2 n^2 + (c_1 - c_2)$

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

4] Analyze the order of growth.

$F(n) = 2n^2 + 5$ & $g(n) = 7n$. use the $\Omega(g(n))$ notation.

as n grows $2n^2$ grows much faster than $7n$

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

if $n = 1$; $7 = 7$

$n = 2$; $13 = 14$

$n = 3$; $23 = 14$

$n = 4$; $37 = 28$

$n = 5$; $55 = 35$

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

$\therefore$ $f(n)$ is always greater than or equal to $c \cdot g(n)$, $F(n) = \Omega(g(n))$

$\because$ $F(n)$ is at least as fast as the order of growth of $g(n)$ grows at least as fast as $7n$ as n approaches positivety infinity.