P. Divya
192324086

DESIGN AND ANALYSIS OF ALGORITHMS FOR DIVIDE AND CONQUER TECHNIQUES - CBA 0666

ANALYTICAL ASSIGNMENT - 1

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

$= x(3) + 5$

$= 10 + 5$

$x(4) = 15$

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

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

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

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

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

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

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

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

$= 3x(1)$

$= 3 \times 4 = 12.$

$x(2) = 12$

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

$= 3x(2) \Rightarrow 3(12)$

$= 3 \times 12 = 36$

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

$= 3x(3)$

$= 3 \times 36$

$x(4) = 108$

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

ratio $= 3$

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

Here . $x(1) = 4$, $r = 3$

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

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

$n = 2^k$

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

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

$= 1 + 2 = 3$

$x(3) = 3$

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

$= 3 + 4 = 7$

$x(4) = 7$

$n=8$: $x(8) = x(8/2) + 8 = x(4) + 8$

$x(8) = 7 + 8 = 15$

$n=16$: $x(16) = x(16/2) + 16 = x(8) + 16$

$= 15 + 16 = 31$

$x(16) = 31$

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

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

$\because 2^k = n$

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

$= 2 \cdot 2\log_2 n - 1$

$x(n) = 2n - 1$

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

$x(1) = 1$

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

$= 1 + 1 = 2$

$x(3) = 2$

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

$= x(3) + 1$

$x(9) = 2 + 1 = 3$

$n=27$ $x(27) = x(27/3) + 1$

$= x(9) + 1$

$= 3 + 1 = 4$

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

Ans: $x(n) = 1 + \log_3 n$ //

2. Evaluate the following recurrences completely

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

Assume

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

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

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

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

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

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

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

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

$= T(2^0) + k$

$= T(1) + k$

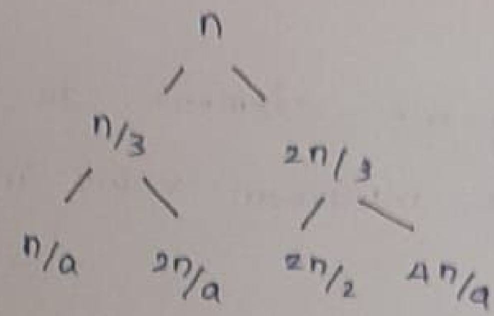
If $T(1) = 1$, we get

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

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

Thus, we got $T(n) = \Theta(\log n)$.

(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 the all numbers in this tree

length = $\log_3 n$

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

$\therefore T$ 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 recursion alogorithm

Min ([A[0.... n-1])

if n-1 return A[0]

Else temp = Min 1 (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

1. Best Case $(n=1)$

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

2. Recursive Case $(n>1)$

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

⇒ Call recursively $(A[0 \text{ to } n-2])$ = first $n-1$ elements

⇒ 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 case ∴ $T(n) = T(n-1) + C_2$ [C_2 → Constant representing the basic operations for comparison and assignment.

final solution

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

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

Analyse the order of growth

$f(n) = 2n^2 + 5$ and $g(n) = 7n$. Use the $\Omega(g(n))$ notation

As n grows, $2n^2$ grows much faster than $7n$

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

if $n = 1$ $7 = 7$

$n = 2$ $13 = 14$

$n = 3$ $23 = 21$

$n = 4$ $37 = 28$

$n = 5$ $55 = 35$

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

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

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

$\because f(n)$ is atleast as fast as the order of growth of $g(n)$.

$F(n)$ grows at least as fast as $7n$ as n approaches positive infinity.