classfellow

Date: / / 20

Page No.

Algorithms lecture-2
Homework

Q1→ Analyze worst-case time complexity of algorithms below.

A1→ Loopmystery1

Proof:-

Line 2, 6 to 10 are not functions or loops only operations, so their complexity is $\Theta(1)$.

For line 5, 'l' loop runs from 0 to k, hence it runs $(k+1)$ times.

For line 4, 'j' loop runs from i to n & step k, hence it runs $\left(\frac{n-i}{k}\right)+1$ times.

For line 3, 'i' loops runs 1 to n, So it runs 'n' times

For total number of iterations, we need summation of all loops' iterations.

For 'j' & 'l' loops, total iterations $= \left(\frac{n-i}{k}+1\right)(k+1)$

Summation from 'i' loop, and using identities,

$$= \sum_{i=1}^{n} \Theta\left(\left(\frac{n-i}{k}+1\right)(k+1)\right).$$

$$= \Theta(k+1)\left[\sum_{i=1}^{n}\left(\frac{n-i}{k}+1\right)\right]$$

$$= \Theta(k+1)\left[\frac{1}{k}\left(n^2 - \frac{n(n+1)}{2}\right)+n\right]$$

$$= \Theta(k+1)\left[\frac{n^2 - \frac{(n^2+n)}{2}}{k} + n\right]$$

$$= \Theta(k+1)\left[\frac{n^2 - \frac{n^2}{2} - \frac{n}{2}}{k} + n\right] = \Theta(k+1)\left[\frac{n^2}{2k} - \frac{n}{2k} + n\right]$$

classfellow

Date: / / 20

Page No.

It can ignore n, as n^2 is dominant term,

So,

$$\Theta\,(k+1)\left[\frac{n^2}{2k} - \frac{n}{2k} + n\right]$$

$$= \Theta\left(\frac{n^2(k+1)}{2k}\right)$$

$$= \Theta\left(\frac{n^2}{k}\right) \quad \text{as } (n^2) \text{ is dominant term}$$

Q3 LoopMystery2 algorithm

A3 Proof

line 2 to 4 and 6, to 9 are single operations, thus takes $\Theta(1)$ time.

For line 5 (while loop) it runs from $i < max$, and i is doubled each iteration.

As it runs from 1 to i, sequence $= 1, 2, 4, 8, 16 \ldots$

After k iterations, value of $i = 2^k$

max is $n \cdot n \cdot n = n^3$

So, $2^k \leq n^3$

Take log both sides,

$$\log_2(2^k) \leq \log_2(n^3)$$

$$k \log_2(2) \leq 3 \log_2(n)$$

$$k \leq 3 \log_2(n)$$

Using identities, we can say time complexity is $\Theta(3 \log_2(n))$

$= \Theta(\log n)$ (3 can be ignored)

Q3

A3 a) Non-recursive complexity of a single call to Recursive-Mystery with an input of size k 3.

For non-recursive complexity, we can consider RecursionMystery's time complexity as $O(n)$.

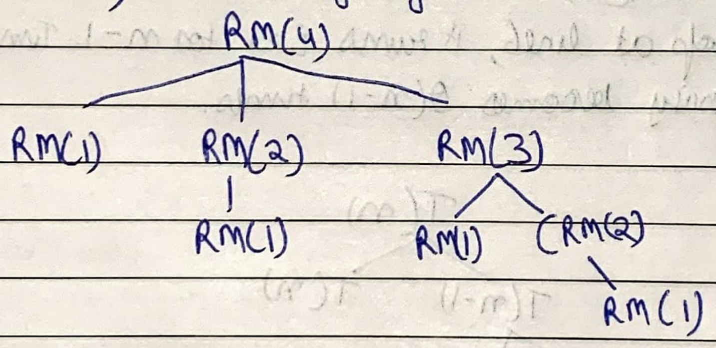
There's no other function or loops in code, so for loop in line 6, time complexity will be $O(n)$ as it runs from $i=1$ to $n-1$.

So, time complexity will summate,

$$O(n) + O(1) + O(1) + \cdots$$
$$= O(n).$$

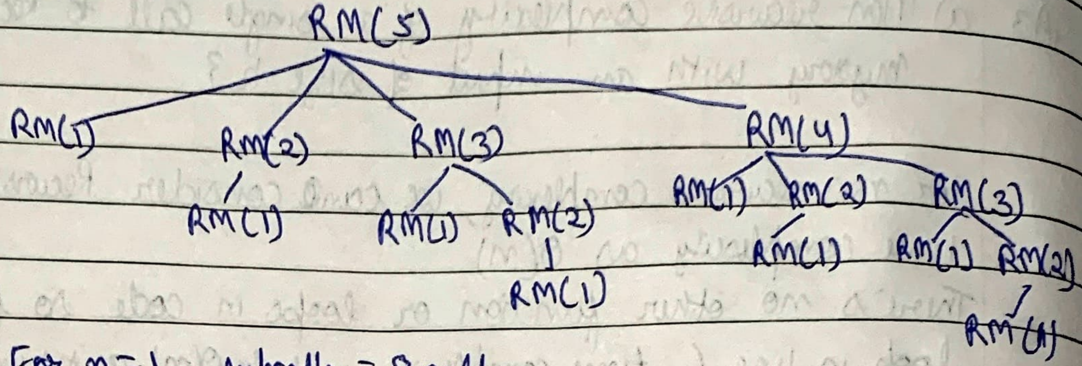
So, non-recursive complexity is $O(k)$ for size k.

b) Recursion tree, Recursion Mystery = RM



For $n=1$, subcalls = 4 calls
$n=2$, subcalls = 2 calls
$n=3$, subcalls = 1 call
$n=4$, subcalls = 1 call.

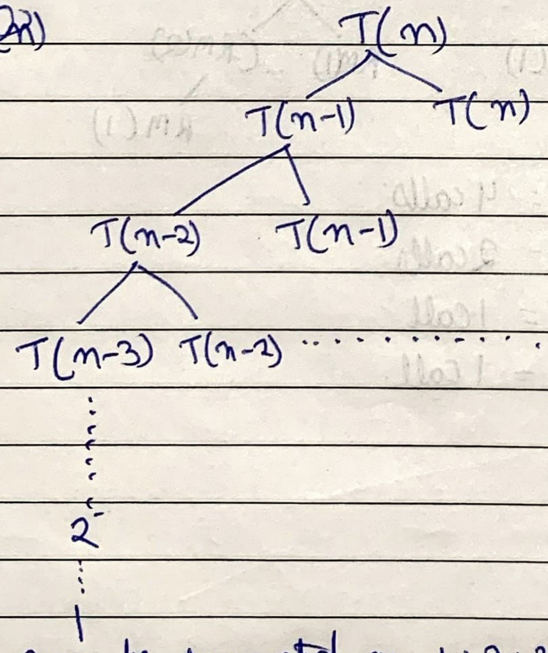
c. Sketch



For $n=1$, subcalls $= 8$ calls
$n=2$, subcalls $= 4$ calls
$n=3$, subcalls $= 2$ calls
$n=4$, subcalls $= 1$ call
$n=5$, subcalls $= 1$ call

d. Write a summation that approximates time complexity for RM(n).

⇒ For loop at line6, it runs $i=1$ to $n-1$. Time complexity becomes $O(n-1)$ times.



This can be summated as $1+2+3 - - - - + (n-1)$.
Equation becomes $= T(n) = \sum_{i=1}^{n-1} T(n-1) + O(n)$.