Shubh Maheshwari
20161170

Algorithm Written Assignment-2

1) We can clearly see the function (foo) holds a recursive relation. Hence

1.) for $n=0$ $T(0) = 1$

for $n \leq 2$ $T(n) = n$

for $n > 2$

$T(n) = T(n/3) + \cancel{T(n/3) + T(n/3)} + c$

where c = constant

2) $T(n) = T\left(\frac{n}{3}\right) + c \times n^0$

By master's theorem

$T(n) = a\,T\left(\frac{n}{b}\right) + c \times n^0$

$T(n) = \cancel{c \times n^{\log} \log_3 ...} \; c(n^0 \log_3 n)$

$= \Theta(\log_3 n)$

Hence Ans $= O(\log_3 n)$

③ 1, 2, 3, 4, 5

foo (3, 3)

+ foo (

3 | 24

f(3, 3, 3)

+

Scanned by CamScanner

2) Given a list of size M which has S elements which are not sorted.

Let see insertion sort:

At the K^{th} step, elements from 0 ... K-1 are sorted

Hence at K^{th} step if we find an element which is not sorted, it would take O(K) time

Hence ~~at~~ for S elements the cost would be $O(S \times K)$

for a large value of M we get $O(S \times M)$

Scanned by CamScanner

3)

1) Value of $f(n)$ is always $n!$ which is greater than n. Hence the Loop runs $n!$ times and each time takes $O(n)$

Hence $T(n) = O(n! \times n)$

2) Value of $f(n)$ is ~~always greater~~ n hence the code runs n times and each time takes $O(n)$

Hence

$$T(n) = O(n \times n)$$

3) Value of $f(n) = n$ Hence code runs n times also each time takes $O(n^2)$

Hence $T(n) = O(n^2 \times n)$

$$= O(n^3)$$

4) as $f(n) = 0$ code ~~for~~ runs for 0 iteration. Hence

$$T(n) = O(1)$$

4) 1) $\Rightarrow$ $T(n) = \sqrt{n}\,T(\sqrt{n}) + 100n$

Lets replace n by ~~m^2~~ 2^m

~~$T(m^2) = mT(m) + 100m^2$~~

$$T(2^m) = 2^{m/2}\,T(2^{m/2}) + 100\,2^m$$

By dividing this by 2^m on both sides

$$\frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 100$$

Let $\frac{T(2^m)}{2^m} = k(m)$

$K(n) = K(n/2) + 100$

$K(n) = \cancel{1+}\ 100 \times \log_2 n$

$K(n) \Rightarrow \frac{T(2^m)}{2^m} = 100 \times \log_2 2^m$

$\frac{T(2^m)}{2^m} = 100 \times m$

$T(2^m) = 2^m \times 100\, m$

$\boxed{2^m = t \qquad m = \log_2 t}$

$T(t) = t \times 100 \times \log_2 t$

$= t \log_2 t \times 100$

$T(n) = O(n \log n)$

ii) $T(n) = T(n/5) + T(\frac{4n}{5}) + O(n)$

$T(\frac{n}{5}) = T(\frac{n}{25}) + T(\frac{4n}{25}) + O(\frac{n}{5})$

$T(\frac{4n}{5}) = T(\frac{4n}{25}) + T(\frac{16n}{25}) + O(\frac{4n}{5})$

$T(n) = T(\frac{n}{25}) + 2T(\frac{4n}{25}) + T(\frac{16n}{25}) + 2 \times n$

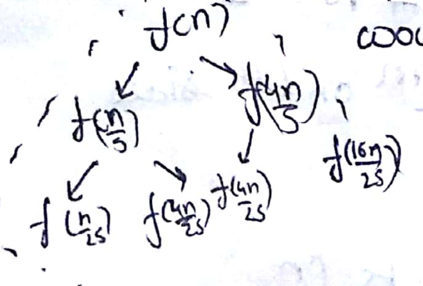
iii) $T(n) = T(n-2) + \log n$

$T(n-2) = T(n-4) + \log(n-2)$

$T(n) = T(0) + \log(n) + \log(n-2) + \log(n-4) \ldots$

$T(n) = T(0) + \log\, n \times (n-2) \times (n-4) \ldots\ 2$

ii) By making the recursion tree Hence ~~max~~ height of the tree could be $\log_{5/4} n$.



and as they max time for each level would be $O(n)$

Hence for $\log_{5/4}(n)$ levels

~~Rank~~ Complexity $= O(n \log_{5/4} n)$

$= \Theta(n \log n)$

Scanned by CamScanner

Q5)

Algorithm to use: Divide and Conquer.

Similiar to merge sort

Where we divide the problem into 2 parts ~~calculate~~. Then these parts are further divided until we reach single elements.

Now we ~~recursively~~ ~~calc~~ calculate the max out of the 2 parts and ~~display~~ ~~it push it upwards until~~. We keep on doing this until we reach the max of whole array.

Algo: $A[n] = \{a_1, a_2 \ldots, a_n\}$

~~Arraymax (A, n):~~

maximum_of_array (A, low, high): ~~if (low~~

mid = $\left(\frac{low + high}{2}\right)$

if (Low < high):

return max (maximum_of_array(A, low, mid), maximum_of_array(A, ~~mid~~ mid+1, high))

elif (low == high)

return A[low]

else

return -INF (Least possible value)

max(a,b): return (a<b ? b:a)

main: Ans = maximum_of_array (A, 0, n-1)

Scanned by CamScanner

Recurrence solution.

By looking at the algorithm we can see that

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

c = time by max function

$$T(n/2) = 2T\left(\frac{n}{4}\right) + c$$

$$T(n) = 2^2 T\left(\frac{n}{4}\right) + (2+1)c$$

$$T(n) = 2^r T\left(\frac{n}{2^r}\right) + (1+2 \dots 2^{r-1})c$$

~~T(n) =~~ for some value $\frac{n}{2^r} = 1$

$$r = \log n$$

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^r}\right) + c\frac{(2^r - 1)}{2-1}$$

$$= nT(1) + c(2^{\log n} - 1)$$

$$= O(n)$$

3) Loop Invariant is the largest element of the array because it is never changed in the whole algorithm

Scanned by CamScanner