CS213: Data Structures and Algorithms **Deadline: 12:00 PM, Feb-09 2021**

Task #2

Instructor: Prof. Sharat Chandran

Minor
Name: Akshay Iyer
Roll No: 190070006
I pledge on my honor that I have not given or received any unauthorized assistance on this assignment or any previous task.

1 Composite Recurrence

Consider two functions f(int n), g(int n) defined as:

```
def f(int n):
    if n == 1:
        return 1
    return f(n-1) + g(n-1)

def g(int n):
    if n == 1:
        return 1
    if (n%2 == 0):
        return g(n/2)
    return g((n+1)/2)
```

What is the big-Oh complexity of $f(n)$?

Let $G(n)$ be the time taken by function g and $T(n)$ be the time taken by the function f to run-

$$T(n) = T(n-1) + G(n-1)$$
$$G(n) = G(\frac{n}{2}) + 1$$

Thus by plug and chug method,

$$G(n) = (....((((G(1)+1)+1)+1)+1).....)$$

(log(n) times 1 is added)

$$G(n) = O(log(n))$$
$$\therefore T(n) = T(n-1) + O(log(n))$$

Thus by plug and chug method,

$$T(n) = (....((((T(1)+log(2))+log(3))+log(4)).... + log(n-1))$$

(Which is equal to O(log(n-1)!)

$$\therefore T(n) = O(nlog(n))$$

2 More Recurrence

Let

$$T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + 1$$

Take the base case as $T(1) = 1$ and you can assume n to be an even power of 2 so that the inputs to T are always integers.
Find $T(n)$ in terms of Ω notation.

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + 1$$
$$T(n) = 2T(\frac{n}{4}) + T(\frac{n}{8}) + 2$$
$$T(n) = 2T(\frac{n}{16}) + 3T(\frac{n}{8}) + 4$$
$$T(n) = 5T(\frac{n}{16}) + 3T(\frac{n}{32}) + 7$$
$$T(n) = 5T(\frac{n}{64}) + 8T(\frac{n}{32}) + 12$$
$$T(n) = 13T(\frac{n}{64}) + 8T(\frac{n}{128}) + 20$$

Thus we see a fibonnaci like sequence, when we expand the higher term in the sum each time(as we are adding the coefficients of the 2 terms in every step to get the new coefficient of the terms in the next step)

Let the nth fibonacci number be F(n)-
$T(n) = F(log(n))T(1) + F(log(n) + 1)T(2) + F(log(n) + 2) - 1$

Assuming T(1)=T(2)=1(We know that T(2) has to be greater than T(1) due to the recurrence relation assuming n=2. thus even by assuming T(2)=1, I am getting a lower bound) (We have to make an assumption for T(2) as nothing is given in the question),
We get-
$T(n) = F(log(n)) + F(log(n) + 1) + F(log(n) + 2) - 1$
$T(n) = 2F(log(n) + 2) - 1$
$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$
Hence, $T(n) = \theta(\left(\frac{1+\sqrt{5}}{2}\right)^{log(n)+2} - \left(\frac{\sqrt{5}-1}{2}\right)^{log(n)+2})$
(As log(n) is always even as given in the question)
Thus, $T(n) = \Omega(\left(\frac{1+\sqrt{5}}{2}\right)^{log(n)} - \left(\frac{\sqrt{5}-1}{2}\right)^{log(n)})$
(As dividing the whole θ quantity by $((1+\sqrt{5})/2)^2$ is greater than the RHS quantity above. Thus the RHS quantity gives us a lower bound).

3 Transformed Recurrences

Let

$$T(n) = 2T(\sqrt{n}) + c$$

and the base case:

$$T(2) = T(1) = 1$$

Give a bound on $T(n)$ in terms of Θ notation. Prove how you obtained the bound.

For simplicity you may consider n to be of a form that ensures only integers are found when you unroll the recursion (that is the inputs to T are always integers).

Here we use the induction method by guessing a function that satisfies the above equation. This is as it is fairly obvious it will be of a log n form. We see that here $T(n)$ is of the same form as $2T(\sqrt{n})$. Thus it has a form very similar to log n. Also, if we assume a constant also added to it, we see that-
constant= 2*constant + c. Thus our constant is -c.
Thus let us assume that-

T(n) = log(n)-c
Thus assuming this is true for some k, we see that-
LHS = $2(\log(\sqrt{k}) - c) + c$

$LHS = 2(0.5log(k) - c) + c$

$LHS = log(k) - c$

$LHS = RHS$

c in the question will be such that it satisfies $T(1)$ and $T(2)$. Thus we do not need to check for the initial case as we will need the value of c, which will be taken care of by the question.
Hence our assumption was correct, and $T(n)$ is of the form $\theta(logn)$

4 His Master's Voice

For each of the recurrences below, state whether the master theorem is applicable or not. If yes, state to which of the three cases the recursion belongs to and find the asymptotic bound. If not, state reasons why the theorem is not applicable. In the cases where master theorem is not applicable, can you find the asymptotic bound using other methods? *(this is not necessary but may fetch you bonus marks)*

The base case for each of these recurrences is $T(1) = \Theta(1)$

(i) $T(n) = 4T(\frac{n}{2}) + n^2 \log^4 n$

Here a=4, b=2.

F(n) = $n^2(logn)^4$

$n^{\log_2 4} = n^2$

$F(n) \neq \theta(n^2)$

Also, for all $n > n_0$-
$\theta(n^{2+\epsilon}) > n^2(logn)^4$

As,
There exists an n_o *such that* $n^\epsilon > (logn)^k$ *for all* $n > n_o$ *, for every* k *and* ϵ

Hence we see that the master theorem is not valid in this case.

(ii) $T(n) = T(\frac{n}{2}) + \tanh n$

Here a=1, b=2.

$$\mathrm{F(n)} = \tanh n = \frac{\sinh n}{\cosh n} = \frac{e^n - e^{-n}}{e^n + e^{-n}} = \frac{e^{2n} - 1}{e^{2n} + 1}$$

$n^{\log_2 1} = 1$

Now, for every ϵ, there exists an n_0 such that for all $n > n_0$-
1-$\epsilon < \frac{e^{2n}-1}{e^{2n}+1} < 1 + \epsilon$

hence,
$F(n) = \theta(1)$

Thus we see that the master theorem is valid in this case, and $T(n) = \theta(\log_2 n)$.
(As limit F(n) as n tends to infinity is 1).

(iii) $T(n) = T(\frac{n}{2}) + n(2 - \cos n)$

Here a=1, b=2.

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

Now, for every n,
$n \leq n(2 - cos(n)) \leq 3n$
(As cos(n) lies between -1 and 1)

hence,
F(n) = $\theta(n)$

Now,
$(n^{0+\epsilon}) < n \ \forall \ \epsilon < 1 \ and \ n > 1$

$T(n) = \theta(F(n)) = \theta(n)$.
But now, $a.f(n/b) \nleq cf(n)$, as $1f(n/2) \leq 3n/2$
We need a cn such that $cn \geq 3n/2$, for some $c < 1$, which is not possible, as we got $c \geq 3/2$ as above.
Thus the master theorem is not applicable here.

5 I Hate Loops!!!

Algorithm 1: : I Hate Loops!!!

```
int a = 0;
for i=1; i≤n; i++ do
    for j=i; j≤n; j+=i do
        a++;
    end
end
```

Find the asymptotic complexity of the above code in terms of n.

In the first i loop, j goes from 1 to n.
In every iteration of j, an increment takes place.

In the second iteration of i, j goes from 2 to n.
Thus j takes (n-1)/2 values.
(n-1) increments take place.

Similarly for the third iteration of i,
(n-2)/3 increments take place.

Thus the total number of increments taking place are-
((1/(n))+(2/(n-1))+(3/(n-2))+.....+n)= Z

Z =((1/(n)+1/(n-1)+1/(n-2).....)+((1/(n-1))+(1/(n-2))....)+.....+1)

Z =$H_n + H_{n-1} + ...H_1$

$H_n = O(log(n+1))$

Z = O(log(n+1))+O(log(n))+.....+ O(log(2)) = O(nlog(n))

(The H_n above is the nth harmonic sum.
We got H_n as $O(log(n))$, by integrating (1/n) from 1 to (n+1) (Which gives us log(n+1)-log(1)), as area under the curve which is the integral, is less than the sum of the n rectangles of width 1 which gives us H_n)
Thus the asymptotic complexity of the above code is O(nlog(n)).