Theory Assignment - 1 , B19003

1) a) Yes, the algorithm can take $O(n)$ on some inputs. $O(n^2)$ does not place asymtotically ~~strong~~ tight bounds all the time. It may or may not place asymtotically tight bounds. So, the algorithm may not take $O(n^2)$ for all algorithms. For some cases, it can take $O(n)$. eg - Consider insertion sort, it take $\Theta(n^2)$ for worst cases i.e. we can say that it takes $O(n^2)$ for worst case. This is when all integer are arranged in descending order and we want sort it in ascending order. However, if the integers are already arranged in ascending order, the time complexity is $O(n)$.

b) Yes, the algorithm can take $O(n)$ on all inputs. $O(n^2)$ does not place asymtotically tight bounds. It just places an upper bound i.e. we are assured that the algorithm will not take more ~~more~~ time than $O(n^2)$ for any input. eg - Insertion sort takes $O(n^2)$ ~~for~~ time complexity for worst case. So, we can also say that it takes $O(n^3)$, as it just places an upper bound.

c) Yes, the algorithm can take $O(n)$ for some inputs. $\Theta(n^2)$ bound does not imply a $\Theta(n^2)$. bound on the running time of insertion sort on every input. eg - in insertion sort, worst case running time is $\Theta(n^2)$. But if the input is already sorted, then we can say that it runs in $O(n)$ time complexity ~~or placing~~ or placing an upper bound on it, we can say $O(n)$ time complexity, for that input.

3) a) $T(n^2) = 7T(n^2/4) + cn^2$ and $T(1) = 1$

changing variable from n^2 to x, we have -

$T(x) = 7 \cdot T(x/4) + cx$

Now, using Master's theorem, we have the first case;

$f(x) = cx = O(x)$

$b = 4, a = 7$

$f(x) = O(x^{\log_4 7 - \varepsilon}) = O(x^{1.403 - \varepsilon})$ and we have

$f(x) = O(x)$. so, $\varepsilon = 0.403 > 0$.

So, ~~$T(x) = \Theta(n^{\log_4 7})$~~.

So, $T(x) = \Theta(x^{\log_4 7})$

But, if we carefully see in $T(x)$, x takes only perfect squares i.e. $x = \{1, 4, 9, \cdots\}$. In real world scenario, we may have x which is not a perfect square. So, to compensate for that, I am redefining:

$$\underline{T(x) = O(x^{\log_4 7})}.$$

Big-oh places an upper bound on the inputs, i.e. if we have $x = 2$ or $x = 3$, we can very well say that $T(x) \leq T(4)$ for $x \leq 4$. ~~In that case, we write~~ ~~it as~~ That's why, I'm preferring big-oh notation.

b) $T(n) = n \times T(\sqrt{n})$.

$T(n^{1/2}) = n^{1/2}\, T(n^{1/4})$

$T(n^{1/4}) = n^{1/4}\, T(n^{1/8})$.

In general form, we can write:–

$$T(n) = n^{1 + 1/2 + 1/4 + \cdots \text{upto } 2^i \text{ terms}}\, T(2)$$

Now, assuming i takes larger values like 1000, 10,000 etc., then 2^i is very large number. So,

$$T(n) = n^{1 + 1/2 + 1/4 + \cdots + \text{to } 2^i \text{ term}}\, T(2) \leq n^{1 + 1/2 + 1/4 \cdots \infty}\, T(2)$$

$$= n^{\frac{1}{1 - 1/2}}\, T(2)$$

$$= n^2\, T(2)$$

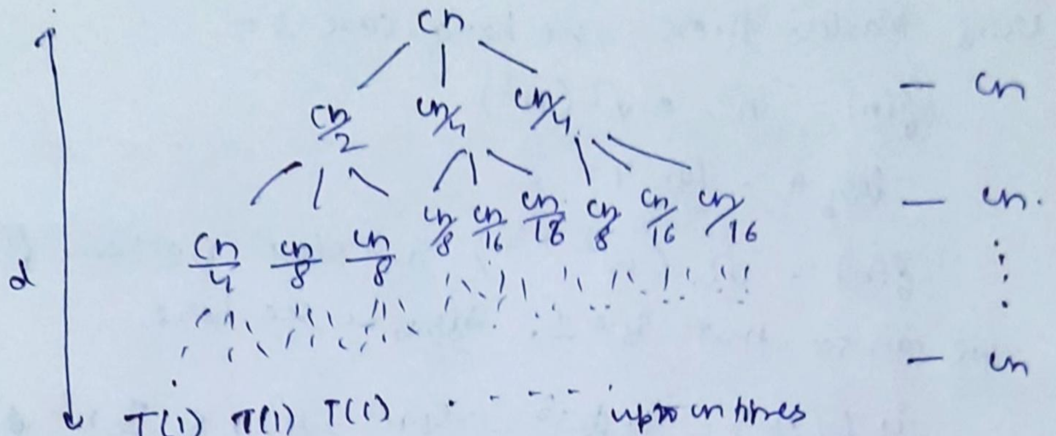
$$= 4n^2$$

Thus, we have $T(n) \leq 4n^2$. Or maybe we can write it as $\underline{T(n) = O(n^2)}$.

c) $T(n) = T(n/2) + 2T(n/4) + 3n/2$.

$T(n) = T(n/2) + 2T(n/2) + cn$ where $c = 3/2$.

Using recursion tree, we have -



Now, considering the depth of the tree, we have different depths for $cn/2$, $cn/4$ and $cn/4$: since, the depths are not equal as the tree is imbalanced, sum of each layer will not give cn. So, let us guess that the complexity is $O(n\lg n)$.

$\cancel{T(n) \le T(n/2) + 2T(n/4) + cn}$

$\cancel{\le d(n/2)\lg(n/2) + 2d(n/4)\lg(n/4) + cn}$

So, $O(n\lg n)$ is the upper bound on the time complexity. We show that $T(n) \le dn\lg n$, where d is a suitable positive constant.

$$
\begin{aligned}
T(n) &\le T(n/2) + 2T(n/4) + cn. \\
&\le d\, n/2 \lg(n/2) + 2d\, n/4 \log(n/4) + cn \\
&= d\, n/2 \lg n + 2d\, \tfrac{n}{4}\lg n - d\left(n/2 \lg 2 + n\lg 2\right) + cn \\
&= dn\lg n - d\left(\frac{3n}{2}\right) + cn \\
&= dn\lg n - dn(3/2) + cn. \\
&\le dn\lg n, \text{ as long as } d \ge \frac{2c}{3} \text{ or } d \ge 1 \quad (\because c = 3/2)
\end{aligned}
$$

So, $T(n) \le dn\lg n$

$\underline{T(n) = O(n\lg n)}$

d) $T(n) = 4T(n/2) + n^3$ and $T(1) = 1$

Using Master's theorem, we have case 3 -

$$f(n) = n^3 = \Omega(n^3)$$

$$\log_b a = \log_2 4 = 2$$

$f(n) = \Omega(n^{2+\varepsilon})$ and since we have $f(n) = \Omega(n^3)$, we can see that $\varepsilon = 1$. Also, we see here,

$$4f(n/2) = 4\left(\frac{n}{2}\right)^3 = \frac{4n^3}{8} = \frac{n^3}{2} \leq \frac{3}{4} n^3 = cn^3$$

where $c = \frac{3}{4}$.

So, $4f(n/2) \leq \frac{3}{4} f(n)$

Thus, by master's theorem, $T(n) = \Theta(f(n))$

$$\underline{T(n) = \Theta(n^3)}$$

e) $T(n) = T(n/3) + c\lg n$.

In book, "Algorithm Design: Foundation, Analysis, and Internet Examples" by Michael T. Goodrich and Roberto Tamassia. The new master theorem mentioned in the book (Pg-268-270) is stronger than that given in CLRS.

So, acc- to new master's theorem,

$$c_{crit} = \log_b a = \lg_3 1 = 0.$$

We have $f(n) = c\lg n = (n^0 \log^1 n)$. Hence, $c = 0$ and $k = 1$. So, $c = c_{crit}$. We fall in 2nd case. We have

$f(n) = \Theta(\lg n)$, so by new Master's theorem -

$$T(n) = \Theta(n^0 \lg^{1+1} n)$$

$$= \Theta(\lg^2 n)$$

Considering, n is not of form 2^i where $i \in \mathbb{Z}^+$, we can write $T(n)$ as -

$\underline{T(n) = O(\lg^2 n)}$, thus placing an upper bound.

4) if $f(n) = o(g(n))$, we say that - * Here o denotes "small-oh"

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0. \quad -①$$

So, in order to find the order, let us use eq ① -

① considering $f(n) = n\lg n$ and $g(n) = \frac{n^{1.2}}{\lg n}$.

$$\lim_{n\to\infty} \frac{\lg^2 n}{n^{0.2}}$$

$$= \lim_{n\to\infty} \frac{2\lg n\,(1/n)}{0.2 n^{-0.8}} \quad \text{(using L' hospital's)}$$

$$= \lim_{n\to\infty} \frac{10\lg n}{n^{0.2}}$$

$$= \lim_{n\to\infty} \frac{10\,(1/n)}{0.2 n^{-0.8}} \quad \text{(using L' hospital)}$$

$$= \lim_{n\to\infty} \frac{50}{n^{0.2}} = 0$$

~~So,~~ $\cancel{\frac{n^{1.2}}{\lg n}} = \cancel{\Omega(n\lg n)}$

So, $n\lg n = o\left(\frac{n^{1.2}}{\lg n}\right)$ or $n\lg n < \frac{n^{1.2}}{\lg n}$.

② Considering. $f(n) = \frac{n^{1.2}}{\lg n}$ and $g(n) = n^2$.

$$\lim_{n\to\infty} \frac{n^{1.2}}{\lg n\,(n^2)}$$

$$= \lim_{n\to\infty} \frac{1}{\lg n\; n^{0.8}}$$

(as $\lg n \to \infty$ and $n^{0.8} \to \infty$)

$$= 0$$

So, $\frac{n^{1.2}}{\lg n} = o(n^2)$ or $\frac{n^{1.2}}{\lg n} < n^2$

③ considering $f(n) = n^2$ and $g(n) = 1.1^n$.

$$\lim_{n\to\infty} \frac{n^2}{(1.1)^n}$$

$$= \lim_{n\to\infty} \frac{2n}{(1.1)^n \ln(1.1)}$$ (using L'hospital)

$$= \lim_{n\to\infty} \frac{2}{(1.1)^n \ln^2(1.1)}$$ (using L' hospital)

$$= 0$$

So, $n^2 = o(1.1^n)$ or $n^2 < (1.1)^n$

(4) considering $f(n) = lg^3(n)$ and $g(n) = n lg n$,

$$\lim_{n\to\infty} \frac{\cancel{n} lg^3 n}{n lg n}$$

$$= \lim_{n\to\infty} \frac{lg^2 n}{n}$$ (using L' hospital)

$$= \lim_{n\to\infty} \frac{2 lg n (1/n)}{1}$$ (using L' hospital)

$$= \lim_{n\to\infty} \frac{2(1/n)}{1}$$

$$= 0$$

So, $\cancel{n} lg^3 n = O(n lg n)$ or $lg^3 n < n lg n$

(5) considering $f(n) = (0.9)^n$ and $g(n) = lg^3(n)$.

$$\lim_{n\to\infty} \frac{(0.9)^n}{lg^3 n}$$

$$= \cancel{\lim_{n\to\infty} \frac{(0.9)^n \ln(0.9)}{3 lg^2 n (1/n)}}$$

$$= 0$$ (as $\lim_{n\to\infty} lg^3 n \to \infty$ and $\lim_{n\to\infty} (0.9)^n \to 0$)

So, $(0.9)^n = o(lg^3 n)$ or $(0.9)^n < lg^3(n)$.

So, finally, we can write the total order as –

$$1.1^n > n^2 > \frac{n^{1.2}}{lg n} > n lg n > lg^3 n > (0.9)^n$$

5) The value returned by the function is $\frac{n(n+1)(n+2)}{3}$ and the worst case running time is $O(n^3/3)$. ~~or~~ $\cancel{O(n^2)}$.

Explanation:

We usually use sigma notation for solving these types of ~~[illegible]~~ looping questions. Consider the following loop -

```
for i = 1 to n :
    for j = 1 to n :
        r = r + 1.
print (r).
```

So, the value of r is given by $\sum_{i=1}^{n} \sum_{j=1}^{n} 1$. So, the value printed is $2n$.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = 2n.$$

Now, for ~~[illegible]~~ our question, we write the output as -

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i+j} 1$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{i} i+1$$

$$= \cancel{\sum_{i=1}^{n} \frac{i(i+1)}{2} + i} = \sum_{i=1}^{n} i^2 + i \quad \left(\because \sum_{j=1}^{i} i = i^2 \quad \sum_{j=1}^{i} 1 = i \right)$$

$$= \sum_{i=1}^{n} i^2 + i$$

$$= \frac{i(i+1)(2i+1)}{6} + \frac{i(i+1)}{2} \Big|_1^n$$

$$= \frac{n(n+1)(2n+1)}{6} + \frac{3n(n+1)}{6}$$

$$= \frac{2n(n+1)(n+2)}{6}$$

$$= \frac{n(n+1)(n+2)}{3} = O(n^3/3)$$

2) We prove by loop invariants. We define the loop invariant as -

"Let M, N be the values typed in by the user into variables a, b. So, just before and just after every iteration $i < \xi_s$,

$$GCD(M,N) = 2^{i+1}\, GCD\left(\frac{M}{2^{i+1}}, \frac{N}{2^{i+1}}\right)$$

Putting $a = M/2^{i+1}$ and $b = N/2^{i+1}$, $GCD(M,N) = 2^{i+1}\, GCD(a, b)$

and for every iteration $i \geq \xi_s$,

$$GCD(M', N') = GCD(a, b)$$

where $M' = M/2^{\xi_s}$ and $N' = N/2^{\xi_s}$, $\xi_s \in Z^+$ and $\xi_s \geq 0$.

Here, a and b are changing, M, N, M', N' are fixed"

<u>Initialization</u>: In the first iteration, we have $a = M$ and $b = N$ where M, N are values typed in by the user. So,

$$GCD(M,N) = GCD(M,N)$$
$$GCD(M,N) = GCD(a,b) \quad \text{as } a = M, b = N.$$

Thus, the invariant condition is true in the first iteration of the loop.

<u>Maintenance</u>: Consider iteration j. Let M is of the form $2^{\alpha_1} 3^{\alpha_2} 5^{\alpha_3} \ldots$ and N of the form $2^{\beta_1} 3^{\beta_2} 5^{\beta_3} \ldots$. Now, we define ξ_s as $\xi_s = \min(\alpha_1, \beta_1)$. Then, we have 2 cases - $j \leq \min(\alpha_1, \beta_1)$ or $j < \xi_s$.

<u>Case 1</u>: $j \leq \min(\alpha_1, \beta_1)$

So, by property of GCD, we know $GCD(M,N) = 2^{j+1}\, GCD\left(\frac{M}{2^{j+1}}, \frac{N}{2^{j+1}}\right)$

That is what is being done by the code. We have defined k which is getting multiplied by 2 at each iteration until i has reached $\xi_s - 1$. According to code,

$$a = M/2^{i+1} \text{ and } b = N/2^{i+1}$$

as a and b are getting updated after each iteration.

$$\text{So, } GCD(M,N) = 2^{i+1}\, GCD(a,b)$$

Case 2: $j \geq \min(\alpha_1, \beta_1)$ or $j \geq \xi_s$.

From $i = \min(\alpha_1, \beta_1)$ to j, we define c as $c = a-b$.

So,

~~GCD (a, b)~~ $a = (a-b) \bmod b$. ($\because$ b divides b)

$a = c \bmod b$.

Then, by Euclid's algorithm, ($\because c < a$)

$$GCD(a,b) = GCD(c, b)$$

$$GCD(a,b) = GCD(a-b, b)$$

The above thing is true for $a > b$. Similarly, we can show for $b > a$. Actually, in code, we don't have c, we are just updating a as $a = a-b$, keeping b constant when $a > b$

and b as $b = b-a$, keeping a constant when $b > a$.

This will go on until $a = b$.

Finally, I will conclude this section by saying that from $i = \xi_s$, $a = \frac{M}{2^{\xi+1}}$ and $b = \frac{N}{2^{\xi+1}}$, and then it is getting updated as $a = a-b$ and $b = b-a$.

<u>Termination</u>: While terminating, or at the last iteration, we will have the condition $a = b$ (check code). So, we know

$$GCD(a, b) = GCD(a, a)$$

$$= a.$$

Then, we give the output as $2^{\xi+1} \cdot a$ as $GCD(M, N)$.

$$GCD(M, N) = 2^{\xi+1} \cdot a$$

\# We have also placed a base condition in code. Base condition is when either of a or b is 0. So, $GCD(a, 0) = a$, $GCD(0, 0) = 0$ and $GCD(0, b) = b$.