Homework-2:

① True or False:

(a.) If f, g, h are positive increasing functions with f in $O(h)$ and g in $\Omega(h)$, then the function $f+g$ must be in $\Theta(h)$.

→ $\boxed{\text{TRUE}}$

(b.) Given a problem with i/p size n, a soln with $O(n)$ time complexity always costs less in computing time than a soln with $O(n^2)$ time complexity.

→ $\boxed{\text{FALSE}}$

$O(n)$ and $O(n^2)$ shows proportionality to n and n^2 respectively, but for cost we calculate with constants as well. So the given statement cannot be generalized for every case.

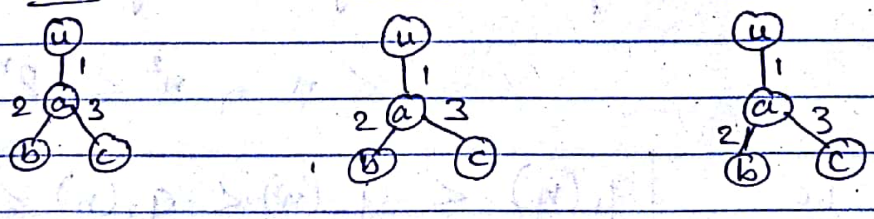
(c.) $F(n) = 4n + \sqrt{3n}$ is both $O(n)$ and $\Theta(n)$.

→ $\boxed{\text{TRUE}}$

(d.) For a search starting at node s in graph G, the DFS Tree is never the same as BFS Tree.

→ $\boxed{\text{FALSE}}$.

eg: $\boxed{G}$ DFS = BFS



(e.) BFS can be used to find the shortest path between any two nodes in a non-weighted graph.

→ $\boxed{\text{TRUE}}$

② Reading Assignment - Ch-2 and 3 ✓

③ Ascending order of growth rate:

$$\boxed{\sqrt{2n} < n+10 < n^2 \log n < n^{2.5} < 10^n < 100^n}$$

i.e. <u>ans</u> $f_2(n) < f_3(n) < f_6(n) < f_1(n) < f_4(n) < f_5(n)$

④ Ascending order of growth rate:

Let us add log to every value:

$g_1(n)$: $\log 2^{\sqrt{\log n}} = \sqrt{\log n}\ \log_2 2$. (considering base 2)
$= \sqrt{\log n}$

$g_2(n)$: $\log 2^n = n \log_2 2 = n$

$g_3(n)$: $n(\log n)^3 = \log[n \cdot (\log n)^3] = \log n + \log(\log n)^3$
$= \log n + 3 \log \log n$

$g_4(n)$: $\log n^{4/3} = \frac{4}{3} \log n$

$g_5(n)$: $\log n^{\log n} = \log n \cdot \log n = \cancel{2 \log n}\ (\log n)^2$

$g_6(n)$: $\log 2^{2^n} = 2^n \log_2 2 = 2^n$

$g_7(n)$: $\log 2^{n^2} = n^2$.

So we get:

$$\sqrt{\log n} < \cancel{2}(\log n)^2 < \log n + 3\log\log n < \frac{4}{3} \log n$$
$$< n < n^2 < 2^n$$

i.e. $\boxed{g_1(n) < g_5(n) < g_3(n) < g_4(n) < g_2(n) < g_7(n) < g_6(n)}$

Scanned by CamScanner

(5) $f(n) = O(g(n))$

(A) $\log_2 f(n)$ is $O(\log_2 g(n))$

$\boxed{\text{False}}$

For any constant value within the function, the value may or may not be small enough to ignore to generalize. For $2 \log 3 \rightarrow \log 2 + \log \log 3$ Now, $\log \log 3 << \log 2$ and so $\log 2$ can't be ignored. Hence $\log_2 f(n)$ is not in $O(\log_2 g(n))$.

(B.) $2^{f(n)}$ is $O(2^{g(n)})$

Let $f(n) = 10n$, so $g(n) = O(n)$
~~Let f(n)~~ $= 2^{f(n)} = 2^{10n}$; and $2^{g(n)} = 2^n$

$2^{10n} \neq 2^n$

Hence, the given statement is $\boxed{\text{False}}$.

(C.) $f(n)^2$ is $O(g(n)^2)$

Let $f(n) = 2n$; $\therefore g(n) = O(n)$

$(2n)^2 \Rightarrow O(n^2)$

Let $f(n) = n^{1/2}$; $\therefore g(n) = O(n^{1/2})$

$\therefore (n^{1/2})^2 = n \Rightarrow O(n)$

So $f(n)$ is always $\leq$ some constant $*$ $g(n)$

Similarly, $f(n)^2$ is always $\leq$ some constant $*$ $(g(n))^2$

Hence, the given statement is $\boxed{\text{True}}$

(6) for $i = 1, 2, \dots, n$
for $j = i+1, i+2, \dots, n$
add up array entries $A[i]$ through $A[j]$
Store the result in $B[i, j]$
Endfor
Endfor

(A.) First loop runs for 1 to n iterations
i.e. total → n iterations.
Second loop runs for i+1 to n iterations
i.e. total n−i times for each i
Add operations adds each entry at most till n.
Store operation takes constant time.
So, the first loop takes some $O(n)$,
second loop: $O(n-i)$, add operation $O(n)$,
store operation: $O(1)$

Hence, the time complexity will be $\boxed{O(n^3)}$ order.

(B.) For the given algorithm, for any value of n (as input), both the loops and add operation will run for that much amount of time i.e. $n, (n-i), n \Rightarrow O(n^3) \Rightarrow \Omega(n^3)$
So, the running time of the algorithm on an input size n is also $\Omega(f(n))$.
In simple words, there is no best case or worst case here, or in other words, the best case and the worst case are the same here.

(C) More efficient algorithm:

Let there be a variable 'x' such that $x=0$

```
For i = 1, 2, 3, . . . , n
    For j = i+1, i+2, i+3, . . . , n
        If (j-i == 1) then
                x = A[i] + A[j]
        Else ~~Store x in B~~
                x = x + A[j]
        End If
      Store x in B[i, j]
    End For
End For
```

In this algorithm:

> Outer loop will run upto n-iterations:
So the running time will be $O(n)$

> Inner loop will run upto $n-i$ iterations:
So the running time will be at most $O(n)$

> If - Else condition having add operations will take constant time.

> Store operation will take constant time.

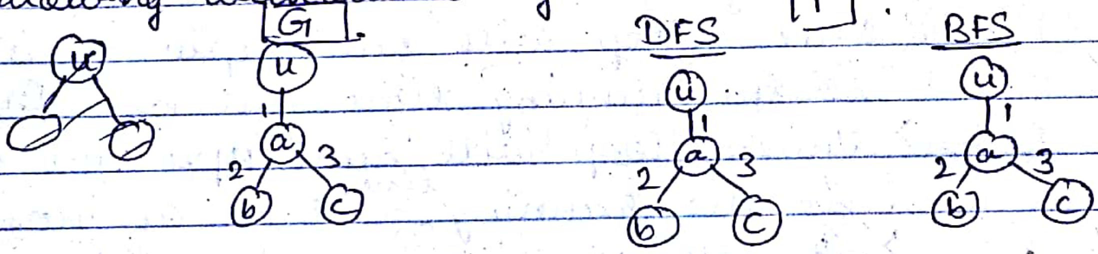
→ Hence, the more efficient way to resolve this problem as per the above algorithm gives the time complexity of at most $\boxed{O(n^2)}$.

Scanned by CamScanner

(7) Let e be an unexplored edge
Let v be any visited node
Let v_i be any node (for our starting condition)
Let v_j be an adjacent node to v_i that is explored with its respective edge explored as well

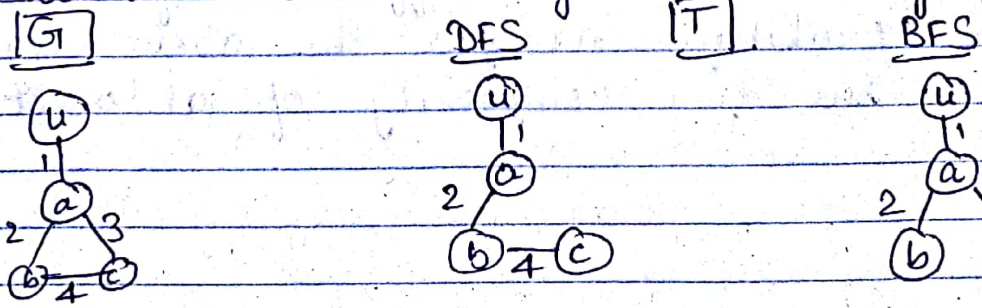
For every v_i
If e leads to v && v_j is the parent of v
then return cycle '$v_i - v - v_j$'
Else If e does not lead to v
then return 'Cycle does not exist' for v_i.
EndIf
EndFor.

(8) For a connected graph G, let us consider the following without a cycle: $\boxed{G}$. $\boxed{T}$.



So the DFS and BFS trees are the same as graph i.e. $\boxed{G = T}$.

→ Now, let us consider a cycle in the graph: $\boxed{T}$.



G contains extra edge '3' that does not belong to DFS tree; and G contains edge '4' that does not belong to BFS tree.
→ Hence, by this we can say that for G = T, G cannot contain any edges that does not belong to T.

Scanned by CamScanner