Mid Semester Examination (5 The Sem), 2021

Subject Name :→ Algorithms.

Subject Code :→ IT 3104.

Date of Examination :→ 08.10.2021.

Name :→ Aniket Majhi.

Examination Roll No → 510819019.

GSuite ID :→ 510819019.aniket@students.iiests.ac.in

Number of sheets uploaded :→ 11

Answer.

(3.) (a.) $T(n) = pn^2 + qn + r$ ——— (i)

for, $n > 1$.

$q.n^2 > q.n$ and $r.n^2 > r$

and $n = 1$

$q.n^2 = q.n$ and $r.n^2 = r.n$

$\therefore$ for $n \geq 1$,

$q.n^2 \geq q.n$

and $r.n^2 \geq r$

So from the (i),

We got,

$T(n) \leq p.n^2 + q.n^2 + r.n^2$

$\Rightarrow T(n) \leq (p+q+r).n^2$.

for all, $n \geq 1$.

Now as p, q, r all are constants, so,

$p+q+r$ is also constant

let $p+q+r = c$.

$\therefore \Rightarrow T(n) \leq c.n^2$.

So, we can write,

$T(n) = O(n^2)$ | c is constant and also $T(n) \geq 0$.

On the other hand,

$T(n) \geq pn^2$ | setting $\underset{=0}{qn}$ and $r = 0$.

p is constant.

so, $T(n) = \Omega(n^2)$.

(b)

~~Given that~~

~~Given that,~~

~~$f = \Theta(g)$ —— (i)~~

~~$g = \Theta(n)$ —— (ii)~~

~~from eqn we get by the definition of:~~

We could have chosen, n^3 for the big oh part but ~~we are inte~~ vice

$$T(n) \le c \cdot n^3$$

but, we are interest in the tighest upper bound which we can get by n^2 so $T(n) = O(n^2)$.

for the other case, we could also say,

$$T(n) \le q \cdot n$$

but we are interested in finding maximum lower bound which can be found by taking n^2.

so, $T(n) = \Omega(n^2)$.

∴ we got,

$$T(n) = O(n^2)$$
$$T(n) = \Omega(n^2)$$

(Proved).

(b.) $f = \Theta(g)$

By the definition,
We can say

there exist positive constant c_1, c_2 and n_0

~~for~~ which,

$c_1 \cdot g \leq f \leq c_2 \cdot g$. for all $n \geq n_0$.

——(i)

Again, $g = \Theta(n)$

Similarly we can write,

$c_1' \cdot n \leq g \leq c_2' \cdot n$ for $c_1', c_2 > 0$ and $n_0 \geq n$.

——(ii)

So, from (ii) we write →

~~$f \geq c_1 \cdot g$ and $f \leq$~~

$g \geq c_1' \cdot n$, $g \leq c_2' \cdot n$.

Putting these two values of g in eq^n (i).

We get,

$f \geq c_1 \cdot g$.	$f \leq c_2 \cdot g$
and $g \leq c_2' \cdot n$	and $g \geq c_1' \cdot n$
$\therefore f \geq c_1 \cdot c_2' \cdot n$	$\therefore f \leq c_1' \cdot c_2 \cdot n$.

$\therefore c_1 \cdot c_2' \cdot n \leq f \leq c_1' \cdot c_2 \cdot n$.

$c_1 \cdot c_2'$ and c_1', c_2 are again positive constant, as $c_1 > 0$, $c_2' > 0$, $c_1' > 0$, $c_2 > 0$

So, by definition, $f = \Theta(n)$. [Proved].

(5.)

(a.) Master theorem :→

let $a \geq 1$ and $b > 1$ be constants.

let $f(n)$ be a function,

and also let $T(n)$ be defined by the non negative integers by the recurrence as,

$$T(n) = aT(n/b) + f(n).$$

Then, [here $n/b \Rightarrow \lceil n/b \rceil$ or $\lfloor n/b \rfloor$]

$T(n)$ can be bounded asymptotaeally as follows

(i) If $f(n) = O(n^{\log_a b - \epsilon})$ for $\epsilon > 0$

then $T(n) = \Theta(n^{\log_a b})$.

(ii) If $f(n) = \Theta(n^{\log_a b})$ then.

$T(n) = \Theta(n^{\log_a b} \cdot \log n)$.

(iii) If $f(n) = \Omega(n^{\log_a b + \epsilon})$ for $\epsilon > 0$

and if $af(n/b) \leq c \cdot f(n)$ for $c < 1$.

and all large n,

then $T(n) = \Theta(f(n))$.

(b.) $T(n) = T(2n/3) + 1$.

Here, ~~$n^{\log_a b}$~~ $a = 1$, $b = 3/2$ and $f(n) = 1$.

~~$n^{\log}$~~

Now,

$$n^{\log_a b}$$

$$\Rightarrow n^{\log_{3/2} 1}$$

$$\Rightarrow n^{0}$$

$$= 1.$$

Now, $f(n) = \Theta(n^{\log_a b})$

$$= \Theta(1)$$

This is the case 2 forom the master theorem,

by the master theorem,

we can write,

$$T(n) = \Theta(\log n).$$

~~Merge~~ Procedure of merge sort :→

(2) (a) Given that,

$$f = O(g).$$

From ~~the~~ definition of Big oh notation,

We can say,

$$0 \leq f \leq c \cdot g \quad \text{for } c > 0$$

$$\therefore 0 \leq f(n) \leq c \cdot g(n) \quad \text{for } n \geq n_0$$

$$\Rightarrow \frac{0}{c} \leq \frac{1}{c} \cdot f(n) \leq g(n) \quad [\text{dividing by } c.]$$

$$\Rightarrow 0 \leq \frac{1}{c} \cdot f(n) \leq g(n)$$

As, c is a Const. we can say $\frac{1}{c}$ is also a Const.

Let us say $\frac{1}{c} = d$.

$$\therefore 0 \leq d \cdot f(n) \leq g(n) \quad \forall n \geq n_0 \quad \text{—(i)}$$

From the definition of $g = \Omega(f)$

$$0 \leq c \cdot f(n) \leq g(n) \quad \Big| \quad \text{for } \forall n \geq n_0, \; c > 0 \quad \text{—(ii)}$$

Comparing (i) and (ii) we can say.

~~$f(n) = \Omega(g(n))$~~

$$g = \Omega(f) \quad [\text{Proved}].$$

(b.)

Given that

$f = O(h)$

So by the definition,

We can write,

$$0 \leq f \leq c_1 \cdot h \quad [\text{for } c_1 > 0]$$

$$\Rightarrow 0 \leq f(n) \leq c_1 \cdot h(n) \quad [\text{for } \forall n \geq n_0]$$

$$\text{———} (i)$$

Similarly,

from $g = O(h)$

We can write,

$$0 \leq g \leq c_2 \cdot h \quad [\text{for } c_2 > 0]$$

$$\Rightarrow 0 \leq g(n) \leq c_2 \cdot h(n) \quad [\text{for } \forall n \geq n_0]$$

$$\text{———} (ii)$$

By ~~Combining~~ Adding $[(i)+(ii)]$ we get.

$$0 \leq f(n) + g(n) \leq c_1 \cdot h(n) + c_2 \cdot h(n)$$

$$\Rightarrow 0 \leq f(n)+g(n) \leq h(n)\,[c_1+c_2]$$

$$0 \leq f(n)+g(n) \leq c_3\, h(n)$$

$c_3 > 0$ as $c_1 > 0$, $c_2 > 0$

By the definion we can write,

$$f+g = O(h) \quad [\text{Proved}).$$

(4.) Merge Procedure of merge sort :-

```
MERGE ( array A, int lo, int mid, int hi ){

    ~~array B[lo]~~
    left_size = mid - lo + 1;
    right_size = hi - mid;

    array B[left_size];
    array C[right_size];

    for i from 0 to left_size :
        B[i] = A[i + lo];

    for i from 0 to right_size ;
        C[i] = A[i + mid + 1];

    i = 0
    j = 0
    k = lo
    while (i < left_size and j < right_size){
        if ( B[i] <= C[j])
            A[k++] = B[i++];
        else A[k++] = C[j++];
    }
    while (i < left_size) A[k++] = B[i++];
    while (j < right_size) A[k++] = C[j++];
```

(b.) <u>Running time of merge sort :-</u>

The recurrence relation for mergesort algo is,

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

By combining,

$$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$$

we say, $2T(n/2)$.

$$\therefore T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & \text{others.} \end{cases}$$

Now,

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\left[2T(n/4) + n/2\right] + n \\ &= 2\left[2\left[2T(n/8) + n/4\right] + n/2\right] + n \\ &= 8T(n/8) + 3n \\ &= 2^3 \cdot T(n/2^3) + 3 \cdot n \end{aligned}$$

in general, $2^k \cdot T(n/2^k) + k \cdot n$.

Now, assume, $n = 2^k$ | as $T(1) = 1$.
<u>$\therefore$ $T(n/2^k) = 1$</u>

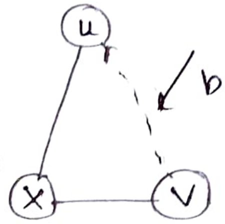
$$\begin{aligned} \therefore T(n) &= n \cdot T(1) + \log n \cdot n \\ &= n \log n + n. \end{aligned}$$

Thus running time <u>$\Theta(n \log n)$</u>.

(7.)

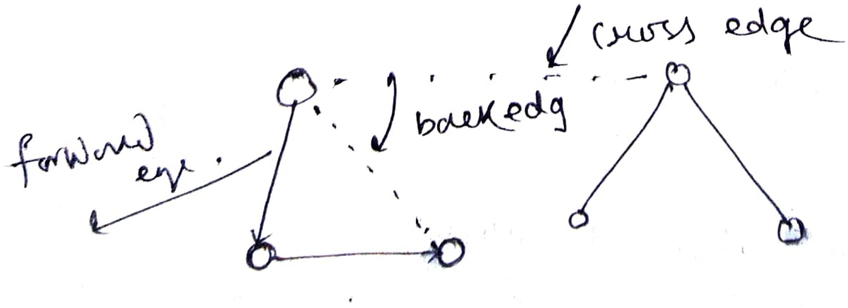
(a)

(i) <u>Back edge</u> :→ Back edges in a DFS are those edges (u,v) conneting a vertex u to an ancestor v in a DFS tree.



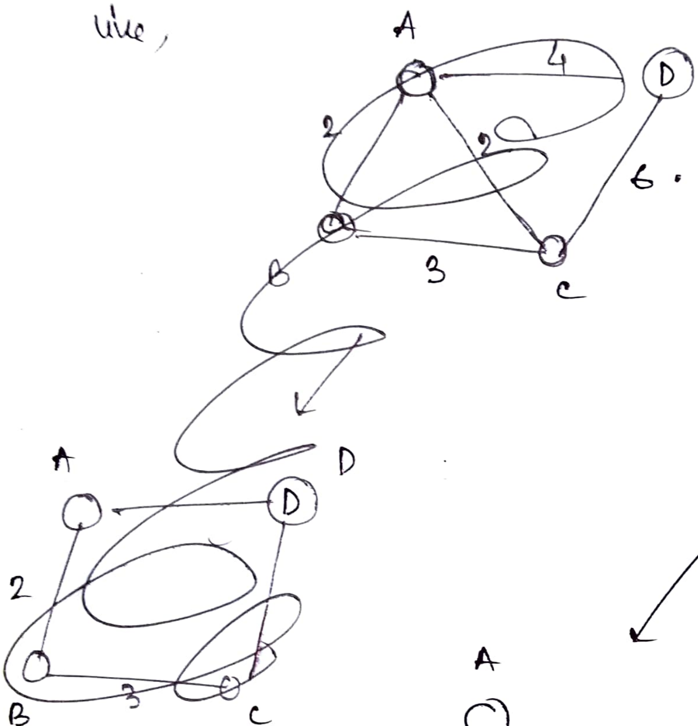
(ii) <u>Forward ege</u> :→ Forward edges are those non tree ~~ed~~ edgers (u,v) Conneting a vertex u to its decendant v.

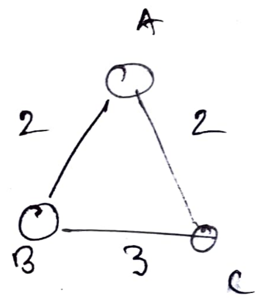
(iii) <u>Cross edge</u> :→ Cross edges are all other edges, means they can go vertices in the same DFS tree, as long as one vertex is not an ancestor to another or they can go between vertices in different DFS trees.

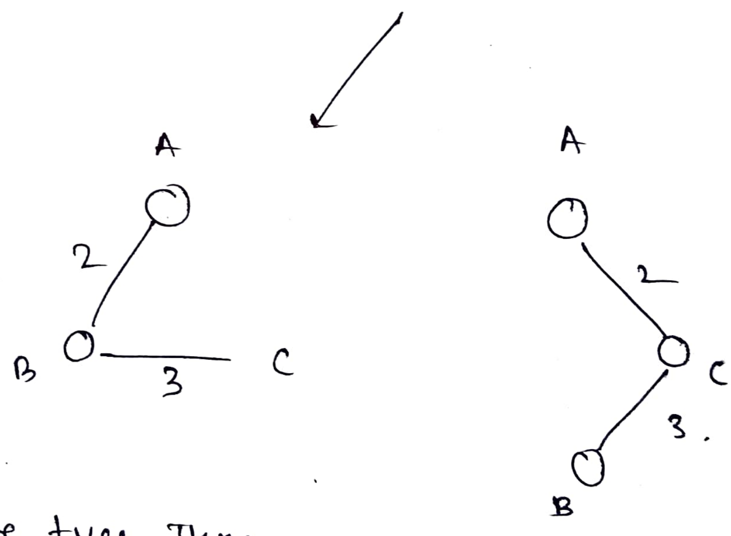


(b.) The minimum spanning tree of a graph may not be unique if there ~~are~~ exist other ~~~~ edges with the same weight as that of the current edges inside the mst.

like,







In the above tree there are two mst possible as shown in the figure.