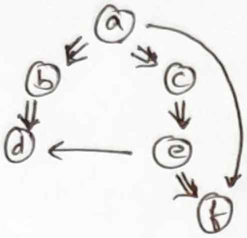
Q1) a) BFS ⇒ The order for BFS is

ⓢ, ⓐ, ⓒ, ⓓ, ⓔ, ⓑ when started from ⓢ.

b) DFS ⇒ The order for DFS is

ⓢ, ⓐ, ⓒ, ⓔ, ⓑ, ⓓ, when started from ⓢ

Q2) a) False; That edge can also be a forward edge, as in the example,

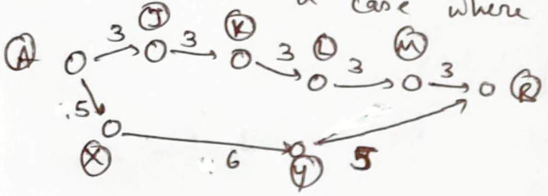


⇒ Tree edge

e → d is cross edge
a → f is forward edge.
But, here both the edges are used on finished vertices.

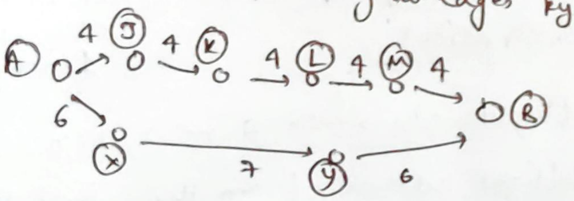
~~b) True; This is because~~

b) False; we can have a case where the path turns out to be longer.



AJKLMB is shorter than AXYB, since the first path has a weight of 15 and the second has a weight of 16.

But after incrementing all edges by,



AJKLMB becomes 20, while AXYB becomes 19. Here the second path is shorter now.

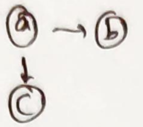
c) False; Dynamic programming follows a top down approach. Similarly, DFS follows a top down approach.

Even in a bottom up fashion, it is similar to DFS, since it completely clears out one path and then moves to the next or sub problem.

d) False; For different order of adjacency lists, DFS algorithma visits different nodes first.

Similarly, if the starting point is different, the whole tree will be different. (for order of vertices)

Eg →



If order is given as starting node from a ~~Then~~ and list as adj (a) = [b ; c]

The ~~lis~~ tree is a, b, c

But if starting node is c and list is adj (a) = [b ; c], then we have 2 trees for DFS, giving us a forest of DFS trees

e) True; If all back edges are removed, no cycle remains. Now running DFS again would give same order (since back edges are not counted for the DFS tree anyways).

Q3) a) n airports and m daily flights

For the number of vertices, $V = 2m$ (m arrivals, m departs)

the number of edges, $E = 1 + m + 99n$.

Edges are this because then 100 events ⇒ 99 intervals (for 1 airport)

∴ 99 × n (for n airports)

Then we have m flight edges ⇒ ~~1~~ m + 99n.

Also, we have 1 flight interval from the start time ⇒ 1 + m + 99n

b) After we have constructed the graph, the shortest travel time can be found out using a simple BFS or a DFS. ⇒ All the vertices that can be arrived from start vertex give us the time required by traveller. So we find the vertex for airport that is quickest to reach.

We can also use Dijkstra for the same.

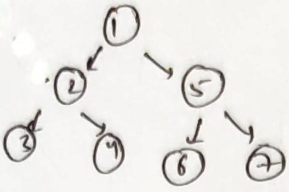
Q4)

a) True; We can just insert the elements 1, 2, 5, 3, 4, 6, 7 - in this order and then do the preorder traversal. the output will be sorted.

This can be a min heap.

1	2	5	3	4	6	7

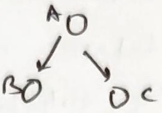
⇒



Inorder gives 1, 2, 3, 4, 5, 6, 7

b) ~~True~~ False; For Inorder, the elements cannot be inserted into a min-heap or a max-heap.

Since for



for inorder, we have BAC.

Now, for this to be sorted, $B > A > C$ or $B < A < C$.

But this definition doesnt match with either min heap or max heap,

For min heap, $A < B$
$A < C$

For max heap $A > B$
$A > C$

If we bypass min and max heaps, then the heap of elements

4	2	6	1	3	5	7

gives a sorted order for inorder traversal. ⇒ 1, 2, 3, 4, 5, 6, 7

c) True; For group of 5 elements, we have. $T(n) = T(\frac{n}{5}+1) + T(\frac{7n}{10}+5) + O(n)$

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

For group of 3 elements, we have $T(n) = T(\frac{n}{3}+1) + T(\frac{2n}{3}+3) + O(n)$

($\because$ Number of elements less than median $\geq 2\left(\lceil \frac{n}{3} \rceil / 2 - 2\right)$
$\geq \frac{n}{3} - 4.$)

Thus, this recurrence also gives us $T(n) = O(n)$

d) False;
For heaps (priority queues), ~~get~~ find Max is $O(1)$, but but ~~delt~~ deleteMax and insert are $O(\log n)$

find Max is $O(1)$, since we just give the first element of array of max heap.

Insert is $O(\log n)$ since we insert at last space and then shift that element up to follow the max heap conditions. (Heapify takes $O(\log n)$).

Deletemax is $O(\log n)$ since we swap first and last element and then delete last element and then reorder the heap to satisfy max heap conditions. (Heapify takes $O(\log n)$).