Name: Animesh Patni

A#: A20403240

Assignment 2: Design Analysis if Algorithm.

Question 1 :→

Given a graph → (V, E)

$w : E \rightarrow Z^+$

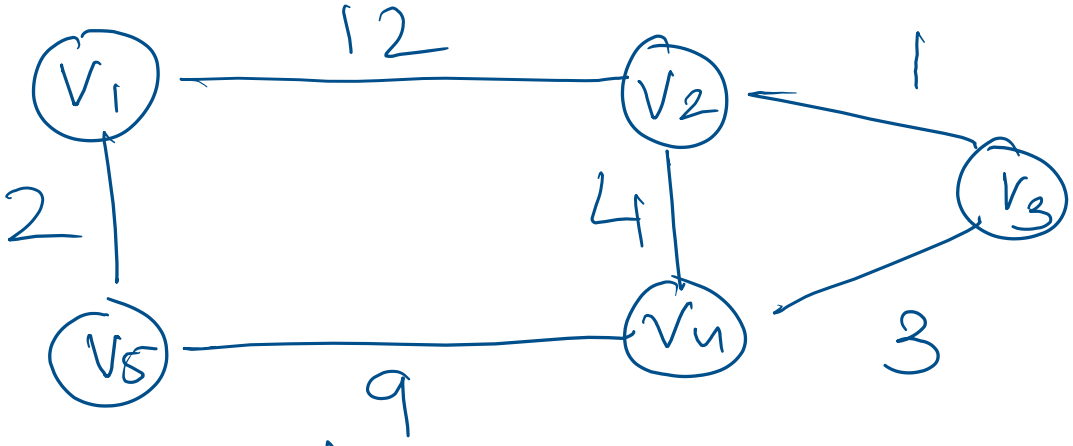
Goal :- $\langle v_1, v_2, \cdots v_n \rangle$

Pseudo code :→

```
1  LEAST_INDUCTIVITY (G, w)
2      H → MAKE_HEAP ()
3      Q → WEIGHTED_DEGREE (V[G], w)
4      FOREACH u ∈ Q do:
5          INSERT (u)
6      WHILE (H ∈ Θ) do:
7          X = X + EXTRACT_MIN (H)
8          FOREACH i ∈ Adj (X):
9              j = NEW_WEIGHTED (x, i)
10             DECREASE_KEY (H, i, j)
11     FOREACH i ← X[V] to 0:
```

(12) PRINT (i).

Example :→



Weighted degree :→

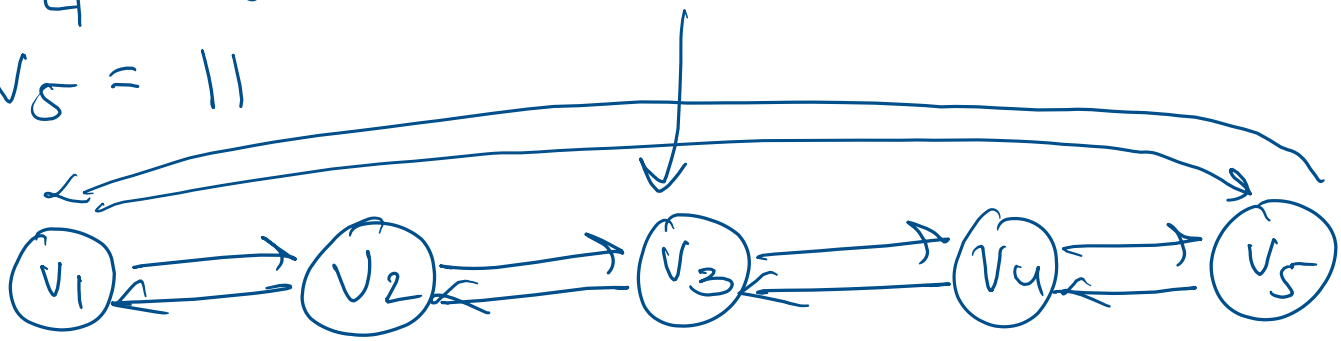
$V_1 = 14$

$V_2 = 17$

$V_3 = 4$

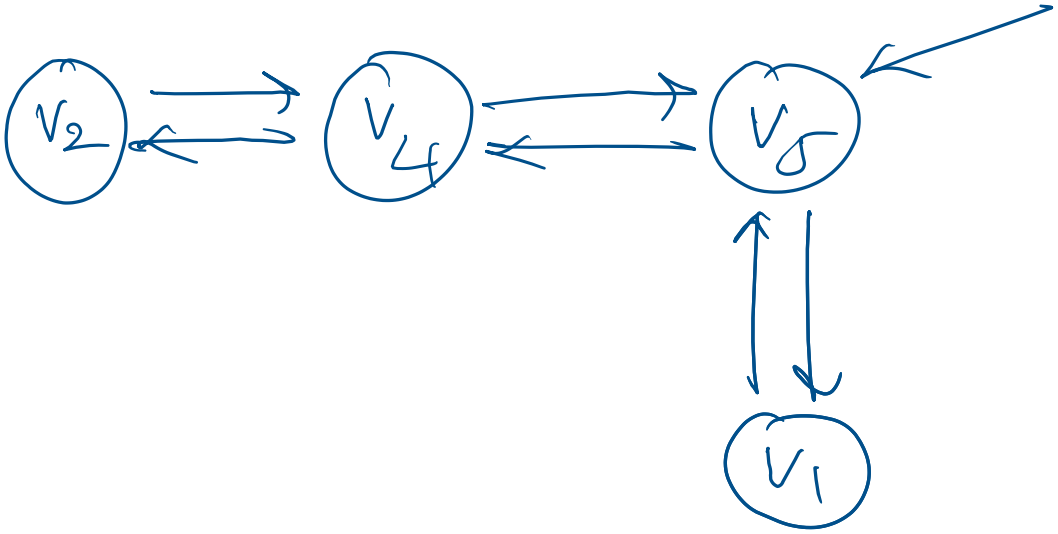
$V_4 = 16$

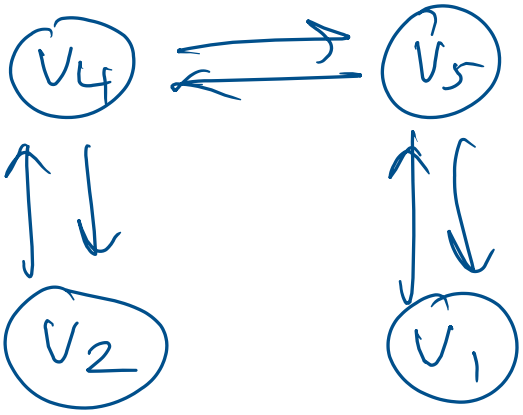
$V_5 = 11$

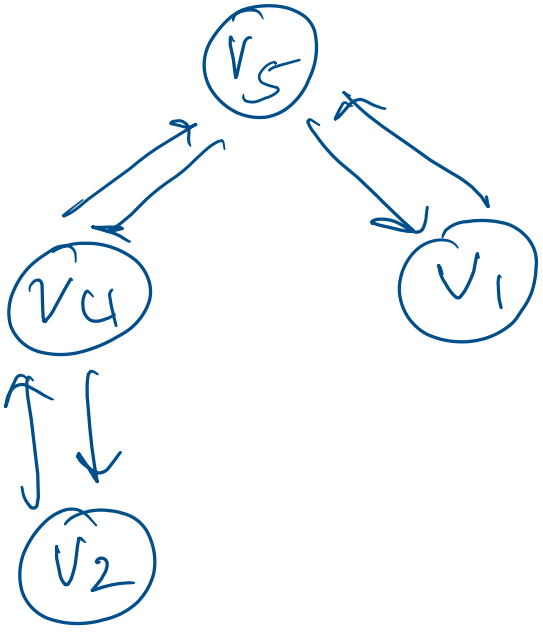


Removing V_3 :→ (Extract_min)

$X = X + V_3$.


v_2
v_4
v_5
v_1


v_4
v_5
v_2
v_1


v_5
v_4
v_1
v_2

New weighted degree after extract min:→

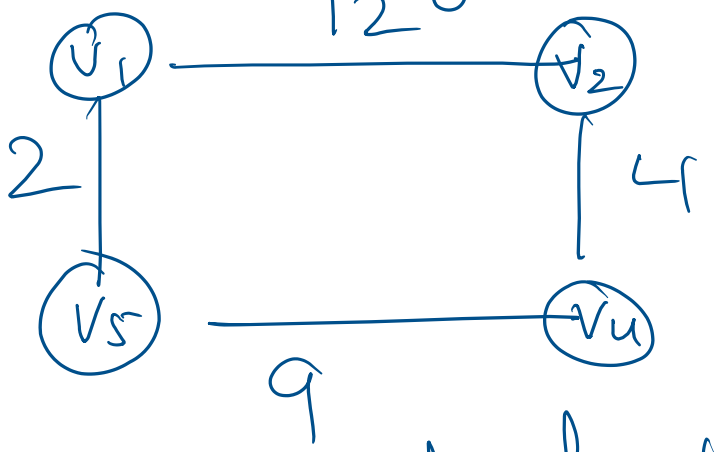
$V_4 = 13$

$V_2 = 16$

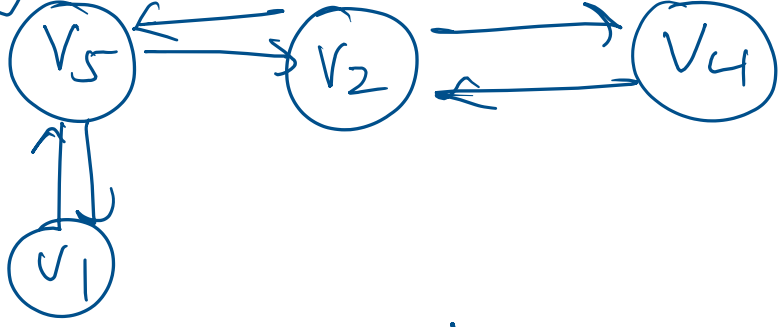
$V_1 = 14$

$V_5 = 11$

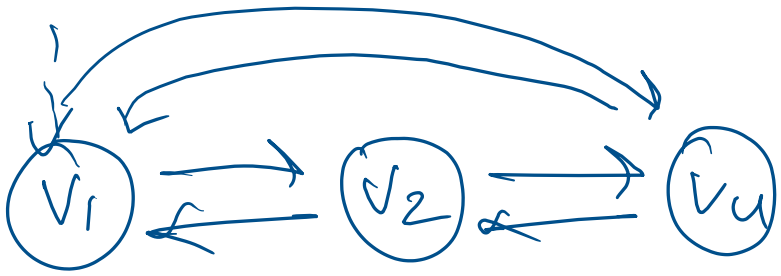
New graph:→

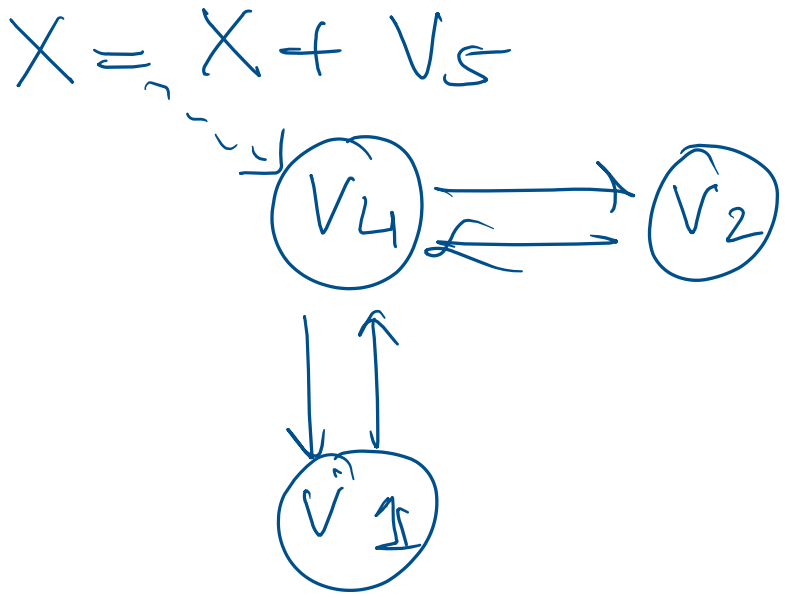


After decrease-key & cut operation new →



Now, running extract-min again we remove V_5 and consolodate the Heap:



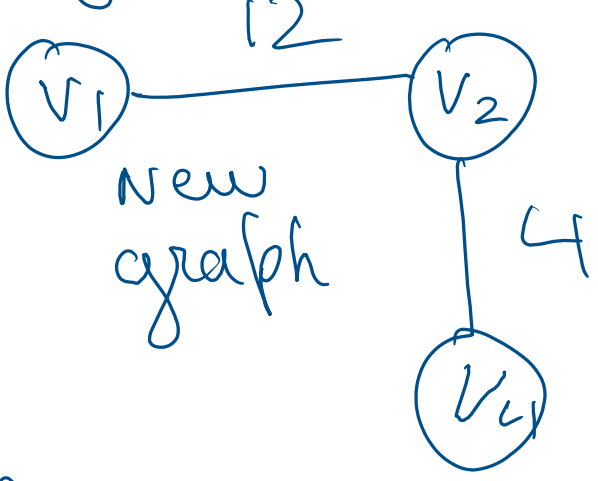


New weighted degree after extract min :→

$V_4 = 4$

$V_2 = 16$

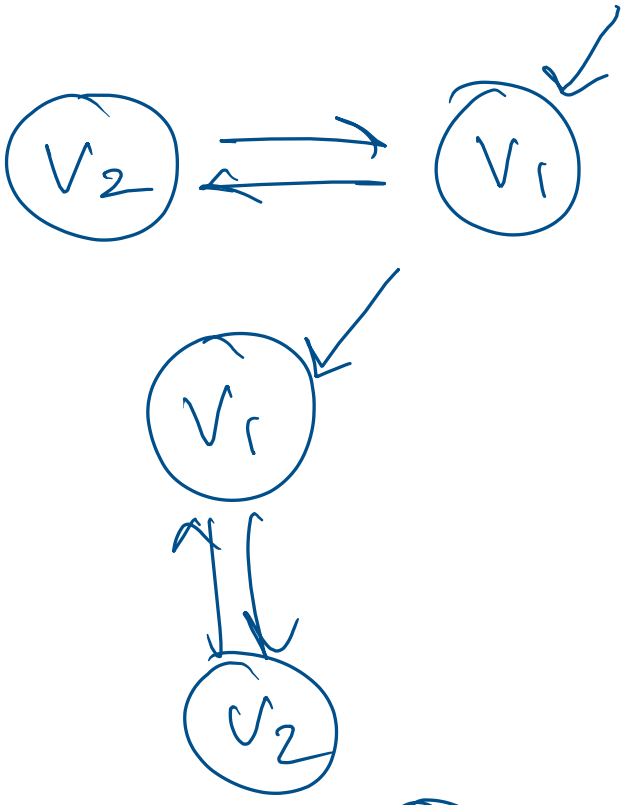
$V_1 = 12$



Decrease-key operation :→

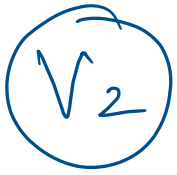
Extracting the min-key V_4 & consolidating the graph.

$X = X + V_4$



Now extracting V_1 from the graph it. The only node left is V_2

$$X = X + V_1$$



Now decreasing the key of V_2 & extracting it out.

$$X = X + V_2.$$

So, the inductivity in this case would be printed as follows

$\max \langle v_2, v_1, v_4, v_5, v_3 \rangle$

So, the result will be 12 in this scenario.

Now if we take any other permutation, it will be greater than 12.

eg. $\langle v_5, v_4, v_1, v_2, v_3 \rangle$

$= \max \langle 9, 16, 2, 9 \rangle$

$= \underline{16}$.

Now, computing the running time:

So, we know that each extract-min method take $O(\lg V)$ time.

So, we are running extract min for exactly |V| times.

∴ Total time for extract_min is :→

$O(|V| \log |V|)$.

Now, as we are computing the weighted degree, we are traversing to the adjacent edges of each vertex. So, the running time for that computation will be $O(E)$ ∴ The total running time = $O(|E| + |V| \log |V|)$.

Proof of correctness :→

I am going to prove the correctness of the algorithm using the induction method

Base:-

If there is one element then we return that element itself.

Inductive step :→

$V_K \rightarrow$ elements

V_{K+1} will be the smallest element of $n - V_K$ array.

Hence, using the greedy approach the next element we will take is the smallest element.

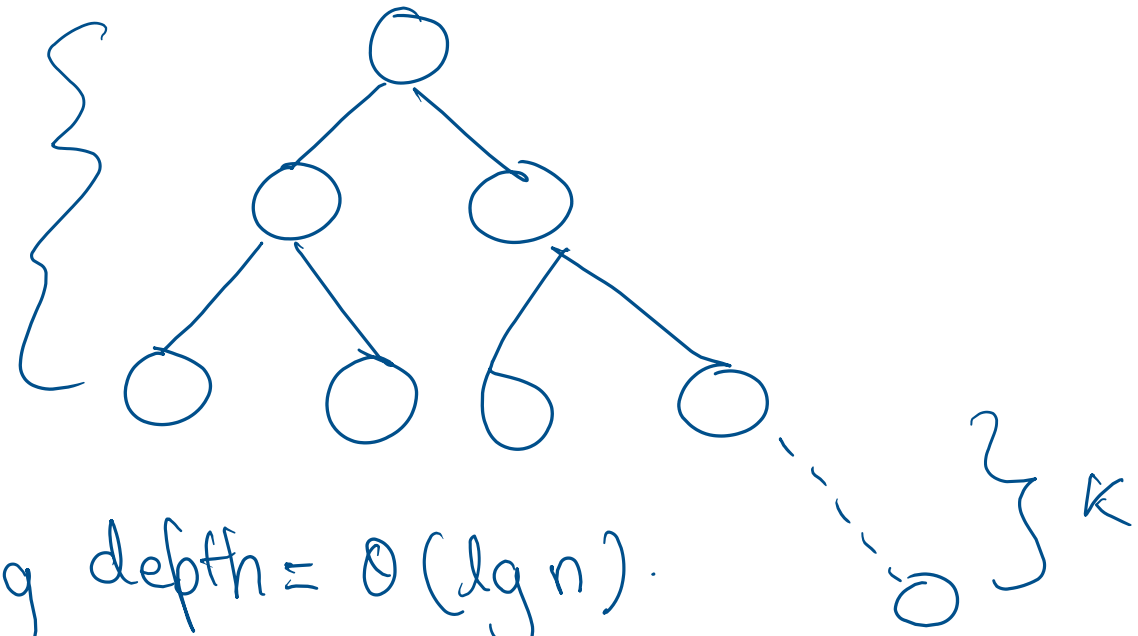
PART - 2:→

So, in this part we can just use custom heap & take an array of size $E+1$, with A[i] pointing to a doubly linked list. To

find a vertex of minimum degree we start with A[0] & go till the point A[c] is empty. As, we are going to traverse the whole graph one at a time using an approach of DFS in this case. So, the running time of DFS is $O(|E|+|V|)$; Hence proved.

P.T.O.

Question 2:→



Assuming depth = $O(\lg n)$.

So, the first part I will prove it by contraction. The height of a balanced tree is $O(\lg n)$. If we add K-nodes to the right subtree, the height of the tree would increase from $O(\log n)$ to $O(\log n + K)$ K being the tail from the right subtree. which clearly shows the height h would be greater than $O(\log n)$ being $O(\log n + K) > O(\log n)$.

So, now proving that after adding the chain of nodes the average depth remains $O(\lg n)$. As, I have calculated height of the tree B ie $\kappa = \sqrt{n \lg n}$.
we calculate the depth in upper bound :→

$$\frac{1}{n} \cdot (\text{Depth}(B) + \text{Depth}(A))$$

from figure X.

$$= \frac{1}{n} O\left(\sqrt{n \lg n}\left(\lg n + \sqrt{n \lg n}\right) + \left(n - \sqrt{n \lg n}\right) \lg n\right)$$

$$= \frac{1}{n} O\left(\cancel{\sqrt{n \lg n} \cdot \lg n} + n \lg n + n \lg n - \cancel{\sqrt{n \lg n} \cdot \lg n}\right)$$

$$= \frac{1}{n} O(2n \lg n) \Rightarrow \underline{O(\lg n)}.$$

Now, we calculate the same in lower bound :→

$$\frac{1}{n} \cdot (\text{Depth}(B) + \text{Depth}(A))$$

$$\frac{1}{n} \cdot \omega\left(\sqrt{n \lg n}\left(\lg n + \sqrt{n \lg n}\right) + \left(n - \sqrt{n \lg n}\right) \lg n\right)$$

$$\frac{1}{n} \cdot \omega\left(\sqrt{n \lg n} \cdot \lg n + n \lg n + n \lg n - \lg n \sqrt{n \lg n}\right)$$

$$\frac{1}{n}\, \omega(2n \lg n) \Rightarrow \omega(\lg n)$$

Average both is both $O(\lg n)$ & $\omega(\lg n)$ ie eventually $\Theta(\lg n)$

This eventually shows that if the height of the tree is increased by k which is $\sqrt{n \lg n}$. Total height being $O(\lg n + \sqrt{n \lg n})$ which is eventually $\geq O(\lg n)$ but as shown above the depth of the tree remains $O(\lg n)$.

P.T.O.

Part B :→

There are n nodes in BST
Avg. depth : $O(\lg n)$.
Height : h.
So, from node at 0 .... n there is a path.

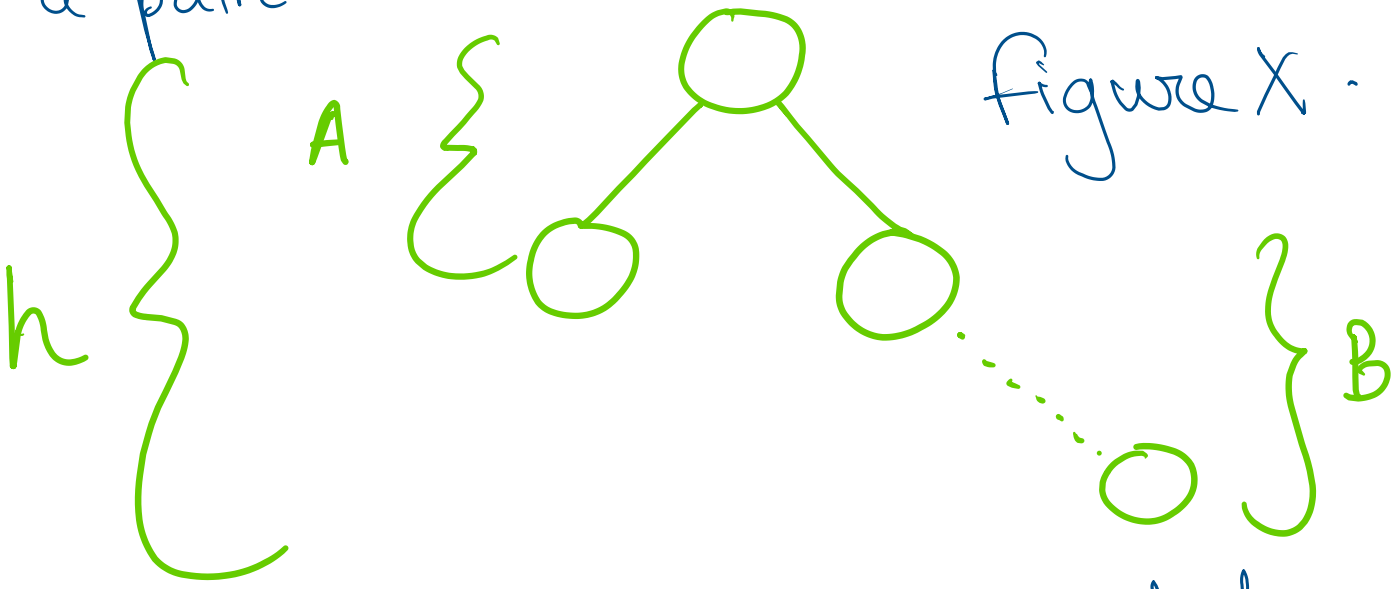


figure X.

So, now calculating the depth we know that for depth of any given node we take $\sum_{x=0}^{h} x \rightarrow \frac{h(h+1)}{2}$

$$= \frac{h^2}{2}$$

$$\text{Avg. Depth} = \frac{h^2}{2} \times \frac{1}{n}$$

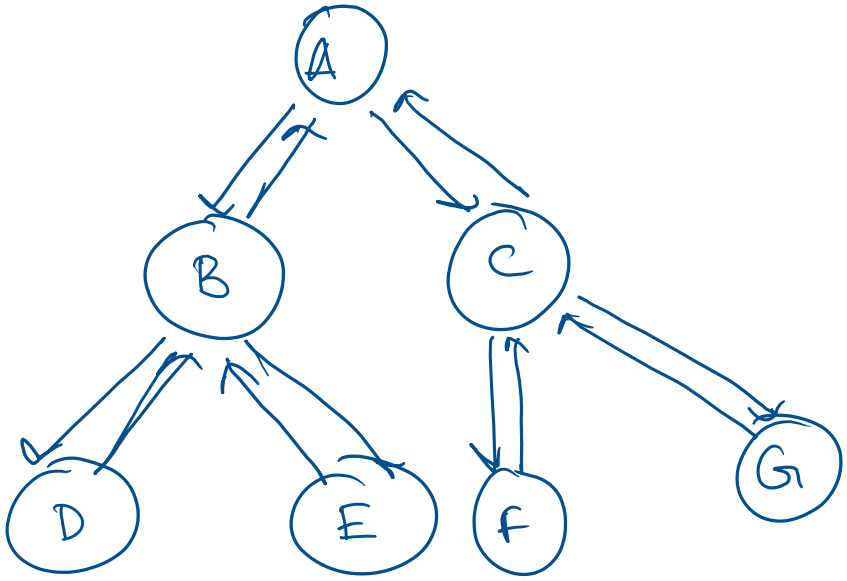
$$\therefore O(\log n) = \frac{h^2}{2n}$$

Taking 2 out & sending n to the other side we get

$$n \log n = h^2$$

$$h = \sqrt{n \log n}$$

Question 3:→



So, to find the successor of a given node. The pseudo-code is as follows:→

```
1  SUCCESSOR (A):
2      if (A.right ≠ NIL)
3          while A → left ≠ NIL
4              A = A → left
5          return A.
6      B = A → parent
7      while (B ≠ NIL and A == B.right):
8          A = B
9          B = B.parent
10     return B.
```

So, in the worst case scenario the running time of the above pseudo code is $O(h)$. In this case, we will travel from the right most node to the root node as shown in the example:→

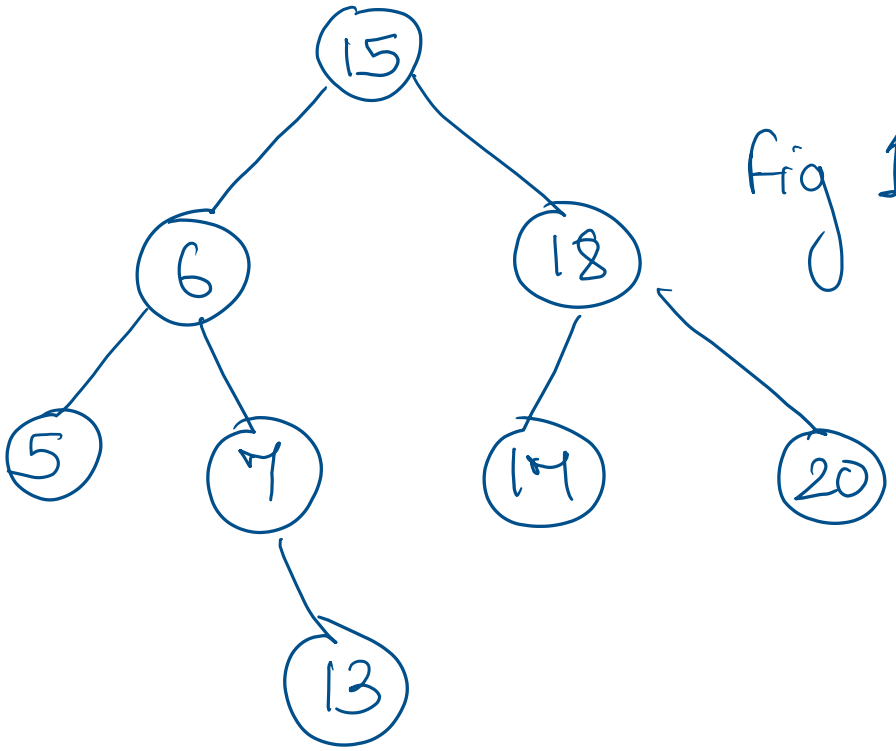


Fig 1

So, the successor of 13 by using the above algorithm will be 15. i.e traversing the whole height of the tree in the worst case.

Part - B :→

So, for analyzing the running time of each successive calls, we look at the word case scenario first. So, in the worst case as shown

in the figure 1, to find the successor of 13, which would be 15 (root) & to find the successor of 15 we go down the right subtree & that would be 17 (successor). So, if we look at the worst case we will traverse the height of the tree twice ie 2·h. And now if we look at edges, so each edge in the graph will traverse at most of 2 times. So, if we make s successive calls so in the worst case each edge in the

tree will cross at most 2 times. till we find the successor.
So, using the amortized analysis we can take the constant 2 out from height as well as successive edge call (s) we will get O(s+h). We take 2 out as it is the constant cost that has been spend in the worst case scenario. It cannot cross 2 in any case.

Question 4:→

As the hint says, we use a pointer to point to the high order 1.

So, we are going to take a pointer P to point to the high order 1.

So, for INCREMENT we initially are going to set the pointer as -99

So, a pseudo-code for increment is:

INCREMENT (ARRAY): (Algo from book)

```
i = 0
P = -99
while (i < ARRAY.LENGTH and A[i] == ):
        A[i] = 0
        i = i + 1
if (i < ARRAY.LENGTH):
        A[i] = 1
```

if $i > P$

$P = i$

else

$P = -99$.

In, the above algorithm, we are just going to set the pointer value to the index which has the highest degree order of 1. Now, while resetting the binary counter to 0, we will traverse right of the high order bit & set it back to 0.

```
RESET (ARRAY):
    WHILE (P>0):
        A[P] = 0
        P = P - 1
    P = -99
```

So, for the amortized analysis of this problem, I am going to use the credit method. In the credit method we pay 1 credit for flipping the bit to 1 and one more credit to flipping the bit back to 0. This is normally what happens in the binary counter increment problem. In this case we have a 3rd credit as well for the case in which we traverse through the array & check the bit which has already been turned to 0. So, the total of 3 credits comes into play. As, we have only used constant values at each increment of 1 and even resetting it, the amortized cost is constant

as each increment operation just requires flipping one bit ie setting only one single bit to 1., the amortized for resetting is 0 as we have already taken care of that case & we are not setting any bits to 1. So, this proves that the cost of this increment and reset operation is O(n) as in the worst case we will traverse the whole array of n bits with a constant credit ie 3 → O(3n), which eventually comes to O(n).

Example to illustrate the algo :→

0 0 0 0
+ 1
---
0 0 0 1 ← P = 1
+ 1

0 0 1 0

↳ P = 2.

And so on for insertion, now setting the bits back to 0, we go right from P till all is 0; P in this case is 2. I am not putting any credits for updating & resetting the pointer P because that will always be a constant value.