Name : Snehi Jain
Student ID: 801198603

H.W. 1 (Algo & Data Structure)

Q.1 - The given algorithm calculates: (a^n)

- It takes 2 input a, n and returns b.

- For e.x:

  For values (2, 5) we will get $2^5 = 32$.

- Since, algorithm is dividing k by 2 each time if its even that makes run time as logn.

- For cases, where k is subtracted by 1 can be ignored because the occurence is very low.

- Therefore, the worst case running time complexity will be O(logn).

Q.2

$T_A(n) = 0.1n^2 \log_{10} n$ &

$T_B(n) = 2.5n^2$

Algorithm B is better in the Big O sense. Since,

$$O(\log n) < O(n) < O(n \log n) < O(n^2)$$

$$0.1n^2 \log_{10} n \leq 2.5n^2$$

$$0.1 \log_{10} n \leq 2.5$$

$$\boxed{\log_{10} n \leq 25}$$

This equation is only possible when $10^{25} \leq n$.

Therefore, For input size greater than 10^{25} Algo B will always outperform Algo A.

If problem size is $n \leq 10^9$, Algo A is better. and recommended to use.

Q. 3

We can use a stack to reverse a Queue because it uses LIFO system. (Last in First Out).

We can dequeue and push the element in the stack till the Queue's empty.
Then, we can enqueue the Queue using pop().
That's how we get reverse Queue.

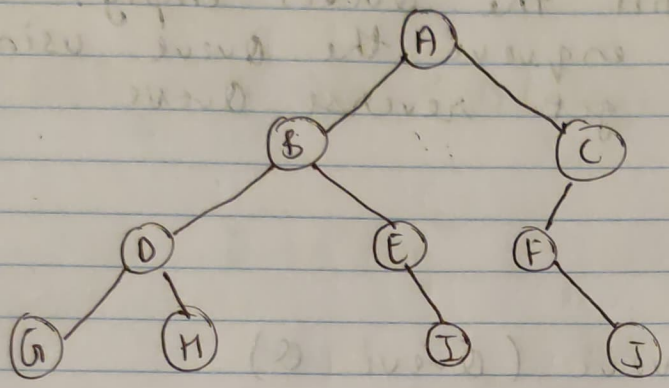
Pseudo code:

```
func revQueue (Queue Q)
    Stack S = Stack()
    While (!Q.isEmpty()) do
        S.push (Q.dequeue())
    While (!S.isEmpty()) do
        Q.enqueue (S.pop())
```

This is a linear-time algorithm.

Q.4 Preorder - A B D G H B E I C F J
Inorder - G D H B E I A F J C

Binary Tree,



Q.5

Stack follows the LIFO system. which can be developed using both single and doubly linked list.

I will choose singly linked list as it is less complex.

In doubly linked list, we need to handle two variables, head and tail, which is bad in the terms of memory and complexity.

We ideally want a list that is fast and cheap.

Therefore, we will use singly linked list to implement a stack.

Q.6 Least common ancestor (LCA)

Input - v, w

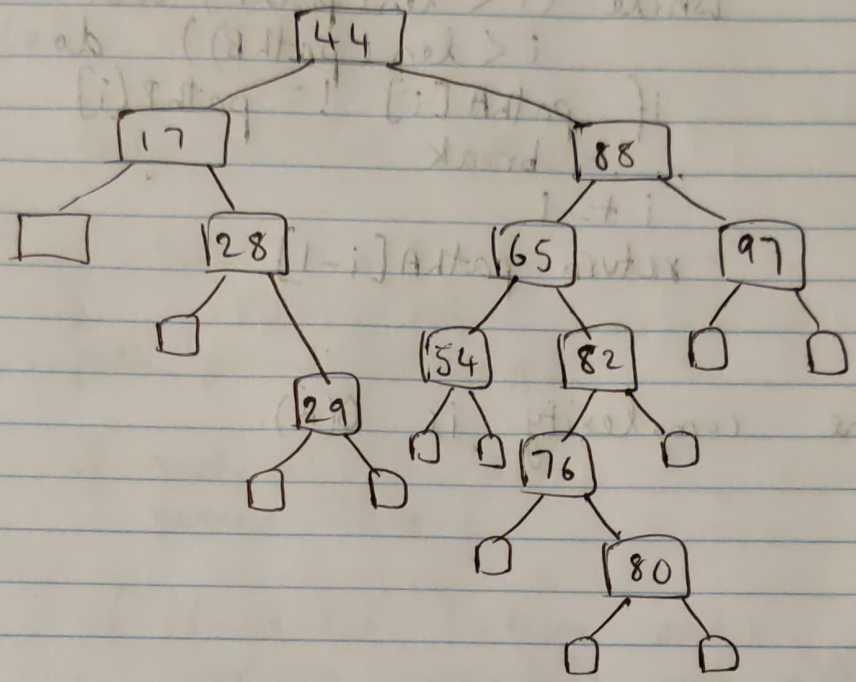
Pseudo code -

```
func findpath (root, path, v)
    if root is None
        return FALSE
    path.append (root.value)

    if root.value == v
        return TRUE

    if ((root.left != None and
        findpath (root.left, path, v)) or
       (root.right != None and
        findpath (root.right, path, v)))
       return TRUE

    path.pop()
    return FALSE

func findLCA (root, v, w)
    pathA = []
    pathB = []
    if (! findpath (root, pathA, v) or
        ! findpath (root, path, w))
        return FAILURE
```

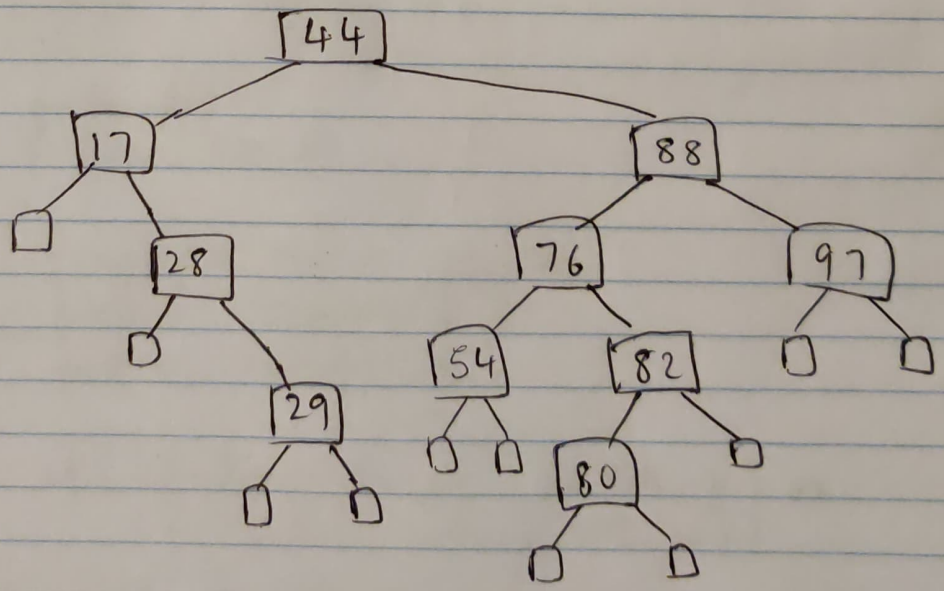
```
i = 0
while (i < len(pathA) and i < len(pathB)) do
    if pathA[i] != pathB[i]
        break
    i += 1
return pathA[i-1]
```

Time complexity is O(n).

Q.7 Tree -

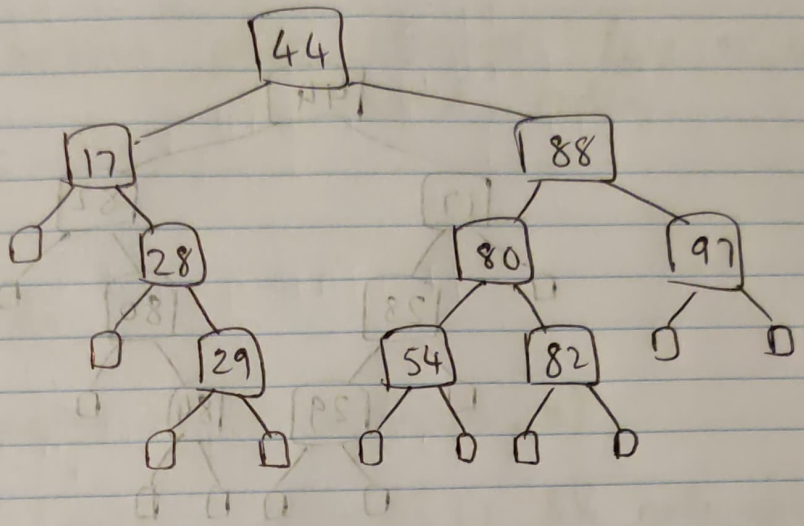


Remove 65,

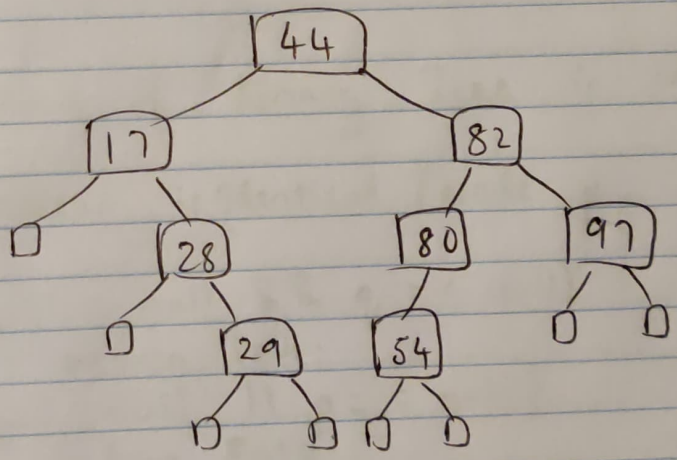


Rep Put the inorder successor after deletion then sort the tree.

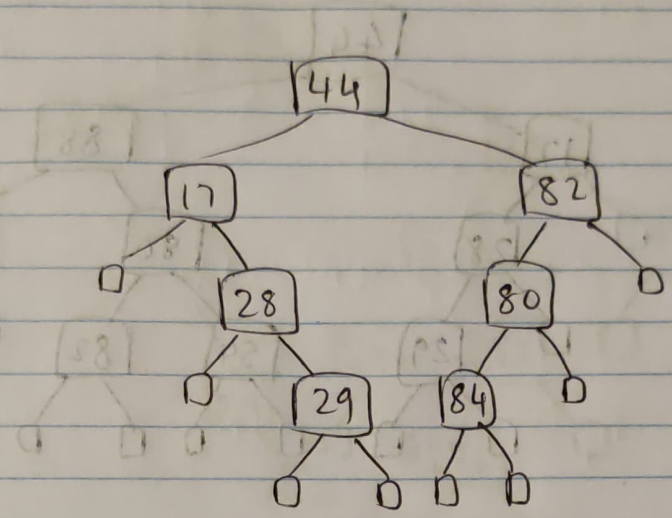
Remove 76,



Remove 88,



Remove 97,



Q.8

Input : x
Output : ~~return~~ print elements less or equal to x in sorted order.
Complexity : O(K)

Pseudo code:

```
Func smallerKeyAlgo (node, x)

    if (node.isInternal() && node.value() <= x)
        print node.value()
        smallerKeyAlgo (node.left(), x) &&
            smallerKeyAlgo (node.right(), x)
```

Q.9

Code to check if Binary Tree is identical or not.

```
public boolean isIdentical (Node p, Node q) {

    if (p == null && q == null)
        return true;
    if (q == null || p == null)
        return false;
    if (p.val != q.val)
        return false;

    return isIdentical (p.right, q.right) &&
        isIdentical (p.left, q.left);
}
```