**SET A (Solution)**

SEC –A

Q.1 **What is a Priority Queue?**

Priority queue is a data structure in which the intrinsic ordering of the elements does determine the results of its basic operations. Ascending and Descending priority queue are the two types of Priority queue.

Q.2 **How do you test for an empty queue?**

To test for an empty queue, we have to check whether READ=HEAD where REAR is a pointer pointing to the last node in a queue and HEAD is a pointer that pointer to the dummy header.

In the case of array implementation of queue, the condition to be checked for an empty queue is READ<FRONT.

Q.3 **Define recursion?**

It is a technique and it can be defined as any function that calls itself is called recursion. There are some applications which are suitable for only recursion such as, tower of Hanoi, binary tree traversals etc, can be implementing very easily and efficiently.

Q.4  **What is the need for the header?**

Header of the linked list is the first element in the list and it stores the number of elements in the list. It points to the first data element of the list.

Q.5 **How to search an element in list.**

 Searching can be initiated from first node and it is compared with given element one after the other until the specified key is found or until the end of the list is encountered.

SEC-B

Q.1 **Write down the algorithm for solving Towers of Hanoi problem?**

1. Push parameters and return address on stack.
2. If the stopping value has been reached then pop the stack to return to previous level else move all except the final disc from starting to intermediate needle.
3. Move final discs from start to destination needle.
4. Move remaining discs from intermediate to destination needle.
5. Return to previous level by popping stack.

Q.2 write a short note on binary search and counting sort.

Solution:

**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

[**Counting sort**](http://en.wikipedia.org/wiki/Counting_sort) is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Q.3 Define Dqueue? Create a Dqueue two insert the data from both side but deletion can be perform one side.

**Solution:**

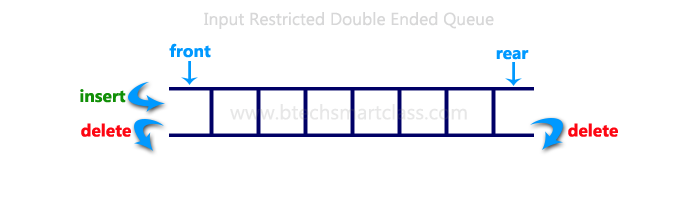
Dqueue is also data structure where elements can be inserted from both ends and deleted from both ends. To implement a dqueue operations using singly linked list operations performed insert\_front, delete\_front, insert\_rear, delete\_rear and display functions

ouble Ended Queue can be represented in TWO ways, those are as follows...

1. Input Restricted Double Ended Queue
2. Output Restricted Double Ended Queue

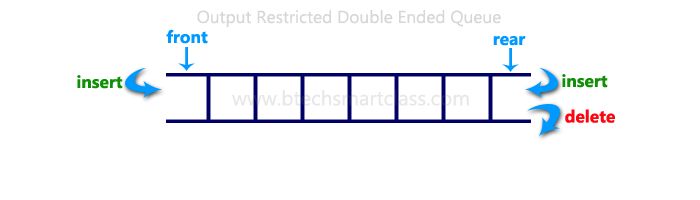
**Input Restricted Double Ended Queue**

In input restricted double ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



**Output Restricted Double Ended Queue**

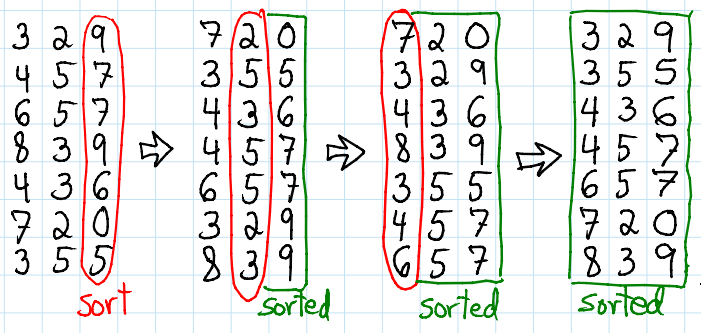
In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



Q.4 Define radix sort? With any example.

**Radix sort** is an integer [sorting algorithm](https://brilliant.org/wiki/sorting-algorithms/) that sorts data with integer keys by grouping the keys by individual digits that share the same significant position and value ([place value](https://brilliant.org/wiki/place-value/)). Radix sort uses [counting sort](https://brilliant.org/wiki/counting-sort/) as a [subroutine](https://brilliant.org/wiki/subroutines/) to sort an [array](https://brilliant.org/wiki/arrays/) of numbers. Because integers can be used to represent [strings](https://brilliant.org/wiki/strings/) (by [hashing](https://brilliant.org/wiki/hashing/?wiki_title=hashing) the strings to integers), radix sort works on data types other than just integers. Because radix sort is not comparison based, it is not bounded by for running time — in fact, radix sort can perform in linear time.

Radix sort incorporates the counting sort algorithm so that it can sort larger, multi-digit numbers without having to potentially decrease the efficiency by increasing the range of keys the algorithm must sort over (since this might cause a lot of wasted time).



Q.5 how to implement multiple stack using single array?

Solution:

# efficiently implement k stacks in a single array:

**Method 1 (Divide the array in slots of size n/k)**   
A simple way to implement k stacks is to divide the array in k slots of size n/k each, and fix the slots for different stacks, i.e., use arr[0] to arr[n/k-1] for first stack, and arr[n/k] to arr[2n/k-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

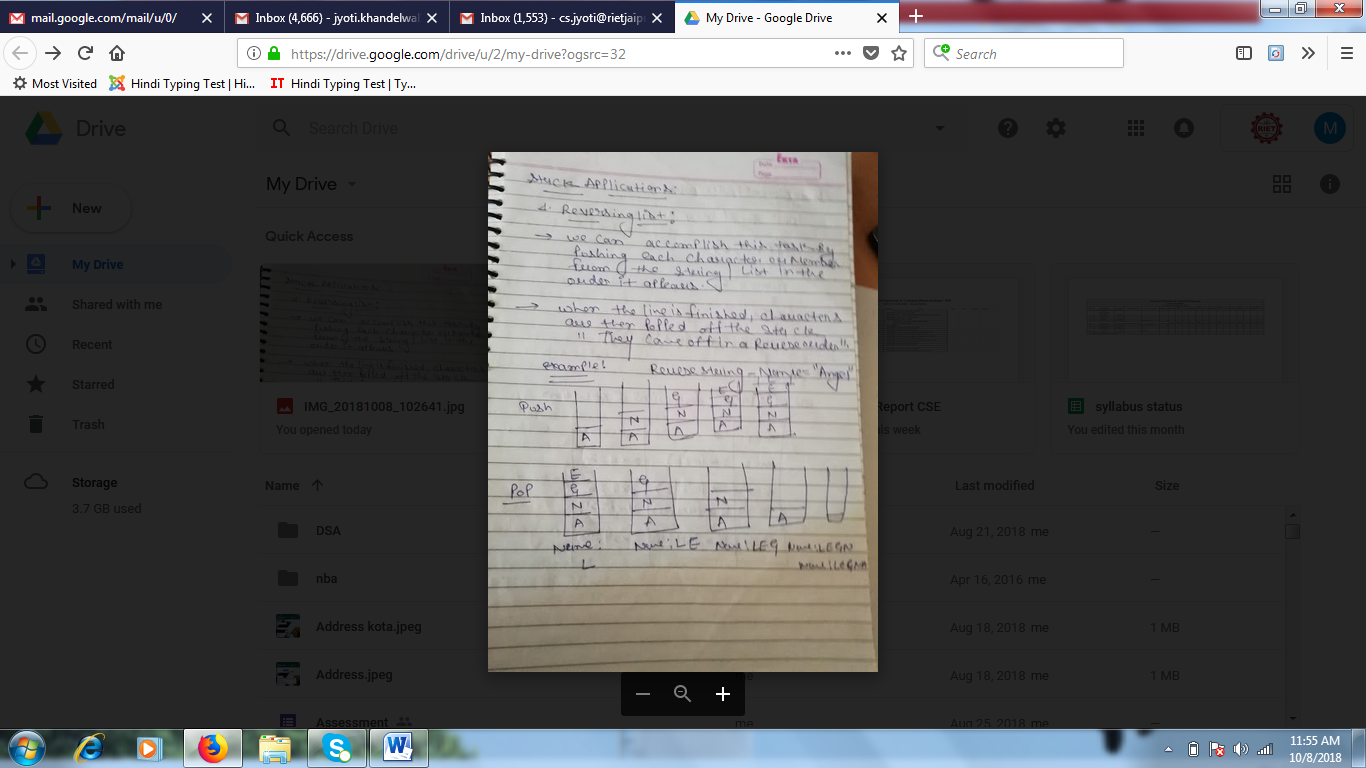
The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the k is 2 and array size (n) is 6 and we push 3 elements to first and do not push anything to second second stack. When we push 4th element to first, there will be overflow even if we have space for 3 more elements in array.

**Method 2 (A space efficient implementation)**  
The idea is to use two extra arrays for efficient implementation of k stacks in an array. This may not make much sense for integer stacks, but stack items can be large for example stacks of employees, students, etc where every item is of hundreds of bytes. For such large stacks, the extra space used is comparatively very less as we use two *integer* arrays as extra space.

Following are the two extra arrays are used:  
***1) top[]:*** This is of size k and stores indexes of top elements in all stacks.  
***2) next[]:*** This is of size n and stores indexes of next item for the items in array arr[]. Here arr[] is actual array that stores k stacks.

# Q.6 what is the application of stack for reversing a list.:

# Solution:



**Sec-C**

Q.1 Circular queue is to be implemented using an array of 10 elements write a pseudo code for implementation of inserting an element in queue and checking whether queue is empty or not.

Solution:

**Implementation of Circular Queue**

To implement a circular queue data structure using array, we first perform the following steps before we implement actual operations.

* **Step 1:** Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2:** Declare all **user defined functions** used in circular queue implementation.
* **Step 3:** Create a one dimensional array with above defined SIZE (**int cQueue[SIZE]**)
* **Step 4:** Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)
* **Step 5:** Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

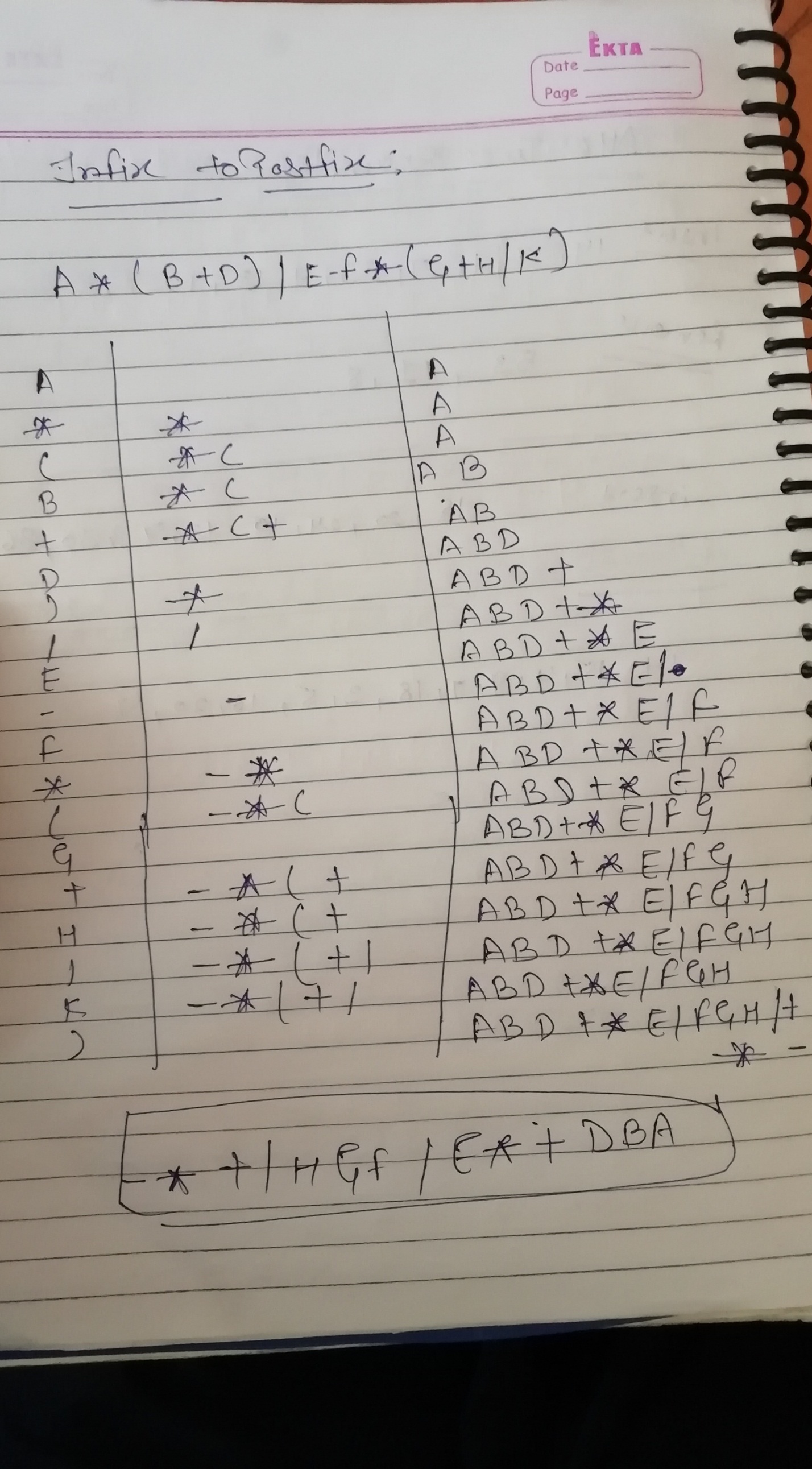
**enQueue(value) - Inserting value into the Circular Queue**

In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...

* **Step 1:** Check whether **queue** is **FULL**. (**(rear == SIZE-1 && front == 0) || (front == rear+1)**)
* **Step 2:** If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3:** If it is **NOT FULL**, then check **rear == SIZE - 1 && front != 0** if it is **TRUE**, then set **rear = -1**.
* **Step 4:** Increment **rear** value by one (**rear++**), set **queue[rear]** = **value** and check '**front == -1**' if it is **TRUE**, then set **front = 0**.

Q.2 Translate, by using stack each infix expression in its equivalent postfix expression also explain the algorithm :

A\*(B+D)/E-F\*(G+H/K)



Q.3 Write an algorithm to perform the following operations in singly linked list

1. To count no of nodes in linked list

**Step 1:** Count = 0  
           SAVE = FIRST  
**Step 2:** Repeat step 3 while SAVE ≠ NULL  
**Step 3:** Count= Count + 1  
           SAVE=SAVE->LINK  
**Step 4:** Return Count

OR

1. To Create and traverse a linked list

Beginning from the head,

1. check, if the end of a list hasn't been reached yet;
2. do some actions with the current node, which is specific for particular algorithm;
3. current node becomes previous and next node becomes current. Go to the step 1