ASSIGNMENT 1

PART I

Implement a stack/queue using an array. Interface is given at the end. Use generics in Java/C++ to make sure that you can use the stack/queue for any type of data. You may assume that the number of elements inserted into the stack/queue never exceeds 100,000. Do **NOT** use the inbuilt Stack/Queue class of java/C++.

PART II

We have two queues, the input queue Q1 and the output queue Q2, and one stack S. We are only allowed to dequeue from Q1 and we are only allowed to enqueue into the output queue Q2. We are allowed to both push into and pop from the stack.

Consider that we have the numbers 1 2 3 4 5 6 enqueued in Q1. Suppose we perform the following sequence of operations.

enqueue(Q2, dequeue(Q1))
push(S, dequeue(Q1))
enqueue(Q2, dequeue(Q1))
push(S, dequeue(Q1))
enqueue(Q2, pop(S))
enqueue(Q2, pop(S))
enqueue(Q2, dequeue(Q1))
enqueue(Q2, dequeue(Q1))

then the output queue contains 1 3 4 2 5 6

This is an example of what we call a *stack permutation* i.e. A stack permutation is a ordering of numbers from 1 to n that can be obtained from the initial ordering 1, 2, ... n by a sequence of stack operations as described above.

To clarify this, note that 1 5 3 4 2 6 is *not* a stack permutation. Intuitively this is because to enqueue 5 into Q2 after 1 we would have to push 2 3 4 into the stack which would then be output in the order 4 3 2, not in the order 3 4 2.

In this assignment you will be given a permutation of n numbers and asked to check if it is a stack permutation or not. The TA will give you the number n as input and will give you a permutation. If it is a stack permutation your program will have to return the sequence of operations that formed that permutation. If it is not a permutation, your program will have to say it is not a permutation, and will have to give the reason why (as given above).

Stack Interface/**

```
 * Interface for a stack: a collection of objects that are inserted
 * and removed according to the last-in first-out principle.  This
 * interface includes the main methods of java.util.Stack.
 *
```

```
  * @author Roberto Tamassia
  * @author Michael Goodrich
  * @see EmptyStackException
  */

public interface Stack<E> {
 /**
  * Return the number of elements in the stack.
  * @return number of elements in the stack.
  */
  public int size();
 /**
  * Return whether the stack is empty.
  * @return true if the stack is empty, false otherwise.
  */
  public boolean isEmpty();
 /**
  * Inspect the element at the top of the stack.
  * @return top element in the stack.
  * @exception EmptyStackException if the stack is empty.
  */
  public E top()
    throws EmptyStackException;
 /**
  * Insert an element at the top of the stack.
  * @param element to be inserted.
  */
  public void push (E element);
 /**
  * Remove the top element from the stack.
  * @return element removed.
  * @exception EmptyStackException if the stack is empty.
  */
  public E pop()
    throws EmptyStackException;
}
```

Queue Interface

```
public interface Queue<E> {
 /**
  * Returns the number of elements in the queue.
  * @return number of elements in the queue.
  */
  public int size();
```

```
 /**
  * Returns whether the queue is empty.
  * @return true if the queue is empty, false otherwise.
  */
  public boolean isEmpty();
 /**
  * Inspects the element at the front of the queue.
  * @return element at the front of the queue.
  * @exception EmptyQueueException if the queue is empty.
  */
  public E front() throws EmptyQueueException;
 /**
  * Inserts an element at the rear of the queue.
  * @param element new element to be inserted.
  */
  public void enqueue (E element);
 /**
  * Removes the element at the front of the queue.
  * @return element removed.
  * @exception EmptyQueueException if the queue is empty.
  */
  public E dequeue() throws EmptyQueueException;
}
```

References

http://www.cse.iitd.ac.in/~subodh/courses/CSL201/Assignment1.htm
http://www.cse.iitd.ernet.in/~tripathy/CSL201_11-12/assign2.html

Assignment 3

You are the accountant of a big company and have been handed the task of maintaining a database of salaries of all the people working in the company. Whenever a person joins or leaves the company, you will need to update the database. Also, at times the manager may ask you for some statistics on the data. Multiple employees could have the same salary. You need to design and implement a data structure which supports the following operations (note x and y are positive integers) :

$\boldsymbol{A}(\boldsymbol{x})$: Add salary x to the database

$\boldsymbol{R}(\boldsymbol{x})$: Remove salary x from the database if it exists, otherwise return the salary closest to x

$\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{y})$: Return the number of employees having salary in the range $[x, y]$

Max: Return the maximum salary

Min: Return the minimum salary

If the current number of distinct salaries in the company is n, queries A, R, Q should not take more than $O(log\ n)$ time. All the salaries are integers in the range $[1,10^6]$. You should initialize the database by reading salaries from a file. The first line of the file is an integer which specifies the number of data points. Subsequent lines contain the salaries (one on each line).

The test cases will consist of a sequence of the operations, A, R, Q, Max, Min and after each operation you should report the relevant information.