A. **LONG DIVISION:**

1. Provide pseudo-code for the Long Division algorithm of a dividend \( E \) by a divisor \( D > 0 \) represented as arrays of digits (base ten) similar to algorithm 1 (grade school addition) and algorithm 2 (grade school multiplication). Your algorithm may terminate when the remainder gets smaller than \( D \). At this point, it should output both the quotient \( Q \) and the remainder \( R \) of division. **Example:**

```
<table>
<thead>
<tr>
<th>E</th>
<th>1</th>
<th>7</th>
<th>2</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Q</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

2. Provide pseudo-code for the Long Division algorithm of a remainder \( R \) by a divisor \( D > R \) represented as arrays of digits (base ten). Your algorithm should output both the tail \( T \) and loop \( L \) as described above.

If you consider the sequence of remainders \( R_1=R, R_2,... \) occurring during the division of \( R \) by \( D \), the loop part extends between the first and second appearances of the earliest repeating remainder, while the tail part is before the loop. More precisely, if \( i \) is the least index for which there exists an index \( j < i \) such that \( R_i=R_j \), then \( T = T_1,...,T_{j-1} \) is produced while handling \( R_1,...,R_{j-1} \) and \( L = L_1,...,L_{i-j} \) is produced while handling \( R_j,...,R_{i-1} \). **Example:**

```
<table>
<thead>
<tr>
<th>R</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
</tr>
</tbody>
</table>
```
3. Assess the running time of the algorithm you provided in part 2 as a function of the size of the input $R,D$ (measured in digits). For that purpose establish the maximum size of $T$ and $L$ as a function of the input-size. Argue that this representation is not a very good idea if you wish to represent rational numbers with absolute precision.

4. Consider the problem you encountered in part 2: your algorithm receives a list of integers, one at a time, $R_1, R_2, \ldots$ and it should find the first time an $R_i$ comes up that has occurred previously, i.e. find the least $i$ such that all the $R_1, R_2, \ldots, R_{i-1}$ are distinct but $R_i = R_j$ for some $1 \leq j < i$.

Describe an $O(i \log i)$ time algorithm to find the least such $i$. (The Master Theorem may come handy in establishing the running-time of your algorithm.)

Note — running time is not a function of the global number of elements but only a function of the position of the first repetition. Your algorithm does not need to stop at the $i^{th}$ integer but total time is limited to $O(i \log i)$.

Hint: Combine mergesort, and binary search…

5. The exact situation in part 4 is slightly better than the general case: in the calculation of the remainders, not only will $R_i = R_j$ for some $1 \leq j < i$, but from that point on $R_{i+k} = R_{j+k}$ for all $k \geq 0$. Floyd defined a linear-time algorithm to find, in this context, the least $i$ such that all the $R_1, R_2, \ldots, R_{i-1}$ are distinct but $R_i = R_j$ for some $1 \leq j < i$. Let $R_1 = R$ be the initial remainder.

$$T=H=1 \quad (T \text{ stands for tortoise and } H \text{ for hare})$$

REPEAT
  $T = T+1; \ H = H+2$
UNTIL $R_T = R_H$

$T=1$
REPEAT
  $T = T+1; \ H = H+1$
UNTIL $R_T = R_H$
REPEAT
  $T = T+1$
UNTIL $R_T = R_H$
RETURN $T$

Prove that this algorithm will return the correct value of $i$ and that it does so in linear time (with respect to $i$).