1. (20 points) Suppose we are given two polynomials $p(x)$ and $q(x)$ of degree $n - 1$ and \( m - 1 \) respectively, \( n \geq m \), and suppose we want to find \( p(x)/q(x) \). Consider the following algorithm:

**Algorithm** Div-Poly\((p, q)\)

1. Let \( n' \) be the smallest power of 2 greater than \( n - 1 \).
2. Use the FFT to compute \( y = \text{DFT}_{n'}(p) \), and \( z = \text{DFT}_{n'}(q) \).
3. Compute the \( n' \)-dimensional vector \( \langle y_0/z_0, y_1/z_1, \ldots, y_{n'}/z_{n'} \rangle \).
4. Compute \( a = \text{DFT}_{n'}^{-1}(\langle y_0/z_0, y_1/z_1, \ldots, y_{n'}/z_{n'} \rangle) \),
   and return this as the coefficient vector for \( p(x)/q(x) \).

Discuss the circumstances under which this procedure will correctly reconstruct \( p(x)/q(x) \),
considering the case when there is a polynomial to represent \( p(x)/q(x) \), and also the case
when \( p(x)/q(x) \) is only representable as an infinite series.

2. (a) (10 points) State the CountingSort algorithm in pseudocode and explain informally how it works. What is the running time of CountingSort?

(b) (10 points) Explain how CountingSort sorts the following list of keyed objects.
The first column contains the key, and the second column contains the object data
(in this case just a letter).

<table>
<thead>
<tr>
<th>key</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
</tr>
</tbody>
</table>
(c) (5 points) Explain what a comparison based sorting algorithm is and name two such algorithms. Moreover, state the lower bound on the running time of comparison based deterministic sorting algorithms. Explain in detail why this lower bound does not apply to CountingSort.

(d) (5 points) Let \( n \) denote the number of objects and let \( m \) denote the number of keys.

For what ratios \( m/n \) is CountingSort faster in the worst case than Quicksort if Quicksort chooses the last element of the input list as the pivot?

(e) (10 points) Let \( x_1, \ldots, x_n \) be (not necessarily distinct) integers in the range \( \{1, \ldots, n\} \).

Let \( (a_1, b_1), \ldots, (a_k, b_k) \) be pairs of integers. Furthermore, for any \( 1 \leq i \leq k \) let \( Z_i \) be the number of indices \( j \) such that \( a_i \leq x_j \leq b_i \). Devise an algorithm that, given \( x_1, \ldots, x_n \) and \( (a_1, b_1), \ldots, (a_k, b_k) \), computes the numbers \( Z_1, \ldots, Z_k \) in time \( O(n + k) \).

3. In this question we consider the Matrix Chain Multiplication problem, where we are given a sequence of \( n \) rectangular matrices \( A_1, \ldots, A_n \), and we are concerned with finding the parenthesisation of \( A_1 \cdot \ldots \cdot A_n \) which uses the fewest number of arithmetic multiplications. We assume that the dimensions of matrices \( A_1, \ldots, A_n \) are described by a sequence \( p = \langle p_0, p_1, \ldots, p_n \rangle \) of natural numbers, such that for every \( 1 \leq i \leq n, A_i \) is a \( p_{i-1} \times p_i \) matrix. The product \( A_1 \cdot A_2 \cdot \ldots \cdot A_n \) is therefore well-defined (the number of columns of \( A_i \) is equal to the number of rows of \( A_{i+1} \), for every \( 1 \leq i \leq n - 1 \)).

Throughout this question, we will assume that the multiplication of two matrices (either rectangular or square) is always done via the naive algorithm, meaning that the arithmetic multiplications taken to multiply two matrices of dimensions \( p \times q \) and \( q \times r \) respectively is \( pqr \).

Below is the Matrix-Chain-Order algorithm, a dynamic programming algorithm which computes the optimal parenthesisation of a sequence of rectangular matrices with dimension sequence \( p = \langle p_0, p_1, \ldots, p_n \rangle \).

\textbf{Algorithm} Matrix-Chain-Order\((p)\)

1. \( n \leftarrow p.length - 1 \)
2. \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( n \) \textbf{do} \( m[i, i] \leftarrow 0 \)
3. \textbf{for} \( \ell \leftarrow 2 \) \textbf{to} \( n \) \textbf{do} \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( n - \ell + 1 \) \textbf{do}
4. \( j \leftarrow i + \ell - 1 \)
5. \( m[i, j] \leftarrow \infty \)
6. \textbf{for} \( k \leftarrow i \) \textbf{to} \( j - 1 \) \textbf{do}
7. \( q \leftarrow m[i, k] + m[k + 1, j] + p_i p_k p_j \)
8. \textbf{if} \( q < m[i, j] \) \textbf{then}
9. \( m[i, j] \leftarrow q \)
10. \( s[i, j] \leftarrow k \)
11. \textbf{return} \( s \)
(a) **(10 points)** Consider the sequence \( p = (20, 30, 15, 30, 40), \) describing a sequence of four matrices with dimensions \( 20 \times 30, 30 \times 15, 15 \times 30, 30 \times 40 \) respectively. Apply the Matrix-Chain-Order algorithm to obtain the optimal parenthesisation of the sequence of matrices. You must use the algorithm above, and must show your workings.

(b) **(10 points)** Show that the running-time of Matrix-Chain-Order is \( O(n^3) \) and also that it is \( \Omega(n^3) \).

(c) **(10 points)** An alternative approach to find the optimal parenthesisation of \( A_1 \cdot A_2 \ldots A_n \) would be to consider every possible parenthesisation, compute its cost, and take the minimum of all such cost values. Note that the number of possible parenthesisations \( P(n) \) of \( n \) matrices satisfies the following recurrence:

\[
P(n) = \begin{cases} 
1 & \text{if } n = 1 \text{ or } n = 2 \\
\sum_{\ell=1}^{n-1} P(\ell) \cdot P(n-\ell) & \text{if } n > 2 
\end{cases}
\]

Prove (giving specific \( c, n_0 \) values) that \( P(n) = \Omega(2^n) \). (Which means that this alternative approach is infeasible.)

(d) **(10 points)** Now suppose we were given only one square matrix \( A \), of dimensions \( k \times k \), and are asked to compute \( A^n \). For this case, all parenthesisations of \( A^n \) will have the same cost, this being \( (n-1)k^3 \).

Devise an algorithm for evaluating \( A^n \) which runs in time \( O(k^3 \lg(n)) \). Justify the correctness of your algorithm, and the \( O(k^3 \lg(n)) \) time.