1. Create a class Rational.

- Create a constructor with two parameters.
- Overload the operators +, -, *, /, ==, !=, <, >.
- Override the methods Equals, GetHashCode, and ToString.

2. Make the Rational class to inherit the IComparable or ICompare interface.

- Implement the CompareTo method of the IComparable interface or the Compare method of the ICompare interface.

3. Create your own exception class RationalException representing errors that occur during application working with rational numbers.

4. Make changes in the Relational class as follows:

- The constructor with two parameters throws the RationalException if the denominator of the rational number is equal to zero or both the nominator and denominator are negative numbers (suppose that the nominator carries the sign of the rational number).
- Write a default constructor that enters values for the nominator and denominator from the keyboard; rethrows an exception back up the call stack if the user enters incorrect integer values for the nominator and denominator; throws the RationalException if the denominator of the rational number is equal to zero or both the nominator and denominator are negative numbers.

5. Create a driver class that tests the operations with rational numbers.

- Add a static method with three parameters: left operand, right operand and operation, that executes the permit operations with rational numbers and throws the RationalException if the operation is not permitted.
- Test all operations with rational numbers.
- Enter an array of rational numbers and sort it in an increasing order using the Array.Sort method.