**Problem #1 (Marks: 7)**

**Sieve of Eratosthenes**

In mathematics, the *sieve of Eratosthenes* is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2.

To find all the prime numbers less than or equal to a given integer *n* by Eratosthenes' method, implement the following steps:

1. Create a global integer array of *n*+1 elements. Call it *primeFlag.* (Assume n <= 10000)
2. Initialize another local integer variable *p* to 2, the smallest prime number.
3. Starting from 2*p*, enumerate the multiples of *p* by counting to *n* in increments of *p*, and set the value of those positions in the *primeFlag* array to 1. These are composite values; since they are multiples of a prime *p*. (Note that position *p* itself should not be marked).
4. Find the first number *q* greater than *p* (but less than or equal to *n*, obviously) such that *primeFlag*[*q*] is 0. If there was no such number, stop. Otherwise, set *p* equal to *q,* which is the next prime; and repeat from step 3.

Implement the above logic in a method called sieve(). Once sieve() is called, you can easily find whether a given integer *r* is prime or not, by checking *primeFlag*[*r*]. Accordingly, implement a method called *isPrime* to detect whether an integer is prime or not.

.

**Problem #2 (Marks: 3)**

**Prime Factors**

A prime factor of a positive integer is a prime that divides that number. Given an integer, you need to enumerate all its prime factors. Use the methods implemented in the above problem and necessary main() method to solve this task. Your input will be an integer *n* such than 1 < *n* < 108. You need to output the prime factors of the integer. See the samples for exact formatting

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| 85000  234  7 | 2 5 17  2 3 13  7 |