Design and Analysis of Algorithms

Implementation Project

Number Breaking

An Algorithmic Approach

By

IMT2019064 Pratik Ratnadeep Ahirrao

IMT2019084 Shrey Tripathi

1 Problem Statement

Every number is multiplication of some prime numbers. A prime number is a number which is only divided by 1 and itself. Here, we are given a number n. We have to find the prime numbers whose multiplication makes this number.

For example, 12 is multiplication of prime numbers 2 and 3. 15 is multiplication of 3 and 5.

Input: First line contains the number of test cases T. Then, each line contains a single integer n.

Constraints: $1 \leq T \leq 1000$, $2 \leq n \leq 1000000$.

Output: For each test case, we print a single line which contains the test case number and the prime numbers in ascending order separated by a single space whose multiplication makes this number.

Example:

Input:

```
3
12
42
84
```

Output:

```
Case 1 : 2 3
Case 2 : 2 3 7
Case 3 : 2 3 7
```

2 Approach

We use the Sieve of Eratosthenes to find all the prime numbers which divide the given n and are less than or equal to n.

This can be done as follows:-

1. We first create a boolean array $isPrime$ of size $n+1$ that checks whether a given number in the range $[0, n]$ is prime or not.

2. We initialize all indices of the array $isPrime$ to $true$, except indices 0 and 1, since $isPrime[0] = false$, and $isPrime[1] = false$.

3. Looping the index (say p), from 2, to $\sqrt{n}$, we mark all the multiples of those indices as $false$, since they are not prime, and the indices themselves as $true$, since they are prime.
   After marking all the multiples of p, it takes the value of the smallest prime number greater than p that is not marked.

4. If no such number exists, we stop. Otherwise, p now will be this new number. We repeat the same above process for this new p.

5. Finally we print the prime numbers which divide n.

Note: It is sufficient to mark the numbers in step 3 starting from p^2, as all the smaller multiples of p will have already been marked at that point. This means that the algorithm can be terminated when p^2 is greater than n.

3 Pseudocode

```
let $isPrime$ be an array of Boolean values, indexed by integers 2 to $n$.
Let there be $Output$ list where we store our results.
Set all values of $isPrime$ to $True$ initially.

Function Sieve($n$):
    for $i \leftarrow 2$ to $\sqrt{n}$ do
        if $isPrime[i] == True$ then
            for $j \leftarrow i^2, i^2 + i, i^2 + 2i, i^2 + 3i, .$ to $n$ do
                $isPrime$[i] == False
            end
        end
        for $p \leftarrow 2$ to $n$ do
            if $isPrime[i] == True$ & $n$ % $p == 0$ then
                Add $p$ to the $Output$ list.
            end
        end
    end
return Output
```

4 Time complexity

The overall complexity of the algorithm is $\mathcal{O}(n \log \log n)$.

We can observe that the algorithm will perform $\frac{n}{p}$ operations for every prime $p \leq n$ in the inner loop.

So, the number of times the loop runs is equal to

$$\tfrac{n}{2} + \tfrac{n}{3} + \tfrac{n}{7} + ...$$

that is,

$$n(\tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{7} + ...)$$

Now, the sum of reciprocals of primes $p \leq n$ asymptotically equals $\mathcal{O}(\log\log n)$. [Reference here] Hence, the overall complexity of the algorithm as $\mathcal{O}(n \log\log n)$.

5 Proof of correctness

- We have to prove that the *Output* array contains only prime numbers, all of which are factors of n. This can be proved using contradiction.
- Assume that a particular number in the *Output* array is marked as $false$, i.e. it is non-prime (or composite). Since it is composite, it must have been marked as a multiple of some other smaller prime number according to the algorithm.
- It implies that it should not be present in the output array.
- This contradicts the fact that it is present in the output array. Hence, our assumption was wrong.

Note: Some numbers may be marked more than once.
Note: There is also a linear time algorithm for Sieve of Eratosthenes. Although this running time of $\mathcal{O}(n)$ is better than $\mathcal{O}(n \log\log n)$ of the classic Sieve of Eratosthenes, the difference between them is not so big. In practice that means just double difference in speed, and the optimized versions of the sieve run as fast as this algorithm.

6 Example

Suppose n equals 25.
We generate a list of numbers from 2 to 25.

$$2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25$$

We start with the first number in the list i.e. 2, and cross out every multiple of 2 in the list except 2.

$$2\ 3\ \cancel{4}\ 5\ \cancel{6}\ 7\ \cancel{8}\ 9\ \cancel{10}\ 11\ \cancel{12}\ 13\ \cancel{14}\ 15\ \cancel{16}\ 17\ \cancel{18}\ 19\ \cancel{20}\ 21\ \cancel{22}\ 23\ \cancel{24}\ 25$$

Now, next uncrossed number in the list after 2 is 3. We cross out every multiple of 3 in the list except 3.

$$2\ 3\ \cancel{4}\ 5\ \cancel{6}\ 7\ \cancel{8}\ \cancel{9}\ \cancel{10}\ 11\ \cancel{12}\ 13\ \cancel{14}\ \cancel{15}\ \cancel{16}\ 17\ \cancel{18}\ 19\ \cancel{20}\ \cancel{21}\ \cancel{22}\ 23\ \cancel{24}\ 25$$

Repeating the above process for the remaining elements in the list, we get:

$$2\ 3\ 5\ 7\ 11\ 17\ 19\ 23$$

Hence, $\{2, 3, 5, 7, 11, 17, 19, 23\}$ are the prime numbers that are less than n.

From this list, if a prime number divides n, we add it to the *Output* array

Hence, the final output is :

$$\{5\}$$

7 References

`https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`