Prime Factorization

"God may not play dice with the universe, but something strange is going on with the prime numbers."

```
Question:
--------------
1. You are required to display the prime factorization of a number.
2. Take as input a number n.
3. Print all its prime factors from smallest to largest.

Input format
--------------
n, an integer


Output format
---------------
p1  p2  p3  p4.. all prime factors of n


Constraints
---------------
2 <= n < 10 ^ 9
```

In number theory, integer factorization is the decomposition of a composite number into a product of smaller integers. If these factors are further restricted to prime numbers, the process is called prime factorization.

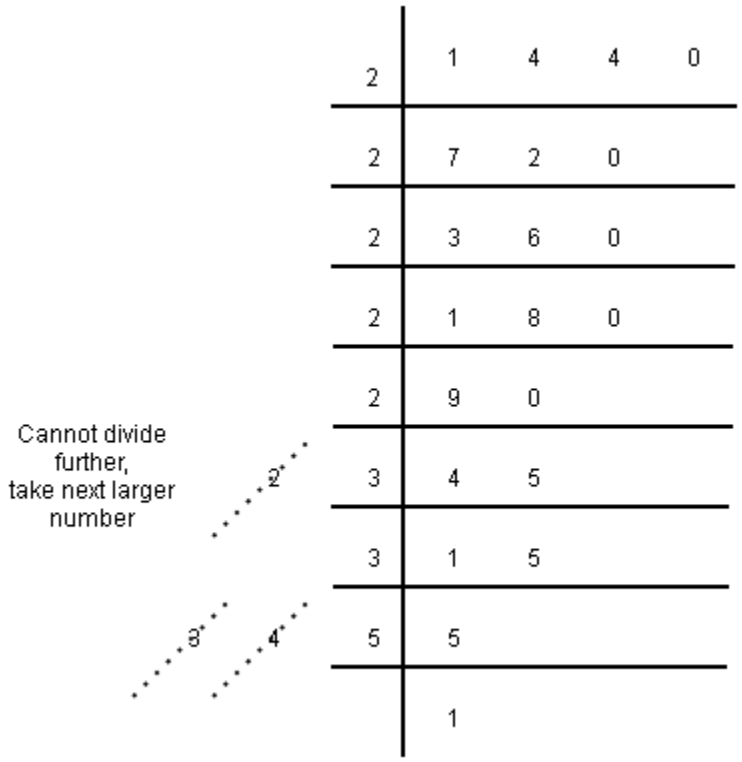
Solution Approach:

We divide the given number by the smallest number and keep on dividing till it no longer could be divided by that number.

As soon as that happens, we check the divisibility of the current state of the given number with the next greater number as divisor and continue this process until the given number becomes 1.

Basically, "*We are dividing the number till it could be divided (Decomposing the number into smaller numbers)*".

For instance, taking prime factorization of 1440:



The prime factors are: 2, 2, 2, 2, 2, 3 and 5.

Programming Implementation for the same:

```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        for(int div= 2; div <= n; div++){
            while(n % div == 0){
                System.out.print(i + " ");
                n = n / div;
            }
        }
    }
}
```

But we can further optimize this code as we know from finding prime numbers, that we do not need to iterate the loop till n. Rather, iterating till $\sqrt{n}$ or conditioning $div^2 < n$ would be sufficient because if there are two integers P and q satisfying the equation $P * q = n$.

It is not possible for $P > \sqrt{n}$ and $q > \sqrt{n}$ at the same time, and we make use of this fact.

Modifying the earlier code accordingly, we optimize the time complexity.

```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        for(int div= 2; div*div <= n; div++){
            while(n % div == 0){
                System.out.print(div + " ");
                n = n / div;
            }
        }
    }
}
```

However, there still exists a drawback to this optimization.

What if a factor lies beyond the $\sqrt{n}$ range?

Well the answer to this query is that, yes there can be a factor beyond $\sqrt{n}$ but there exists only "One" factor beyond this range.

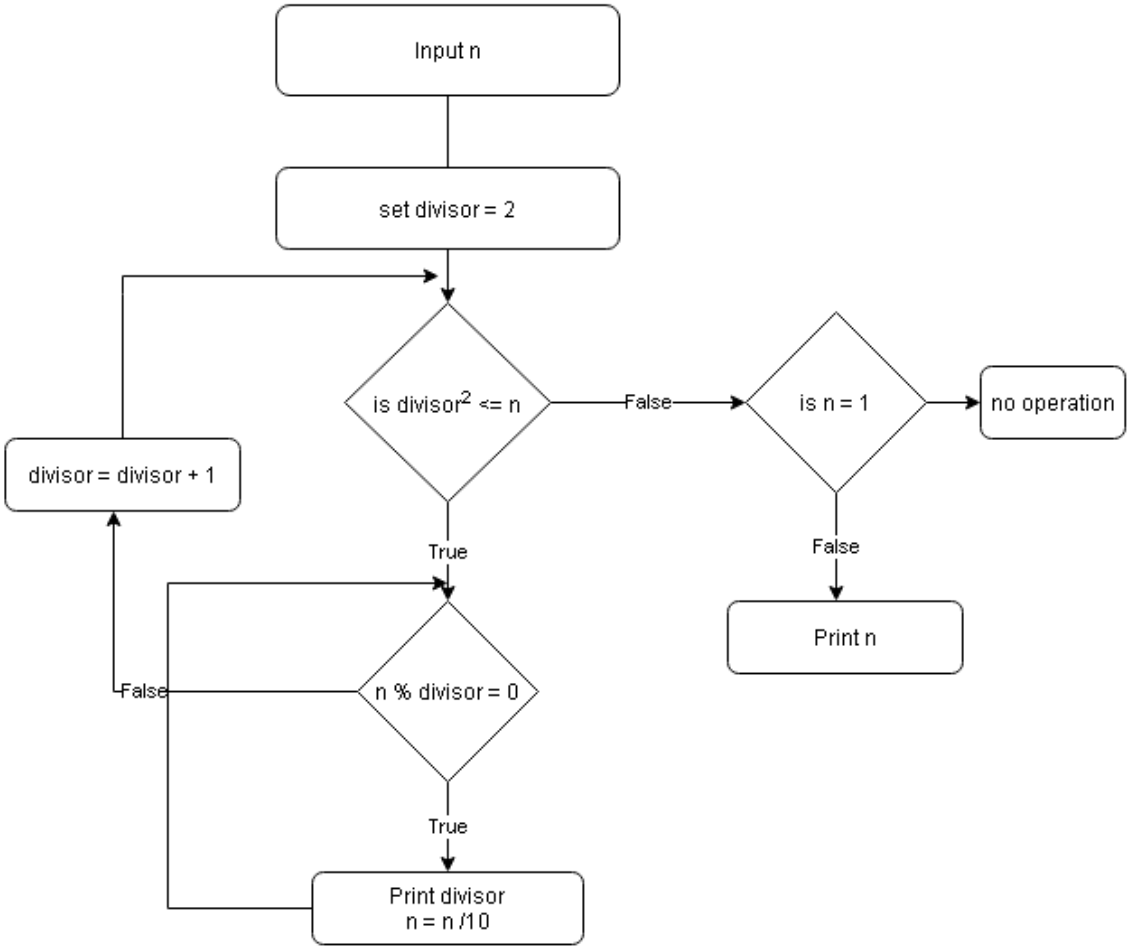
Take the example of 46:

After the first division with 2, we are left with 23, which is quite clearly out of the range of $\sqrt{n}$.

But it is also the only factor that remains. So, we can conclude that after the division process is over and the initial given number to us does not become zero, the current state of the number will be our last prime factor.

With this last modification to our code, we handle all the cases for this problem.

Algorithm:



Final solution code:

```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        for(int div= 2; div*div <= n; div++){
            while(n % div == 0){
                System.out.print(div + " ");
                n = n / div;
            }

        }
         if(n != 1)
            {
                System.out.print(n);
            }
    }
}
```