Lazy Evaluation

Lazy evaluation is an evaluation strategy that delays the assessment of an expression until its value is needed.

Ruby 2.0 introduced a lazy enumeration feature. Lazy evaluation increases performance by avoiding needless calculations, and it has the ability to create potentially infinite data structures.

Example:

```
power_array = -> (power, array_size) do
   1.upto(Float::INFINITY).lazy.map { |x| x**power }.first(array_size)
end

puts power_array.(2 , 4)    #[1, 4, 9, 16]
puts power_array.(2 , 10)   #[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
puts power_array.(3, 5)     #[1, 8, 27, 64, 125]
```

In this example, `lazy` avoids needless calculations to compute *power_array*.
If we remove lazy from the above code, then our code would try to compute all x ranging from 1 to *Float::INFINITY*.
To avoid timeouts and memory allocation exceptions, we use `lazy`. Now, our code will only compute up to *first(array_size)*.

Task

Your task is to print an array of the first N palindromic prime numbers.
For example, the first 10 palindromic prime numbers are $[2, 3, 5, 7, 11, 101, 131, 151, 181, 191]$.

Input Format

A single line of input containing the integer N.

Constraints

You are not given how big N is.

Output Format

Print an array of the first N palindromic primes.

Sample Input

```
5
```

Sample Output

```
[2, 3, 5, 7, 11]
```