Problem 5

What is the smallest number divisible by each of the numbers 1 to 20?

Given the small size of the parameter, k = 20, this problem can easily be solved without any programming, as it is equivalent to finding the least common multiple of the set of integers {2, 3, 4, … , 20}. However, our analysis will lead in a different direction with the aim of developing an efficient algorithm capable of solving for larger values of k.

Let N be the smallest number that is divisible by all the integers from 2 to k. For N to be the smallest value with this property we must ensure that in its prime factorisation it does not contain any more prime factors than is absolutely necessary.

Consider the first three cases of k:

k = 2, N = 2
k = 3, N = 2*3 = 6
k = 4, N = 2*3*2 = 12

We can see that for k = 4 we did not need to evaluate 2*3*4, because one of the 2's in the prime factorisation of 4 = 2*2 was already included. If we now consider the next two cases:

k = 5, N = 2*2*3*5 = 60
k = 6, N = 2*2*3*5 = 60

We can see that N = 60 for both k = 5 and k = 6, because if N contains the factors 2 and 3 it already contains everything necessary for it to be divisible by 6.

Applying this principle for the case k = 20:

N = 2 * 3 * 2 * 5 * 7 * 2 * 3 * 11 * 13 * 2 * 17 * 19 = 232792560

So how do we solve this programmatically?

Let p[i] be the i th prime number: p[1] = 2, p[2] = 3, p[3] = 5, … .
Let N = p[1]^a[1] * p[2]^a[2] * p[3]^a[3] * … ; initially, let a[i] = 0 for all i.
For j = 2 to k, write each j in the form, p[1]^b[1] * p[2]^b[2] * p[3]^b[3] * … and set a[i] = max(a[i], b[i]).

Let us observe this algorithm with a trace for k = 10.

j	b[1]	b[2]	b[3]	b[4]	a[1]	a[2]	a[3]	a[4]
2	1				1			
3		1			1	1		
4	2				2	1		
5			1		2	1	1	
6	1	1			2	1	1	
7				1	2	1	1	1
8	3				3	1	1	1
9		2			3	2	1	1
10	1		1		3	2	1	1

Hence N = 2^3 * 3^2 * 5^1 * 7^1 = 2520.

However, this approach requires a function to be constructed to express a given number as a product of prime factors. So we shall consider an alternative approach.

Let us consider the case of finding the least value of N for k = 20. We know that N must be divisible by each of the primes, p[i], less than or equal to k. But what determines the exponent, a[i], in the prime factorisation of N is the greatest perfect power of p[i] that is less than or equal to k. For example, as 2^4 = 16 and 2^5 = 32, we know that a[1] = 4 as no other numbers, 2 ≤ j ≤ 20, can contain more than four repeated factors of 2. Similarly 3^2 = 9 and 3^3 = 27, so a[2] = 2. And for p[i] ≥ 5, a[i] = 1.

Hence N = 2^4 * 3^2 * 5 * 7 * 11 * 13 * 17 * 19 = 232792560.

For a given p[i] we can determine a[i] in the following way.

Let p[i]^a[i] = k. By "logging" both sides: a[i] log(p[i]) = log(k).
So a[i] = log(k) / log(p[i]).
But as a[i] must be integer, a[i] = floor(log(k) / log(p[i])).

For example, when k = 20, the exponent of the first primes factor, p[1] = 2, will be:
a[1] = floor(log(20) / log(2)) = floor(4.32…) = 4

To optimise the approach even further we note that a[i] = 1 for p[i]^2 > k. In other words, we only need to evaluate a[i] for p[i] ≤ sqrt(k).

To put all this together into an algorithm we shall assume that an array of primes p[1], p[2], p[3], … has already been created.

```
k = 20
N = 1
i = 1
check = true
limit = sqrt(k)
while p[i] <= k
      a[i] = 1
      if check then
            if p[i] <= limit then
                  a[i] = floor( log(k) / log(p[i]) )
            else
                  check = false
            end if
      end if
      N = N * p[i] ^ a[i]
      i = i + 1
end while
output N
```

It must be noted that the practical limit of this algorithm is the capability of your programming language in handling the size of N as k increases.