1 Problem Statement

The prime 41, can be written as the sum of six consecutive primes: $41 = 2+3+5+7+11+13$ This is the longest sum of consecutive primes that adds to a prime below one-hundred. The longest sum of consecutive primes below one-thousand that adds to a prime, contains 21 terms, and is equal to 953. Which prime, below one-million, can be written as the sum of the most consecutive primes?

2 Code

```
# Project Euler #50 | Sidharth Baskaran | 01/09/2022

import time
import math

def is_prime(n):
    for x in range(math.ceil(math.sqrt(n)),1,-1):
        if n % x == 0:
            return False
    return True

def get_prime_list(n):
    # boolean array
    A = [True] * (n + 1)
    # sieve of Erastothenes
    i = 2
    while i**2 <= n:
        if A[i]:
            for j in range(i**2, n + 1, i):
                A[j] = False
        i += 1

    # get list of primes
    primes = []
    for i in range(len(A)):
        if A[i]:
            primes.append(i)
    return primes[2:]

def solve(n):
    # get the largest sublist possible
    prime_list = get_prime_list(n)
    s = 0
    max_sub_len = 0
    for x in prime_list:
        if s >= n:
            break
        s += x
        max_sub_len += 1

    # iterate through sublist-offset combinations
    offset = 0
    max_len = 0
    best_sum = s

    while max_sub_len - offset > max_len:
        curr_len = max_sub_len - offset
        while curr_len > max_len:
            curr_sum = sum(prime_list[offset:offset+curr_len])
            if is_prime(curr_sum):
                max_len = curr_len
                best_sum = curr_sum
            curr_len -= 1
        offset += 1

    print(best_sum)

if __name__ == "__main__":
    s = time.time()
    solve(1000000)
    e = time.time()

    print('%.3fms' % ((e-s)*1000))
```

3 Explanation

The algorithm generates a list of primes below n (1×10^6 in this case) starting with 2 using the Sieve of Erastothenes, as outlined in the method `get_prime_list`. A boolean array of length $1 \times 10^6 + 1$ is initially defined with all true values. The sieve then begins by iterating from $i = 2$ to $i = \sqrt{n}$ in the outer loop, and the inner loop marks every ith array location after i^2 false, as defined by the algorithm. Finally, we select all integers corresponding to remaining true values in the array; these constitute a list of prime numbers.

Before entering the `solve` method, we take a quick look at `is_prime`, a method which iterates through all numbers less than $\sqrt{n}$ (excluding 1), checking for divisibility. We only have to check from $\sqrt{n}$ and below because at least one factor of a nonprime number is below its square root.

Since the problem asks for the longest consecutive list, we observe the given example hint of $n = 1000$. If we naively start from the beginning of our prime number list, summing until n is exceeded, we do not get the expected result of 953. Clearly, this simple approach will not work. We need to find *the longest consecutive list* of primes that sum to a prime number, which is not necessarily the list beginning with 2, as in the simple example of 41, the sum from a list of 6 consecutive primes below 100.

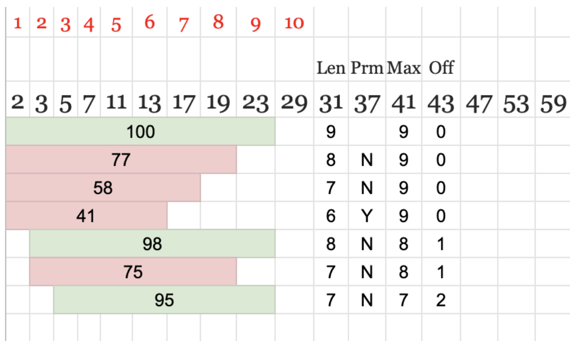

Figure 1: Visual of main algorithm

Figure 1 visualizes an example with $n = 100$, where we want to obtain the . The `solve` method begins by storing the prime number list. Initially, we obtain the largest possible "naive" list by iterating from 2 to some x in the list of prime numbers P where $\sum_{i=0}^{x} P[i] \leq 100$. In the above example, this is the list $2, \ldots, 23$ of length 9 which sums to 100. We store this as our *maximum possible sublist length*.

Then, we iterate through the different possible sublists of length < 9. Our outer loop increments a left offset that we can see easily in the above example. This helps us permutes through all possible starting points for sublists. It also checks to make sure the next sublist length will not fall below a threshold value defined by our current maximum sublist length (initialized to 0), since there is no point looking for smaller sublists than the maximum we will find. On each outer loop iteration, we enter an inner loop. We start with a sublist, denoted in green, check for primality, then successively decrease the sublist size using a "right offset" until we reach the maximum sublist length previously determined (initialized as 0). If the sublist sums to a prime number, we make sure it doesn't exceed a prior maximum and record it as the "best sum" and new maximum length.

```
997651
296.665ms
```

Figure 2: Output