**HACKERRANK: PROJECT EULER**

**✅ Day 1 | Problem 1 – Multiples of 3 and 5**

t = int(input())

for \_ in range(t):

n = int(input()) - 1

def sum\_divisible\_by(x):

p = n // x

return x \* p \* (p + 1) // 2

print(sum\_divisible\_by(3) + sum\_divisible\_by(5) - sum\_divisible\_by(15))

* **Goal:** Find the sum of all the multiples of 3 or 5 below N.
* **Code Strategy:**  
  Used arithmetic series formula for multiples of 3, 5, and subtracted the overlap (multiples of 15).
* **Time Complexity:** O(1)

**✅ Day 2 | Problem 2 – Even Fibonacci Numbers**

t = int(input())

for \_ in range(t):

n = int(input())

a, b, total = 1, 2, 0

while b <= n:

if b % 2 == 0:

total += b

a, b = b, a + b

print(total)

* **Goal:** Find the sum of even Fibonacci numbers not exceeding N.
* **Code Strategy:**  
  Generated Fibonacci numbers iteratively, summing only the even ones.
* **Time Complexity:** O(log N) (because every third Fibonacci number is even)

**✅ Day 3 | Problem 3 – Largest Prime Factor**

t = int(input())

for \_ in range(t):

n = int(input())

i = 2

while i \* i <= n:

if n % i == 0:

n //= i

else:

i += 1

print(n)

* **Goal:** Find the largest prime factor of N.
* **Code Strategy:**  
  Divided out 2, then checked for odd divisors up to √N.
* **Time Complexity:** O(√N)

**✅ Day 4 | Problem 4 – Largest Palindrome Product**

t = int(input())

for \_ in range(t):

n = int(input())

max\_pal = 0

for i in range(100, 1000):

for j in range(i, 1000):

p = i \* j

if p < n and str(p) == str(p)[::-1]:

max\_pal = max(max\_pal, p)

print(max\_pal)

* **Goal:** Find the largest palindrome product of two 3-digit numbers below N.
* **Code Strategy:**  
  Brute-force nested loop (optimized by breaking early), checked string reversal for palindromes.
* **Time Complexity:** O(N²) but optimized for early break

**✅ Day 5 | Problem 5 – Smallest Multiple**

import math

def lcm(a, b):

return a \* b // math.gcd(a, b)

t = int(input())

for \_ in range(t):

n = int(input())

result = 1

for i in range(2, n + 1):

result = lcm(result, i)

print(result)

* **Goal:** Find the smallest number divisible by all numbers from 1 to N.
* **Code Strategy:**  
  Iteratively calculated LCM using GCD from 1 to N.
* **Time Complexity:** O(N log N)

**✅ Day 6 | Problem 6 – Sum Square Difference**

t = int(input())

for \_ in range(t):

n = int(input())

sum\_n = n \* (n + 1) // 2

sum\_squares = (n \* (n + 1) \* (2 \* n + 1)) // 6

print(sum\_n \*\* 2 - sum\_squares)

* **Goal:** Difference between square of sum and sum of squares for first N natural numbers.
* **Time Complexity:** O(1)
* **Code Strategy:**  
  Used direct formulas:
* Sum = n(n+1)/2
* Sum of squares = n(n+1)(2n+1)/6

**✅ Day 7 | Problem 7 – 10001st Prime**

def get\_primes(limit):

primes = []

is\_prime = [True] \* (limit + 1)

is\_prime[0] = is\_prime[1] = False

for number in range(2, limit + 1):

if is\_prime[number]:

primes.append(number)

for multiple in range(number \* 2, limit + 1, number):

is\_prime[multiple] = False

return primes

t = int(input())

primes = get\_primes(9000000)

for \_ in range(t):

n = int(input())

print(primes[n - 1])

* **Goal:** Find the Nth prime number.
* **Code Strategy:**

Precomputed primes using Sieve of Eratosthenes up to a limit (9,000,000), returned primes[n-1].

* **Time Complexity:** O(N log log N) for sieve, O(1) per query

**✅ Day 8 | Problem 8 – Largest Product in a Series**

def max\_product\_of\_consecutive\_digits(n, k, num\_str):

max\_product = 0

for i in range(n - k + 1):

product = 1

for j in range(k):

product \*= int(num\_str[i + j])

max\_product = max(max\_product, product)

return max\_product

t = int(input())

for \_ in range(t):

n, k = map(int, input().split())

num = input().strip()

print(max\_product\_of\_consecutive\_digits(n, k, num))

* **Goal:** Find the greatest product of K consecutive digits in a number.
* **Code Strategy:**

Used a sliding window of size K, calculated product manually.

* **Time Complexity:** O(nk), could be optimized with smarter window handling

**✅ Day 9 | Problem 9 – Special Pythagorean Triplet**

t = int(input())

for \_ in range(t):

n = int(input())

found = -1

for a in range(1, n // 3):

for b in range(a + 1, (n - a) // 2):

c = n - a - b

if a \* a + b \* b == c \* c:

found = a \* b \* c

break

if found != -1: break

print(found)

* **Goal:** Find a Pythagorean triplet (a² + b² = c²) such that a + b + c = N, and return abc.
* **Code Strategy:**  
  Used triple nested loop with condition a + b + c = N and checked for Pythagorean property.
* **Time Complexity:** O(N²), but loop range was narrowed based on constraints