Laporan Final Project Perancangan dan Analisis Algoritma

SPOJ | DCEPC203 | OBSESSION

DOSEN PENGAMPU

Rully Soelaiman S.Kom., M.Kom

Nama dan NRP Anggota Kelompok:

Aryan Shafa Wardana | 5025211031

Arif Nugraha Santosa | 5025211048

Timothy Hosia Budianto | 5025211098

Permasalahan dan Pendekatan:

DCEPC203 - Obsession

#sieve

Ankur sir is really obsessed with dce coders. Every other day he announces to the admins the current number of members and boasts about the continuous increase in the member count. Within a week the admins got bored of his habit and started losing interest in this topic. Ankur sir is very clever. He knows that every body loves to solve puzzles. To maintain their interest he decides to ask a puzzle based on the number of current members. This way he can still bug the admins about the increasing member count without making them feel bored. The puzzle says for the given member count N, tell me how how many smaller numbers (k>=2) exist such that F(k)=2*k*k-1 is a prime.

Input

First line gives T, the number of test cases

Each of the next T lines give a number N.

Output

Print the output for every test case on a new line

Constraints

T<=1000

2<=N<=10000000

Example

Di soal ini digambarkan ada seseorang bernama ankur sir yang senang memberitahu orang - orang jumlah kenaikan member server miliknya. Agar admin tidak bosan dan orang - orang terus mengikuti jumlah kenaikan server miliknya, ankur sir membuatnya menjadi sebuah permainan puzzle memecahkan sebuah algoritma. Soal akan pertama meminta input integer yang dilambangkan T yaitu jumlah test case nya, dan variable integer N sebanyak T. Kemudian kita diminta untuk mencari dan mengoutputkan berapa banyak angka yang kurang dari N dan lebih dari sama dengan 2 yang bila dimasukkan kedalam persamaan $f(k) = 2k^2 - 1$ adalah prima.

Soal ini memiliki batasan / constraint sebagai berikut:

Time limit:	2.355s
Source limit:	3000B
Memory limit:	1536MB

Constraints

T<=1000

2<=N<=10000000

Pada soal ini diberikan sample testcase seperti berikut:

```
Input:
1
5

Output:
3
```

Diberikan nilai N adalah 5 dan menghasilkan nilai 3. 3 angka yang sesuai dengan prasyaratan soal jika dimasukan kedalam algoritma $\boldsymbol{f(k) = 2k^2 - 1}$ adalah

2 → (2 x 2 x 2) - 1 = 7 (prima)

3 → (2 x 3 x 3) - 1 = 17 (prima)

4 → (2 x 4 x 4) - 1 = 31 (prima)

Abstraksi:

Pada soal ini bisa diselesaikan dengan metode *brute force,* dengan menghitung $\boldsymbol{f(k) = 2k^2 - 1}$ untuk $k = 2,3, \ldots, n$ dan cek apabila $f(k)$ adalah prima. Pengecheckan menggunakan brute force dapat kita lambangkan sebagai pseudocode seperti berikut:

Pengecekan prima bisa dilakukan dengan algoritma

```
func isPrime(n: integer)
    if n <= 1 then
        return false
    for i = 2, 3, ..., √n do
        if n is divisible by i then
            return false
    return true
```

Jika soal ini diselesaikan dengan cara brute force maka akan menghasilkan penyelesaian dengan kompleksitas waktu O(T N log N). Dengan kompleksitas waktu tersebut maka akan menghasilkan verdict TLE (Time Limit Exceeded). Oleh karena itu kita perlu membuat solusi agar sesuai dengan batasan waktu yang ditetapkan.

Ditinjau dari metode brute force, kami melihat bahwa pada soal ini dapat mengimplementasikan beberapa metode untuk mengerjakan soal agar lebih efisien. Metode tersebut adalah bilangan prima, *sieve of eratosthenes*, uji primalitas Miller-Rabin, Primalitas bilangan berbentuk ($2n^2$ - 1), dan Algoritma Tonelli-Shanks.

Sebelum itu kita harus mengetahui terlebih dahulu apa itu bilangan prima, Bilangan prima adalah bilangan natural (1, 2, 3, dst.) yang lebih besar daripada 1 dan bukan merupakan hasil perkalian dari 2 bilangan natural yang lain. Dalam kata lain, bilangan prima merupakan bilangan natural yang hanya dapat dibagikan habis oleh 1 dan bilangan itu sendiri. Contohnya bilangan 7 adalah prima karena hanya dapat dibagikan habis oleh 1 dan 7, bilangan yang bukan prima disebut bilangan komposit.

Sieve of Erratosthenes:

Selanjutnya kita akan mengimplementasikan *sieve of erratosthenes* yaitu sebuah algoritma penyaring prima yang dapat mencari semua bilangan prima hingga suatu batas n. *Sieve of Eratosthenes* adalah salah satu cara paling efficient untuk mencari semua bilangan prima kurang dari n ketika n kurang dari kurang lebih 10 juta. Dalam pengimplementasiannya kita dapat merumuskan pseudocode sebagai berikut :

```
Pseudo code:
mark <-- boolean array [2..n]
func sieveOfEratosthenes(n: integer)
   for i = 2, 3, ..., √n do
      if not mark[i] then
         for j = i², i²+i, i²+2i, ..., n do
            mark[i] = true
```

Berikut ini adalah algoritma untuk menemukan semua bilangan prima kurang dari atau sama dengan bilangan bulat n yang diberikan dengan metode Eratosthenes:

Saat algoritma berakhir, semua angka dalam daftar yang tidak ditandai adalah bilangan prima.

Penjelasan dengan Contoh:

Mari kita ambil contoh ketika n = 50. Jadi kita perlu mencetak semua bilangan prima lebih kecil dari atau sama dengan 50.

Kami membuat daftar semua angka dari 2 hingga 50.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sumber dari https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Menurut algoritma, kami akan menandai semua angka yang habis dibagi 2 dan lebih besar dari atau sama dengan kuadratnya.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sumber dari https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Sekarang kita pindah ke angka 3 tanpa tanda berikutnya dan tandai semua angka yang merupakan kelipatan dari 3 dan lebih besar dari atau sama dengan kuadratnya.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sumber dari https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Kami pindah ke angka 5 tanpa tanda berikutnya dan menandai semua kelipatan 5 dan lebih besar dari atau sama dengan kuadratnya.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sumber dari https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Kami melanjutkan proses ini dan tabel akhir kami akan terlihat seperti di bawah ini:

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sumber dari https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Jadi bilangan prima adalah yang tidak bertanda: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47. Sedangkan implementasi dan penerapan algoritma sieve of eratosthenes di soal ini adalah

Dengan menerapkan algoritma *Sieve of Eratosthenes*, kita bisa menghasilkan bilangan prima hingga 10^9. Sedangkan, pada soal terdapat batasan atas N adalah 10^7 dan diminta untuk mencari jika suatu bilangan $2k^2 - 1$ adalah prima. Dikarenakan k dalam bentuk kuadrat, maka akan diminta untuk mencari bilangan prima hingga $k^2 = 10^7 = 10^{14}$ sehingga memori yang digunakan untuk menyimpan bilangan-bilangan prima tidak cukup.

Miller-Rabin Primality Test:

Selanjutnya kita akan menggunakan uji primalitas Miller-Rabin. Uji primalitas Miller-Rabin merupakan algoritma pengujian primalitas suatu bilangan yang menggunakan probabilitas. Algoritma ini menentukan jika suatu bilangan mungkin prima. Algoritma ini memiliki kompleksitas waktu O (k $\log^3$ n) dimana n merupakan bilangan yang ditest peimalitasnya. Probabilitas algoritma ini disebabkan karena menggunakan suatu bilangan random.

Bilangan bilangan prima yang dicek harus hingga k^2 sehingga kita bisa menggunakan pengujian prima yang mampu menangani bilangan prima yang lebih besar daripada 10^9. Algoritma yang dipakai adalah uji primalitas miller-rabin.

Tetapi tetap ada masalah dimana Miller-Rabin jika dilakukan untuk setiap k > 10^9 akan menyebabkan TLE. Oleh karena itu, diperlukan tambahan algoritma yang bisa komputasi terlebih dahulu bilangan-bilangan prima besar.

Algortima Miller-rabin memiliki kemiripan dengan the Fermat primality test and the Solovay–Strassen primality test.

Konsep Algoritma:
Seperti yang dijelaskan sebelumnya, algoritma ini menggunakan konsep probabilitas untuk menentukan apakah suatu bilangan itu mungkin prima atau komposit. Terdapat beberapa properti dalam algoritma ini. Properti tersebut akan menyimpan untuk nilai prima, dan menyimpan sebuah nilai untuk dilakukan testing. Langkah – Langkah yang diperlukan untuk menjalankan algoritma ini adalah:

1. Menentukan $n - 1 = 2^k \times m$.

2. Menentukan nilai *a* dimana $1 < a < n - 1$. (Nilai *a* merupakan nilai random dalam range tersebut).
3. Melakukan perhitungan $b_0 = a^m \pmod n, \dots, b_i = b^2_{i-1} \pmod n$

   Pada langkah ketiga nilai dari *a* didapatkan dari langkah kedua, dan nilai *m* didapatkan dari langkah pertama, dan nilai *n* merupakan bilangan yang akan dicek primalitasnya.

   Apabila nilai dari $b = +1$ maka bilangan itu adalah bilangan komposit. Namun apabila nilai $b = -1$ maka bilangan itu kemungkinan prima.

   Apabila nilai dari b_0 tidak memenuhi kedua persyaratan di atas, maka akan dicari b_i dengan persamaan di atas hingga nilai dari *b* adalah +1 atau -1.

Contoh:

Apakah 561 merupakan bilangan prima?

Maka $n = 561$.

Langkah pertama:

$n - 1 = 2^k \times m$

$560 = 2^k \times m$

Kita dapat mencari nilai *k* dan *m* dengan cara berikut:

$$\frac{560}{2^1} = 280 \rightarrow \frac{560}{2^2} = 140 \rightarrow \frac{560}{2^3} = 70 \rightarrow \frac{560}{2^4} = 35 \rightarrow \frac{560}{2^5} = 17{,}5$$

Namun karena 17,5 bukan bilangan bulat (integer), maka kita akan menggunakan persamaan sebelumnya. Sehingga:

$560 = 2^4 \times 35$

Didapatkan $k = 4$ dan $m = 35$.

Langkah kedua:

a = random value dari $1 < a < n - 1$.

Saya memilih $a = 2$.

Langkah ketiga:

$b_0 = a^m \pmod n$

$b_0 = 2^{35} \pmod{561} = 263$

Apakah $b_0 = \pm 1 \pmod{561}$? Tidak

Maka kalkulasikan b_1

$b_1 = b_0^2 \pmod n$

$b_1 = 263^2 \pmod{561} = 166$

Apakah $b_1 = \pm 1 \pmod{561}$? Tidak

Maka kalkulasikan b_2

$b_2 = b_1^2 \pmod n$

$b_2 = 166^2 \pmod{561} = 67$

Apakah $b_2 = \pm 1 \pmod{561}$? Tidak

Maka kalkulasikan b_3

$b_3 = b_2^2 \pmod n$

$b_3 = 67^2 \pmod{561} = 1$

Apakah $b_1 = \pm 1 \pmod{561}$? Ya

561 Merupakan bilangan komposit

Contoh 2:

Apakah 53 merupakan bilangan prima?

Maka $n = 53$.

Langkah pertama:

$n - 1 = 2^k \times m$

$52 = 2^k \times m$

Kita dapat mencari nilai *k* dan *m* dengan cara berikut:

$$\frac{52}{2^1} = 26 \rightarrow \frac{52}{2^2} = 13 \rightarrow \frac{52}{2^3} = 6.5$$

Namun karena 6,5 bukan bilangan bulat (integer), maka kita akan menggunakan persamaan sebelumnya. Sehingga:

$52 = 2^2 \times 13$

Didapatkan $k = 2$ dan $m = 13$.

Langkah kedua:

a = *random value dari* $1 < a < n - 1$.

Saya memilih $a = 2$.

Langkah ketiga:

$b_0 = a^m \pmod n$

$b_0 = 2^{13} \pmod{53} = 30$

Apakah $b_0 = \pm 1 \pmod{53}$? Tidak

Maka kalkulasikan b_1

$b_1 = b_0^2 \pmod n$

$b_1 = 30^2 \pmod{53} = 52$

Apakah $b_1 = \pm 1 \pmod{53}$? Ya

***Note: -1 mod 53 = 52*, sehingga 52 dapat ditulis *-1 mod 53*.**

Karena $b_1 = -1$, maka 53 adalah bilangan prima (mungkin).

Mengapa nilai dari *a* adalah random?

Terdapat properti – properti yang sudah dijelaskan sebelumnya seperti $n - 1 = 2^k \times m$ Dimana *k* adalah bilangan bulat positif dan *m* adalah bilangan bulat ganjil positif. Dan terdapat nilai *a* yang biasa juga disebut *base* dan nilai *a* adalah koprima dengan *n*. Dengan itu *n* dapat dibilang memiliki kemungkinan prima yang lebih besar dengan *base a* apabila salah satu dari *congruence relation* dibawah benar:

- $a^m \equiv 1 \ (mod\ n)$
- $a^{2^r m} \equiv -1 \ (mod\ n); untuk\ sebagian\ 0 \leq r \leq s$

Ide dari algoritma ini adalah apabila *n* adalah sebuah bilangan prima ganjil, maka *n* akan valid karena:

- Melalui teorema fermat, $a^{n-1} \equiv 1 \ (mod\ n)$.
- Akar persamaan dari $1 \ (mod\ n)$ adalah 1 atau -1. (Quadratic residue).

Perhatikan bahwa $a^m \equiv 1\ (mod\ n)$ mengandung nilai yang mirip dengan $a \equiv 1\ (mod\ n)$, karena *congruence relation* kompatibel dengan eksponensial. Dan $a^m = a^{2^0 m} \equiv -1\ (mod\ n)$ mengandung nilai yang mirip dengan $a \equiv -1\ (mod\ n)$ dengan *d* merupakan nilai ganjil. Dengan alasan tersebut, nilai *a* dipilih dengan interval $1 < a < n - 1$.

Pseudocode Miller-Rabin Primality Test:

```
Input #1: n > 2, an odd integer to be tested for primality
Input #2: k, the number of rounds of testing to perform
Output: "composite" if n is found to be composite, "probably prime" otherwise

let s > 0 and d odd > 0 such that n − 1 = 2^s d  # by factoring out powers of 2 from n − 1
repeat k times:
    a ← random(2, n − 2)  # n is always a probable prime to base 1 and n − 1
    x ← a^d mod n
    repeat s times:
        y ← x^2 mod n
        if y = 1 and x ≠ 1 and x ≠ n − 1 then  # nontrivial square root of 1 modulo n
            return "composite"
        x ← y
    if y ≠ 1 then
        return "composite"
return "probably prime"
```

Source: Wikipedia: Miller-rabin primality test

Primalitas bilangan berbentuk $2n^2 - 1$:

Kemudian kita akan menggunakan konsep primalitas bilangan berbentuk $2n^2$- 1. Algoritma ini akan digunakan saat pada case berikut:

Jika terdapat suatu bilangan $n = ak + b$ atau $n = ak - b$ dengan $a \equiv 1, 7\ (\mathbf{mod}\ 8)$, maka bilangan dengan bentuk $2n^2 - 1$ merupakan bilangan komposit untuk semua bilangan k.

Dengan bukti sebagai berikut ;

Bukti:
Misalkan $n = ak + b$ lalu anggap bahwa $p|a$ dan $2b^2 - 1 \equiv 0 \pmod{p}$. Jika disubstitusikan ke $2n^2 - 1$ maka $2n^2 - 1 = 2(ak + b)^2 - 1$ sehingga $a(2ak^2 + 4bk) + 2b^2 - 1 \equiv 0 \pmod{p}$. Dengan demikan, $p|2n^2 - 1$ sehingga $2n^2 - 1$ adalah bilangan komposit untuk semua k. Hal ini juga berlaku untuk $n = ak - b$.
Untuk $2b^2 - 1 \equiv 0 \pmod{p}$ bisa ditulis $b^2 \equiv \frac{1}{2} \pmod{p}$. Persamaan ini bisa diselesaikan menggunakan simbol Legendre. **Simbol Legendre** adalah fungsi a dan p yang didefinisikan sebagai

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod{p}, \\ -1 & \text{if } a \text{ is a non-quadratic residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod{p}. \end{cases}$$

Suatu bilangan q disebut *quadratic residue* atau residu kuadrat modulo n jika q kongruen dengan kuadrat sempurna modulo n, yaitu jika terdapat suatu bilangan bulat x yang memenuhi persamaan kongruen

$$x^2 \equiv q \pmod{n}$$

Kemudian, dengan menggunakan simbol Legendre, b memiliki penyelesaian jika

$$\left(\frac{1/2}{p}\right) = 1$$

Jika kedua ruas dikalikan dengan $\left(\frac{2}{p}\right)$ maka diperoleh

$$\left(\frac{1}{p}\right) = \left(\frac{2}{p}\right) \rightarrow \left(\frac{2}{p}\right) = 1$$

Pada $\left(\frac{2}{p}\right)$, 2 merupakan residu kuadrat untuk $p = 7, 17, 23, 31, 41, \ldots,$ dst. sehingga diperoleh bahwa $p \equiv 1,7 \pmod{8}$

Tonelli-Shanks Algorithm:

Yang terakhir kita akan mengimplementasikan Algotima Tonelli-Shanks sebagai berikut:

Pada dasar teori subbab "Primalitas Bilangan Primalitas $2n - 1$", terdapat bahwa $2n^2 - 1$ adalah bilangan komposit jika $n = ak \pm b$ dan $p|a$. p merupakan bilangan yang memenuhi $p \equiv 1, 7 \pmod{8}$. Karena $p|a$, maka bisa dimisalkan $a \equiv 1, 7 \pmod{8}$.

Untuk mencari b, kita menggunakan persamaan $2b^2 \equiv 1 \pmod{p}$. Untuk memperoleh b tersebut, kita bisa mengimplementasikan algoritma Tonelli-Shanks. Tonelli-Shanks digunakan untuk mencari r dalam $r^2 \equiv n \pmod{p}$. Untuk mengubah bentuk $2b^2 \equiv 1 \pmod{p}$ menjadi bentuk $r^2 \equiv n \pmod{p}$, n harus ditulis sebagai inverse modulo dari $2 \pmod{p}$.

Algoritma Tonelli-Shanks digunakan pada *modular arithmetic* untuk menyelesaikan permasalahan $r^2 \equiv n \, (mod \, p)$ dimana p adalah bilangan prima. Maka dari itu, algoritma ini digunakan untuk mencari hasil dari $\sqrt{n \, modulo \, p}$

Algoritma ini tidak dapat digunakan apabila nilai dari p adalah komposit.

Untuk mengaplikasikan algoritma ini, diperlukan symbol Legendre.

Pseudocode Tonelli-Shanks Algorithm:

All ≡ are taken to mean (mod p) unless stated otherwise.

- Input: **p** an odd prime, and an integer **n** .
- Step 0: Check that **n** is indeed a square: (n | p) must be ≡ 1 .
- Step 1: By factoring out powers of 2 from p - 1, find **q** and **s** such that $p - 1 = q2^s$ with **q** odd .
  - If p ≡ 3 (mod 4) (i.e. s = 1), output the two solutions $r \equiv \pm n^{(p+1)/4}$.
- Step 2: Select a non-square **z** such that (z | p) ≡ -1 and set $c \equiv z^q$.
- Step 3: Set $r \equiv n^{(q+1)/2}$, $t \equiv n^q$, m = s .
- Step 4: Loop the following:
  - If t ≡ 1, output **r** and **p - r** .
  - Otherwise find, by repeated squaring, the lowest **i**, 0 < i < m , such that $t^{2^i} \equiv 1$.
  - Let $b \equiv c^{2^{(m - i - 1)}}$, and set $r \equiv rb$, $t \equiv tb^2$, $c \equiv b^2$ and m = i .

Rosettacode: Tonelli-shanks Algorithm

the algorithm requires O(log A + r * r) multiplications modulo A, where r is the power of 2 dividing A − 1.

Rosettacode: Tonelli-shanks Algorithm

Berdasarkan dasar teori dan metode yang telah disebutkan diatas kita akan mulai mengimplementasikannya dalam source code sebagai berikut

Implemetasi Kode di C++:

```cpp
#include <cstdio>
#include <cstdlib>
#define MAX 10000000
typedef long long ll;

// a * p mod m
ll modmul(ll a, ll b, ll M) {
    ll ret = a * b - M * ll(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}

// a ^ b mod M
ll modex(ll a, ll p, ll m) {
    ll res = 1;

    while(p)
    {
        if (p & 1) res = modmul(res, a, m);
        a = modmul(a, a, m);
        p >>= 1;
    }
    return res;
}

bool isQuadraticResidue(ll a, ll n) {
    return modex(a, (n - 1)/2, n) == 1;
}

ll tonelliShanks(ll n, ll p) {
    if (!isQuadraticResidue(n, p)) {
        return -1;
    }
    ll q = p - 1;
    ll s = 0;
    while (~q & 1) {
        q >>= 1;
        s += 1;
    }
    if (s == 1) {
```

```
        ll x = modex(n, (p + 1)/4, p);
        return x;
    }
    ll z = 0;
    for (ll k = 1; k < p; ++k) {
        if (!isQuadraticResidue(k, p)) {
            z = k;
            break;
        }
    }

    ll c = modex(z, q, p);
    ll r = modex(n, (q + 1)/2, p);
    ll t = modex(n, q, p);
    ll m = s;

    while (t != 1) {
        int i = 1;
        ll x = modmul(t, t, p);
        while (x != 1) {
            x = modmul(x, x, p);
            i += 1;
        }
        ll b = modex(c, (1ll << (m - i - 1)), p);
        r = modmul(r, b, p);
        c = modmul(b, b, p);
        t = modmul(t, c, p);
        m = i;
    }
    return r;
}

#define MX 1000000
bool mark[MX];

bool millerRabin(ll p) {
    if (p < MX) {
        return !mark[p];
    }

    ll a = rand() % (p - 4) + 2;

    ll s = p - 1;
    while (~s & 1) s >>= 1;
    ll mod = modex(a, s, p);
```

```
    if (mod == 1 || mod == p - 1) return 1;

    s <<= 1;
    while(s != p - 1)
    {
        mod = modmul(mod, mod, p);
        if (mod == p - 1) return 1;
        s <<= 1;
    }
    return 0;
}

int cum[MAX];
int vis[MAX];

int main() {
    for (int i = 2; i * i < MX; i++)
        if (!mark[i])
            for(int j = i * i; j < MX; j += i)
                mark[j] = 1;

    for (int i = 2; i < MX; i++) {
        if (!mark[i]) {
            if (i % 8 == 1 || i % 8 == 7){
                int a = tonelliShanks((i + 1)/2, i);
                for(int j = i + i; j + a < MAX; j += i){
                    vis[j + a] = vis[j - a] = i;
                }
            }
        }
    }

    for (int i = 2; i < MAX; i++) {
        if (vis[i]) {
            cum[i + 1] = cum[i];
            continue;
        }
        if (millerRabin(1ll * 2 * i * i - 1)) {
            cum[i + 1]++;
        }
        cum[i + 1] += cum[i];
    }

    int t, n;
```

```
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        printf("%d\n", cum[n]);
    }

    return 0;

}
```

Bukti Accepted dan Hasil:

Dengan code diatas kita akan melakukan pengujian luar menggunakan online judge untuk menguji kebenaran dan kinerja program. Dilakukan dengan mengirimkan kode program ke situs penilaian daring SPOJ sebanyak 10 kali. Setelah kami debug kami mengetahui bahwa testcase yang akan digunakan adalah:

Input:	**Dengan Output:**
7	6
10	45
100	303
1000	2202
10000	17185
100000	141444
1000000	1200975
10000000	

Berikut adalah bukti submission yang kami lakukan sebanyak 10 kali di website SPOJ:

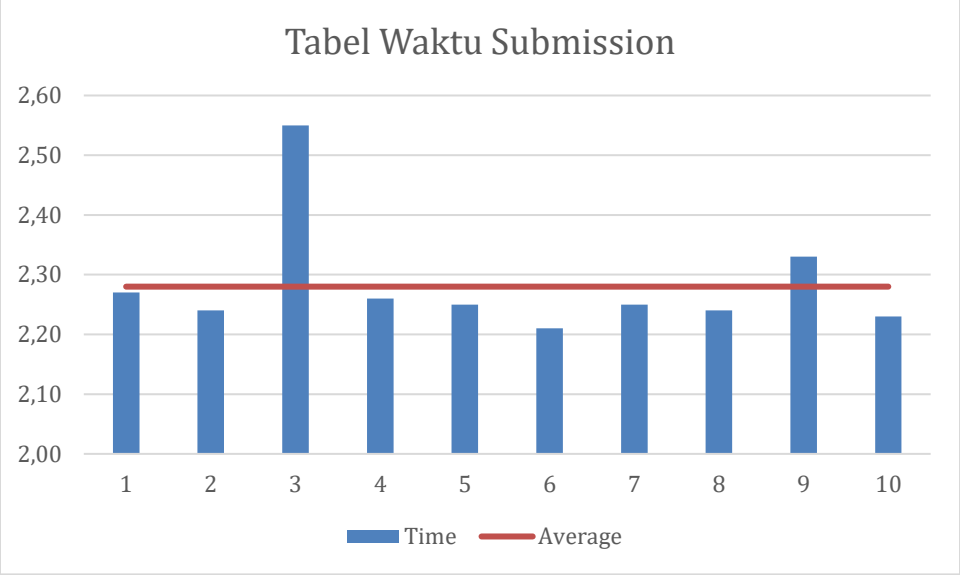
30527516		2022-12-05 17:01:49	Obsession	**accepted** edit ideone it	2.23	80M	CPP14-CLANG
30527514		2022-12-05 17:01:28	Obsession	**accepted** edit ideone it	2.33	80M	CPP14-CLANG
30527510		2022-12-05 17:01:10	Obsession	**accepted** edit ideone it	2.24	80M	CPP14-CLANG
30527507		2022-12-05 17:00:51	Obsession	**accepted** edit ideone it	2.25	80M	CPP14-CLANG
30527498		2022-12-05 16:59:15	Obsession	**accepted** edit ideone it	2.21	80M	CPP14-CLANG
30527494		2022-12-05 16:58:48	Obsession	**accepted** edit ideone it	2.25	80M	CPP14-CLANG
30527492		2022-12-05 16:58:14	Obsession	**accepted** edit ideone it	2.26	80M	CPP14-CLANG
30527490		2022-12-05 16:57:53	Obsession	**accepted** edit ideone it	2.55	80M	CPP14-CLANG
30527488		2022-12-05 16:57:29	Obsession	**accepted** edit ideone it	2.24	80M	CPP14-CLANG
30527483		2022-12-05 16:56:40	Obsession	**accepted** edit ideone it	2.27	80M	CPP14-CLANG

Hasil ranking di SPOJ kami mendaatkan ranking ke 13 sebagai berikut:

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2017-09-16 17:42:23	Min_25	**accepted**	0.16	70M	CPP14
2	2018-02-12 00:25:13	Lovro Puzar	**accepted**	0.27	28M	CPP14
3	2019-12-13 04:36:10	fakesson	**accepted**	0.77	50M	CPP14
4	2016-09-28 17:39:47	Aayush Dwivedi	**accepted**	1.02	65M	CPP14
5	2013-03-12 11:17:23	[Rampage] Blue.Mary	**accepted**	1.10	2.6M	C++ 4.3.2
6	2017-02-10 14:19:39	Francky	**accepted**	1.19	10M	C
7	2017-07-10 18:41:47	[Lakshman]	**accepted**	1.57	92M	CPP
8	2014-07-13 21:16:21	wisfaq	**accepted**	1.58	280k	PAS-FPC
9	2017-07-15 06:34:44	Chandan Pandey	**accepted**	1.61	92M	CPP14
10	2015-06-30 21:13:18	Rivu Das	**accepted**	1.63	57M	C++ 4.3.2
11	2014-12-17 04:22:34	Min_25	**accepted**	1.82	88M	C++ 4.3.2
12	2012-03-11 12:07:33	Hellfire Temple	**accepted**	2.07	117M	C++ 4.3.2
13	2022-12-05 16:59:15	TimothyHosia_5025211098	**accepted**	2.21	80M	CPP14-CLANG

Dari data diatas dapat dihasilkan rata rata waktu dan memory sebagai berikut

uji coba ke	Time	Memory
1	2,27	80M
2	2,24	80M
3	2,55	80M
4	2,26	80M
5	2,25	80M
6	2,21	80M
7	2,25	80M
8	2,24	80M
9	2,33	80M
10	2,23	80M
Rata-Rata	2,28	80



Dapat disimpulkan bahwa metode dan teori yang kita bahas tadi dapat membuat penyelesaian soal lebih efisien sehingga sesuai dengan time dan memory constraint yang ada pada soal. Dapat disimpulkan bahwa metode yang kami rumuskan menghasilkan rata-rata waktu 2,28s dan memory 80MB.

Daftar Pustaka

Youtube - Finding Mod-p Square Roots with the Tonelli-Shanks Algorithm: https://www.youtube.com/watch?v=d7ZFCf95MAQ

Wikipedia - Tonelli–Shanks algorithm: https://en.wikipedia.org/wiki/Tonelli%E2%80%93Shanks_algorithm

Rosettacode - Tonelli-Shanks algorithm: https://rosettacode.org/wiki/Tonelli-Shanks_algorithm

GitHub - Quadratic Congruence Solver: https://github.com/panoti/CH_QuadraticCongruenceSolver/blob/master/main.py

Wikipedia - Miller-Rabin primality test: https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test

Youtube - Miller-Rabin Primality Test: https://www.youtube.com/watch?v=qdylJqXCDGs

Youtube - Testing for Primality (Miller-Rabin Test): https://www.youtube.com/watch?v=8i0UnX7Snkc

Wikipedia - Sieve of Eratosthenes: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
Pembuktian primalitas bilangan berbentuk 2n^2-1: https://math.stackexchange.com/questions/403519/primality-of-the-numbers-in-the-form-of-2n2-1

Wikipedia - Prime Numbers: https://en.wikipedia.org/wiki/Prime_number