Number Theory 1

1) Factorization
2) Sieve of Eratosthenes
3) Prime Factorization
    -> smalles Prime Factor
    -> number of factors
    -> sum of factors

4) Modular Arithmetic
5) Exponentiation

Factores

```
for(int i = 1 ; i < n ; i++){
    if(n % i == 0){
        cnt++;
    }
}
```

1 <= n <= 1e12 --> TLE

How should we solve this problem

clame – 1

–>Factores occure as pairs

Proof:

p * q = n

n % p == 0 –-> p * n/p = n

n/p is an other factor which is q

Example:-

n = 100

1	2	4	5	10
100	50	25	20	10

factors in Pairs

clame – 2

–>min(p,q) <= sqrt(n)

Proof: Contradiction

$$p \geq \sqrt{n} \quad \& \quad q \geq \sqrt{n}$$

$$(\sqrt{n}+1) * (\sqrt{n}+1) = n + 2\sqrt{n} + 1$$

which is Greater Then n //

->iterate over the smaller number

```
vector<long long> FindFactors(long long n){
    vector<long long>factors;
    for(long long i = 1 ; i * i <= n ; i++){
        if(n % i == 0){
            factors.push_back(i);
            if(n / i != i){
                factors.push_back(n/i);
            }
        }
    }
    return factors;
}
```

Time Complexity : $O(\sqrt{n})$

Factorization : Find all factors of a number N

- Naive way

Check every number for factor from 1 to N

Time Complexity: O(N)

- Factors occur in pairs

N = P * Q

Time Complexity: Without loss of generality if P < Q => P <= sqrt(N)

- Efficient way

Check for every P from 1 to sqrt(N). Q = N / P

Time Complexity: O(sqrtN)

Problem 2:

Print all Prime factores of n

O(n^2) –> solution

$n = 18 \rightarrow (2, 3)$

$1 \le n \le 10^5 \rightarrow$ TLE

O(n * sqrt(n)) –> solution

$1 \le n \le 10^{12} \rightarrow$ TLE

⇒ To optimize let us Consider few things

Prime Representation

n = 24 –> 2*2*2*3 –> 2^3 * 3^1

n != m then PR(n) != PR(m)

Proof:

Product

Every number can be Represented in prime factoriz

clame:
the smallest factor of a number > 1 is always a prime

Proof:
consider smallest factor is composite x => p1^k1 * p2^k2 ...

if x is a factor of n
if y is a factor of x then y is a factor of n

$n = C_1 \times P_1 \times P_2 \Leftarrow$ This claim does not hold

Composite

↳ broken down into more primes

$C_1 = P_1^{1k_1} * P_2^{1k_2}$

Then

$n = P_1^{1k_1} * P_2^{1k_2} \times P_1 \times P_2$

↑ smallest factor > 1 is always a Prime //

* First factor is a prime number
* There can be only 1 prime factor > sqrt(n)

$$34 = 17 \times 2$$

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    long long n;
    cin >> n;
    vector<long long>PrimeFactors;
    for(int i = 2 ; i * i <= n; i++){
        if(n % i == 0){
            PrimeFactors.push_back(i);
            while(n % i == 0)n/=i;
        }
    }
    if(n!=1)PrimeFactors.push_back(n);
    for(auto i : PrimeFactors){
        cout << i << " ";
    }
}
```

Time Complexity :- $O(\sqrt{n}) * O(\log(n))$

$$\Rightarrow O(\sqrt{n} * \log(n))$$

$$1 \leq n \leq 10^{12} \rightarrow O(\sqrt{10^{12}} * \log(10^{12}))$$

$$\Rightarrow O(10^{6} * 39)$$

Queries

1e6 Queries

X <= 1e6 find prime or not

Sieve of Eratosthenes

An efficient Algorithm to find all Prime numbers up to N in $O(N \log \log N)$ time. It marks multiples of each Prime starting f

1	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

```
int n = 1e7;
vector<bool>isPrime(n+1,false);
void sieve(){
    for(int i = 2 ; i <= n ; i++)isPrime[i] = true;

    for(int i = 2 ; i * i <= n ; i++){
        if(isPrime[i]){
            for(int j = i * i ; j <= n ; j+=i){
                isPrime[j] = false;
            }
        }
    }

}
```

Smallest Prime Factor [SPF]

SPF of a number N is the Smallest Prime that Divides it. Using a modified Sieve, we can Precompute SPF for every number upto N. which helps in fast Prime factorization.

```cpp
#include<bits/stdc++.h>
using namespace std;
int n = 1e7;
vector<bool>isPrime(n+1,false);
vector<long long>SPF(n+1);
void sieve(){
    for(int i = 2 ; i <= n ; i++)isPrime[i] = true , SPF[i] = i;
    SPF[0] = 0, SPF[1] = 1;
    for(int i = 2 ; i * i <= n ; i++){
        if(isPrime[i]){
            for(int j = i * i ; j <= n ; j+=i){
                isPrime[j] = false;
                if(SPF[j] == j)SPF[j] = i;
            }
        }
    }

}
```

Efficient Prime factorization

1. number of Divisors
2. Sum of Divisors
3. Product of Divisors

1. no of Divisors

$$n = p_1^{k_1} * p_2^{k_2} * p_3^{k_3}$$

$$18 = 2^1 * 3^2$$

2^0 to 2^1 $\quad$ 3^0 to 3^2

$1 = 2^0 * 3^0$
$2 = 2^1 * 3^0$
$3 = 2^0 * 3^1$
$6 = 2^1 * 3^1$
$9 = 2^0 * 3^2$
$18 = 2^1 * 3^2$

in total

$(1+1) * (2+1) = 2 \times 3 = 6$

Hence it is

$(k_1+1) * (k_2+1) \ldots$

will give us no of Divisors.

② Sum of Divisors

$$18 \Rightarrow 1 + 2 + 3 + 6 + 9 + 18$$

$$= 2^0 3^0 + 2^1 3^0 + 2^0 3^1 + 2^1 3^1 + 2^0 3^2 + 2^1 3^2$$

$$= 2^0 \left(3^0 + 3^1 + 3^2\right) + 2^1 \left(3^0 + 3^1 + 3^2\right)$$

$$= \underbrace{\left(2^0 + 2^1\right)}_{GP} * \underbrace{\left(3^0 + 3^1 + 3^2\right)}_{GP}$$

$$\sigma(n) = \left[\frac{p_1^{e_1+1} - 1}{p_1 - 1}\right] \left[\frac{p_2^{e_2+1} - 1}{p_2 - 1}\right] \cdots$$

③ Product of Divisors

$$P(18) = 1 \times 2 \times 3 \times 6 \times 9 \times 18$$

$$P^2(18) = (1 \times 2 \times 3 \times 6 \times 9 \times 18) \times (18 \times 9 \times 6 \times 3 \times 2 \times 1)$$

$$= \underline{1 \times 18} \times \underline{2 \times 9} \times \underline{3 \times 6} \times \underline{6 \times 3} \times \underline{9 \times 2} \times \underline{18 \times 1}$$

$$= 18 \times 18 \times 18 \times 18 \times 18 \times 18$$

$$= 18^{\text{no of factors}}$$

$P^2(18) = 18^6 \rightarrow$ even power

what if power is odd ?

$$P(18) = 18^{6/2}$$

if the power is odd Then one of the factor is Square root

$\therefore$ $P^{f/2} \times \mathrm{Sqrt}(n)$