```
void sieve() {
   for (int i = 3; i * i < N; i += 2) {
      if (marked[i] == false) // i is a prime {
         for (int j = i * i; j < N; j += i + i) {
            marked[j] = true;
         }
      }
   }
}
bool isPrime(int n) {
   if (n < 2) return false;
   if (n == 2) return true;
   if (n % 2 == 0) return false;
   return marked[n] == false;
}
```

Prime Factorization (Integer factorization):

=> O(sqrt(n))

Ex: 36 => 2 2 3 3

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

Prime Factorization using Sieve algorithm:

=> O(log(n))

Ex: 50 => 2 5 5

```
vector<int> spf(N); // SPF : smallest prime factor
void sieve()            // => O( nloglogn)
{
   for (int i = 1; i < N; i++)  spf[i] = i;
   for (int i = 2; i * i < N; i++) {
      if (spf[i] == i) {
         for (int j = i * i; j < N; j += i)
              if (spf[j] == j) spf[j] = i;
      }
   }
}
int main() {
   sieve();
   int n;
   cin >> n;
   while (n != 1) {
      cout << spf[n] << " ";
      n /= spf[n];
   }
}
```

Binary Exponentiation using Iterative method:

=> O(log(b)).

Ex: 3^{13} => $3^{(8+4+0+1)}$ => $3^8 * 3^4 * 3^0 * 3^1$ => 1594323;

→(a^b)

```
const int Mod = 1e9 + 7;
long long BinExpIter(ll a, ll b) {
   ll ans = 1;
   while (b) {
      if (b & 1) ans = (ans * a) % Mod;
      a = (a * a) % Mod;
      b >>= 1;
   }
   return ans;
}
```

Binary Exponentiation for $N^{1/x}$:

=> O(x*log(N*10^d))

$3^{1/5}$**=** 1.2457312346;

```
double eps = 1e-6;   // eps=1e-d; =>with d decimal accuracy
double BinExpPow (double n, int x) {
   double l = 0, r = n, m = (l + r) / 2;
   while (r - l > eps) {
      if (pow(m, x) > n) r = m;
      else l = m;
      m = (l + r) / 2;
   }
   return m;
}
```

Euler Totient Function:

```
// Find the co-prime between(1 to i);
// Time Complexity: O(NloglogN)
const int N = 1e6 + 7;
int coprimeCnt[N];
ll coprimeSum[N];

void generatePhi() {
   for (int i = 0; i < N; ++i) coprimeCnt[i] = i;
   for (int i = 2; i < N; i++) {
      if (coprimeCnt[i] == i) {
         for (int j = i; j < N; j += i)
            coprimeCnt[j] -= coprimeCnt[j] / i;
      }
   }
   // Sum of all coprime values (Ex: 10 => 1 + 3 + 7 + 9 = 20)
   coprimeSum[1] = 1;
   for (ll i = 2; i < N; ++i)
         coprimeSum[i] = (i * coprimeCnt[i]) >> 1;
}
```

Find the co-prime between(1 to i) => O(sqrt(n))