Sieve of Eratosthenes

It is easy to find if some number (say N) is prime or not — you simply need to check if at least one number from numbers lower or equal sqrt(n) is divisor of N. This can be achieved by simple code:

```
boolean isPrime( int n ) {
    if ( n == 1 ) return false; // by definition, 1 is no
t prime number
    if ( n == 2 ) return true; // the only one even prime
    for ( int i = 2; i * i <= n; ++i )
        if ( n%i == 0 ) return false;
    return true;
}
```

So it takes sqrt(n) steps to check this. Of course you do not need to check all even numbers, so it can be “optimized” a bit:

```
boolean isPrime( int n ) {
    if ( n == 1 ) return false; // by definition, 1 is no
t prime number
    if ( n == 2 ) return true; // the only one even prime
    if ( n%2 == 0 ) return false; // check if is even
```

```
    for ( int i = 3; i * i <= n; i += 2 ) // for each odd
 number
        if ( n%i == 0 ) return false;
    return true;
}
```

So let say that it takes ***0.5*sqrt(n) steps***. That means it takes 50,000 steps to check that 10,000,000,000 is a prime.

Problem?

If we have to check numbers upto N, we have to check each number individually. So time complexity will be **O(Nsqrt(N))**.

Can we do better?

Ofcourse! we can use a sieve of numbers upto N. For all prime numbers <= sqrt(N), we can make their multiple non-prime i.e. if **p** is prime, 2*p, 3*p, ..., floor(n/p)*p will be **non-prime**.

Animation

https://upload.wikimedia.org/wikipedia/commons/b/b9/Sieve_of_Eratosthe

Sieve Code

```
void primes(int *p){
    for(int i = 2;i<=1000000;i++)p[i] = 1;
    for(int i = 2;i<=1000000;i++){
        if(p[i]){
            for(int j = 2*i;j<=1000000;j+=i){
```

```
                    p[j] = 0;
                }
            }
        }
        p[1] = 0;
        p[0] = 0;
        return;
}
```

Can we still do better?

Yeah sure! Here we don't need to check for even numbers. Instead of starting the non-prime loop from ***2p*** we can start from ***p^2***.

Optimized Sieve

```
void primes(bool *p){
    for(int i = 3;i<=1000000;i += 2){
        if(p[i]){
            for(int j = i*i;j <= 1000000; j += i){
                    p[j] = 0;
            }
        }
    }
    p[1] = 0;
    p[0] = 0;
    return;
}
```

T = O(NloglogN)

Hence, we have signifiacntly reduced our complexity from N*sqrt(N) to approx linear time.

Segmented Sieve:

```
void sieve(){
    for(int i = 0;i<=1000000;i++)
        p[i] = 1;
    for(int i = 2;i<=1000000;i++){
        if(p[i]){
            for(int j = 2*i;j<=1000000;j+=i)
                p[j] = 0;
        }
    }
    // for(int i=2;i<=20;i++)cout<<i<<"  "<<p[i]<<endl;
}

int segmented_sieve(long long a,long long b){
    sieve(p);
    bool pp[1001];
    memset(pp,1,sizeof pp);
    for(long long i = 2;i*i<=b;i++){
        for(long long j = a;j<=b;j++){
            if(p[i]){
                if(j == i)
```

```
                    continue;
                if(j % i == 0)
                    pp[j-a] = 0;
            }
        }
    }
    int res = 1;
    for(long long i = a;i<b;i++)
        res += pp[i-a];

    return res;
}
```

Ques. Prime Generator

Given n and m (<= 10^9) such that (n-m) <= 10^5. Generate all the primes between m and n.

https://www.codechef.com/problems/PRIME1

Optimized Segmented Sieve

```
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

#define MAXN 1000000050
```

```
#define NN 50000
bool p[NN + 10], pp[NN + 10];
vector<int> primes;


//MAXN is 1e9
//NN is the size of sieve needed
//We will find the primes till NN
//in our sieve
//Size of NN is sqrt(MAXN)
//Approx 50000

void sieve(){
    for(int i = 0;i<=NN;i++)
        p[i] = 1;

    for(int i = 2;i*i<=MAXN;i++){
        if(p[i]){

            //build the primes vector by storing all
            //prime numbers in the primes vector
            primes.push_back(i);
            for(int j = i*i; j <= NN;j += i)
                p[j] = 0;
        }
    }

    p[0] = 0,p[1] = 0;
```

```
    return;
}

void segmented_sieve(long long a,long long b){

    for(int i = 0;i<=(b-a);i++){
        pp[i] = 1;
    }


    //Iterate through all the primes stored upto
    //sqrt(b)
    for(int i = 0;primes[i]*primes[i] <= b && i < primes.
size();i++){

        //find the first multiple of this prime
        //in the range [a,b]

        long long start;
        a % primes[i] == 0 ? start = a : start = a + (pri
mes[i] - (a % primes[i]));

        for(long long j = start; j <= b; j += primes[i]){
                if(primes[i] == j)
                    continue;
                if(j % primes[i] == 0)
                    pp[j-a] = 0;
```

```
        }
    }

    for(long long i = a;i<=b;i++){
        if(i == 1)continue;
        if(pp[i-a])
            printf("%lld\n",i);
    }

    return;
}

int main(){

    //Calculate all primes upto sqrt(MAXN)
    sieve();

    int t;
    scanf("%d",&t);
    while(t--){
        long long m,n;
        scanf("%lld %lld",&m,&n);
        segmented_sieve(m,n);
        printf("\n");
    }

    return 0;
}
```

Created ßy Vishal Panwar