|  |  |
| --- | --- |
| **Num range** | **#of Primes** |
| 1 – 10 | 4 |
| 1 – 100 | 25 |
| 1 – 1E3 | 168 |
| 1 – 1E4 | 1229 |
| 1 – 1E5 | 9592 (1E4) |
| 1 – 1E6 | 78498 (8E4) |
| 1 – 1E7 | 664579 (7E5) |
| 1 – 1E8 | 5761455 (6E6) |
| 1 – 1E9 | 50847534(5.1E7) |

**Sieve of Eratosthenes:**

const int stat\_sz= 1E6;

const int prime\_sz= 8E4;

bool stat[stat\_sz+10];

int primes[prime\_sz];

void sieve(int N)

{

int sqrtN= sqrt(double(N));

for(int i= 3, j; i<=sqrtN; i+=2)

{

if(stat[i] == 0)

{

for(j = i\*i; j<=N; j+= i<<1)

{

stat[j]= 1;

}

}

}

primes[0]= 2;

for(int i= 3, j= 1; i<=N; i+= 2)

{

if(stat[i] == 0){

primes[j]= i;

j++;

}

}

}

**Prime factorization:**

template <typename T>

int factorize(T num)

{

int factcc= 0;

T sqrtN= sqrt((double) num);

for(int i= 0; primes[i]<=sqrtN; i++)

{

if(num % primes[i] == 0)

{

primeFactors[factcc]= primes[i];

factcc++;

while(num % primes[i] == 0){

num /= primes[i];

}

}

sqrtN= sqrt((double) num);

}

if(num != 1){

primeFactors[factcc]= num;

factcc++;

}

return factcc;

}