Number Theory – 2(Primes and factorization)

<https://www.hackerearth.com/practice/math/number-theory/basic-number-theory-2/tutorial/>

**The better approach to check whether the number is prime or not.**

If you have two positive numbers N and D, such that N is divisible by D and D is less than the square root of N.

* (N/D) must be greater than the square root of N.
* N is also divisible by (N/D). If there is a divisor of N that is less than the square root of N, then there will be a divisor of N that is greater than square root of N. You will have to traverse till the square root of N.

**Note**: You are generating all the divisors of N and if the count of divisors is greater than 2, then the number is composite.

For example, if N=50, N=7 (floor value). You will iterate from 1 to 7 and count the number of divisors of N. The divisors of N are 1, 50; 2, 25; 5,10. You have 6 divisors of 50, and therefore, it is not prime.

void checkprime(int N) {

int count = 0;

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

if( N % i == 0) {

if( i \* i == N )

count++;

else // i < sqrt(N) and (N / i) > sqrt(N)

count += 2;

}

}

if(count == 2)

cout << N << “ is a prime number.” << endl;

else

cout << N << “ is not a prime number.” << endl;

}

*Time complexity*

The time complexity of this function is O(sqrt(N)) because you traverse from 1 to sqrt(N).

**Sieve of Eratosthenes**

You can use the *Sieve of Eratosthenes* to find all the prime numbers that are less than or equal to a given number N or to find out whether a number is a prime number.

The basic idea behind the Sieve of Eratosthenes is that at each iteration one prime number is picked up and all its multiples are eliminated. After the elimination process is complete, all the unmarked numbers that remain are prime.

*Pseudo code*

1. Mark all the numbers as prime numbers except 1
2. Traverse over each prime numbers smaller than sqrt(N)
3. For each prime number, mark its multiples as composite numbers
4. Numbers, which are not the multiples of any number, will remain marked as prime number and others will change to composite numbers.

void sieve(int N) {

bool isPrime[N+1];

for(int i = 0; i <= N;++i) {

isPrime[i] = true;

}

isPrime[0] = false;

isPrime[1] = false;

for(int i = 2; i \* i <= N; ++i) {

if(isPrime[i] == true) { //Mark all the multiples of i as composite numbers

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

isPrime[j] = false;

}

}

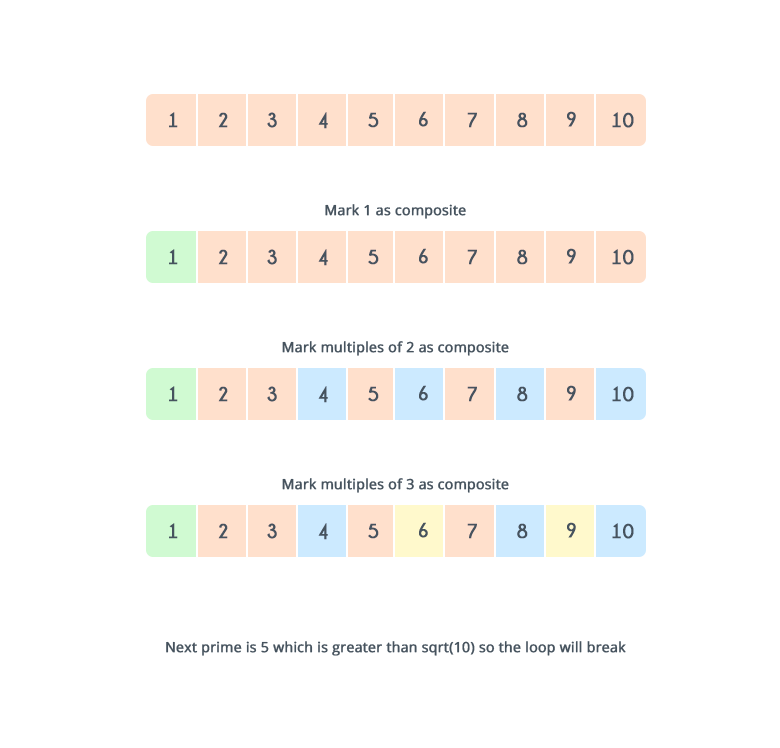
}

This code will compute all the prime numbers that are smaller than or equal to N.

Let us compute prime numbers where N=10.

1. Mark all the numbers as prime
2. Mark 1 as composite

In each iteration, check if a number is prime or not, if it is then mark all of its multiple as composite.



The prime numbers are 2, 3, 5, and 7.

**Time complexity**  
The inner loop that runs for each element is as follows:

1. If i = 2, inner loop runs N / 2 times
2. If i = 3, inner loop runs N / 3 times
3. If i = 5, inner loop runs N / 5 times

*Total complexity*

N \* (½ + ⅓ + ⅕ + … ) = O(NloglogN)