Malay Nath

SEIVE OF ERATOSTHENES

The Seive of Eratosthenes is an efficient algorithm for finding all prime numbers up to a given integer n.

```
#include <stdio.h>
#include <stdbool.h>

void sieve_of_erato(int n) {
    bool is_prime[n+1]; // Boolean array to mark prime numbers

// Initialize all entries as true
    for (int i=0; i<=n; i++) {
        is_prime[i] = true;
    }

is_prime[0] = is_prime[1] = false; // 0 and 1 are not prime numbers
for (int p=2; p*p<=n; p++) {
    // If is_prime[p] is true, mark its multiples as false
    if (is_prime[p]) {
        for (int i=p*p; i<=n; i+=p) {
            is_prime[i] = false;
        }
    }
}
// Start with p=2, the smallest prime number.
```

Initially, we assume all numbers from 0 to n are prime. This loop runs from 0 to n and sets all values of is_prime[i] to true.

iterate over each no. p from 2 to $\sqrt{n}$. if is_prime[p] is true, p is prime no.

start marking multiples of p as false, beginning from p^2. Increment by p in each iteration to cover all multiples of p.

Why start from p^2?
- All smaller multiples $(p, 2p, \ldots, (p-1)p)$ are already marked by smaller primes.

// print all prime numbers

```
printf ("Prime number upto %d: \n", n);
for( int i=2 ; i<=n ; i++) {
    if (is-prime[i]) {
        printf ("%d", i);
    }
}
printf("\n");
}

int main( ) {
int n;
printf("Enter the value of n: ");
scanf("%d", &n);
seive_of_eratb(n);

return 0;
}
```

TIME COMPLEXITY = O(n(log(logn)))

TWINS. → implement using Seive-of-Eratosthenes.

- They are both prime. A prime number is an integer greater than 1 that has no positive divisors other than 1 and itself.
- Their absolute difference is exactly equal to 2 (i.e $|j-i|=2$).
- Given an inclusive interval of integers from n to m, find the number of pairs of twins there are in the interval (i.e [n, m])?

Note that pairs (i, j) and (j, i) are considered to be the same pair.

- Output format

Print a single integer denoting the number of pairs of twins.

[L, R]
[3, 13]

Sample output = 3.

There are three pairs of twins : (3,5), (5,7) and (11, 13).

Complexity

1. T.C = Sieve of Eratosthenes : O(n log log n), where n = R
   → Counting Twins : O (R-L+1) / Filtering Prime : O(R-L+1)
   → Finding twin pairs : O(number of primes in range).
2. S.C = O(MAX) for the is-prime array.
   or
   O(R) for the is-prime array.

FIND PRIME NUMBER USING SEIVE OF ERATOSTHENES

```
// Function to count twin primes in the range [L,R]
int count_twin_primes(int L, int R){
    bool is_prime[MAX+1]; // Declare an array to make primes
    sieve_of_erato(R, is_prime);
    int twin_count=0;
// Iterate through the range [L,R] & check for twin primes
    for(int i=L; i<=R-2; i++){
        if(is_prime[i] && is_prime[i+2]){
            twin_count++;
        }
    }
    return twin_count++;
}

int main(){
    int L,R;
    printf("Range");
    scanf("%d %d", &L, &R);
    if(R > MAX){
        printf("", MAX);
        return 1;
    }
    int result = count_twin_primes(L,R);
    printf("", L, R, result);
    return 0;
}
```