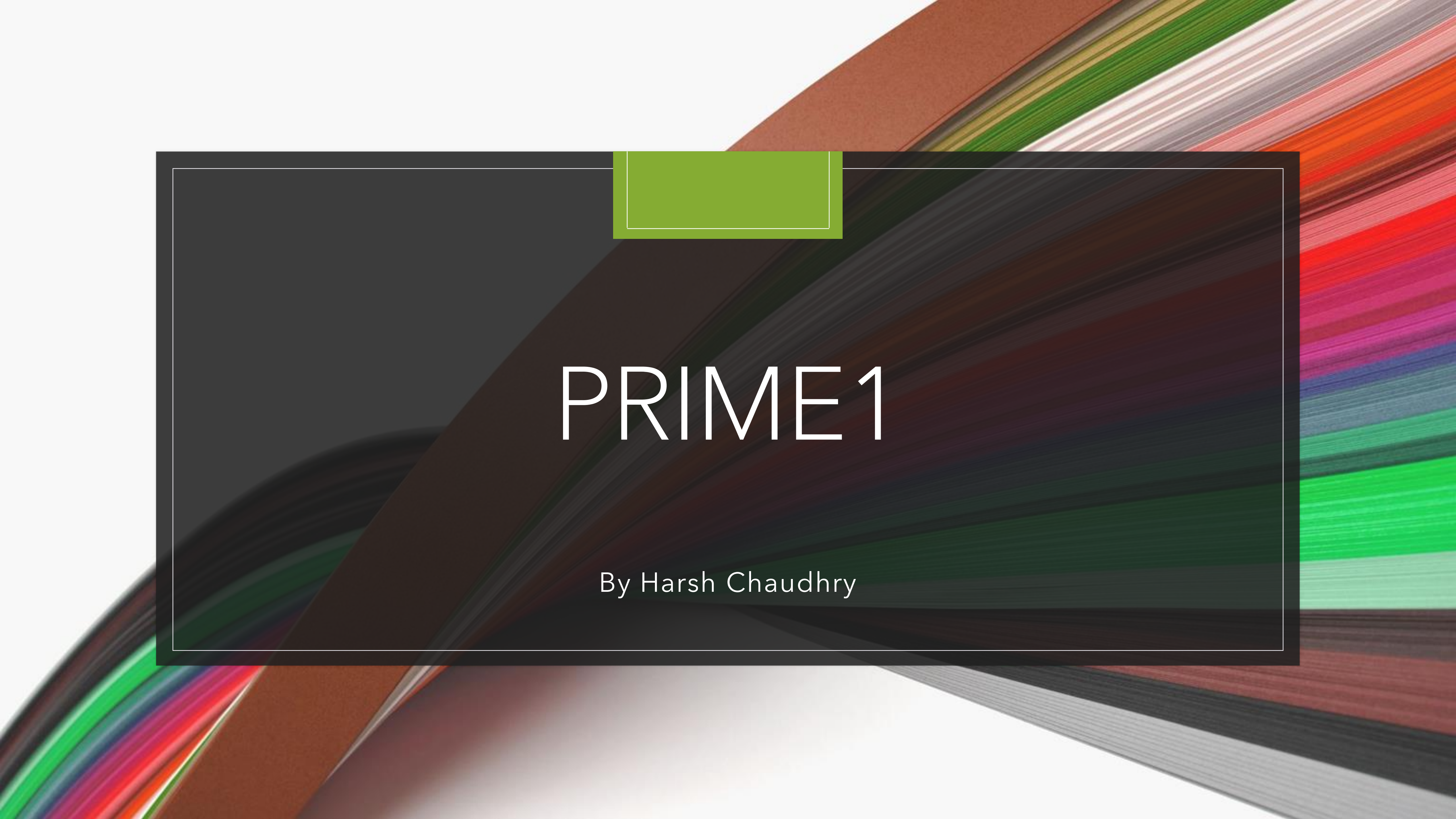


PRIME1

By Harsh Chaudhry

Problem and i/o

- **Inputs**
- For a given ‘t’ number of test cases (ranges) where t<=10.
- Range is defined by m and n where 1 <= m <= n <= 1000000000, n-m<=100000.
- First line of input will give the number of testa cases and subsequent lines will given m and n.

- **Problem**
- Find prime numbers in the given range.

- **Output**
- Enter each prime number in a new line
- There needs to be an extra newline between primes of ranges

Segmented sieve in Range

- Works on the basis of simple sieve. Simple sieve in itself is applied in it.
- Cuts down on calculation as simple sieve is only applied on a small segment.
- The small segment is calculated by taking floor of √max integer in range +1.
- The primes of the rest of range are calculated using the primes derived from the result of simple sieve on the small segment.
- In call cases, evaluation starts from 2 as no primes exist below that number.
- Lets try implementing simple sieve.

Simple Sieve (till 24)

[F]	[F]	[2,F]	[3,F]	[4,F]
[5,F]	[6,F]	[7,F]	[8,F]	[9,F]
[10,F]	[11,F]	[12,F]	[13,F]	[4,F]
[15,F]	[16,F]	[17,F]	[18,F]	[19,F]
[20,F]	[21,F]	[22,F]	[23,F]	[24,F]

Simple Sieve (prime: 2)

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,F]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,F]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,F]	[22,T]	[23,F]	[24,T]

Simple Sieve (prime: 3)

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,T]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,T]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,T]	[22,T]	[23,F]	[24,T]

Simple Sieve (prime: 3)

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,T]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,T]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,T]	[22,T]	[23,F]	[24,T]

Simple Sieve (prime: 5 and the rest)

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,T]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,T]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,T]	[22,T]	[23,F]	[24,T]

Segmented Sieve (till 24)

- For segmented Sieve, we break the range in two parts
  - 1st part: $\sqrt{24} + 1 = 5$.
  - 2nd part: rest of the range we find primes in.
- To find primes in the rest of the range, we use the primes in the 1st part and find their multiples in the 2nd range.
- Everything that was not a multiple in the first part, is a prime number.

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]	[5,F]

[6,F]	[7,F]	[8,F]	[9,F]	[10,F]
[11,F]	[12,F]	[13,F]	[14,F]	[15,F]
[16,F]	[17,F]	[18,F]	[19,F]	[20,F]
[21,F]	[22,F]	[23,F]	[24,F]	

Segmented Sieve (discard multiples)

[2,F]	[3,F]	[5,F]

[6,T]	[7,F]	[8,T]	[9,T]	[10,T]
[11,F]	[12,T]	[13,F]	[14,F]	[15,T]
[16,T]	[17,F]	[18,T]	[19,F]	[20,T]
[21,T]	[22,T]	[23,F]	[24,T]	

Simple Sieve

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,T]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,T]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,T]	[22,T]	[23,F]	[24,T]

Segmented Sieve

[0,F]	[1,F]	[2,F]	[3,F]	[4,T]
[5,F]	[6,T]	[7,F]	[8,T]	[9,T]
[10,T]	[11,F]	[12,T]	[13,F]	[14,T]
[15,T]	[16,T]	[17,F]	[18,T]	[19,F]
[20,T]	[21,T]	[22,T]	[23,F]	[24,T]

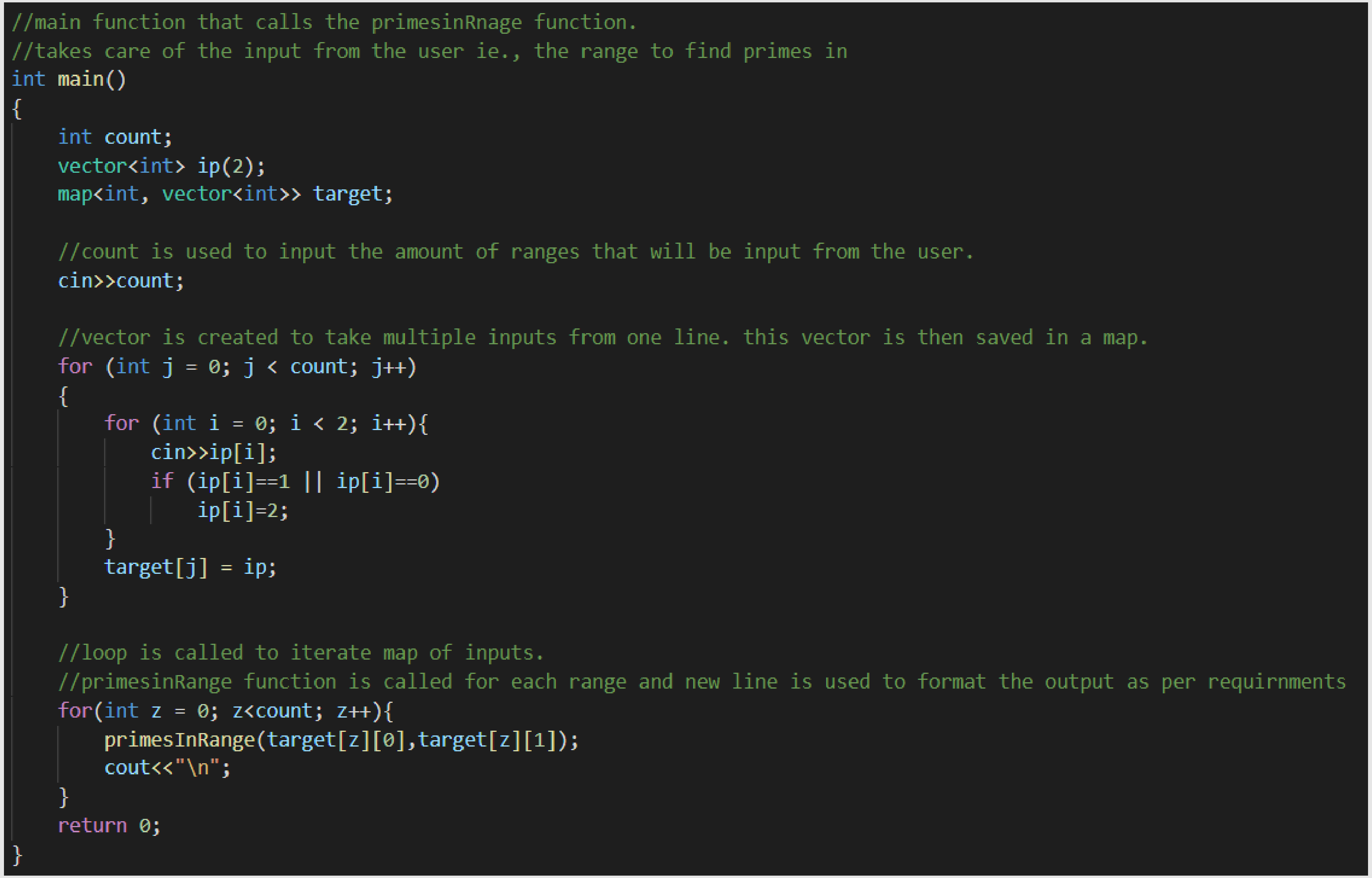
Simple Sieve

- Goes through entire range to find primes.
- Calculates multiples of every prime in range
- Easier to implement.
- Faster than brute force and dynamic programming combination.
- Time complexity: n*log(log n). Works the same way for small integers and ranges. But in larger numbers disk paging might be required, which slows down the process.

Segmented Sieve

- Goes through smaller range to find primes.
- Does not need to calculate multiples of every prime in range.
- More time consuming to implement.
- Faster than simple sieve and dynamic programming combination.
- Same time complexity but is better as it requires less space hence faster for larger ranges or bigger integers.

Segmented Sieve in given Range (main)



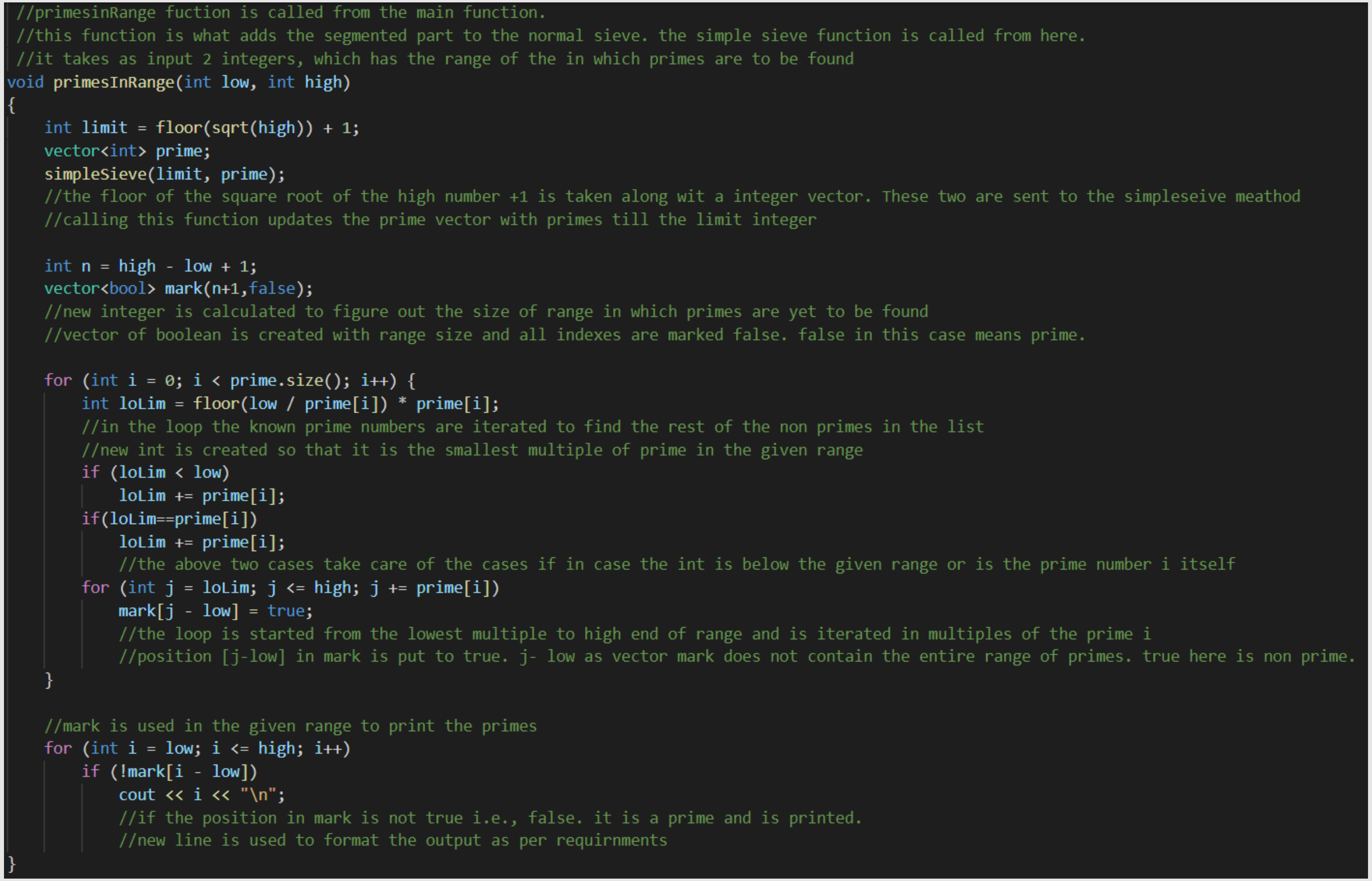
```cpp
//main function that calls the primesinRnage function.
//takes care of the input from the user ie., the range to find primes in
int main()
{
    int count;
    vector<int> ip(2);
    map<int, vector<int>> target;

    //count is used to input the amount of ranges that will be input from the user.
    cin>>count;

    //vector is created to take multiple inputs from one line. this vector is then saved in a map.
    for (int j = 0; j < count; j++)
    {
        for (int i = 0; i < 2; i++){
            cin>>ip[i];
            if (ip[i]==1 || ip[i]==0)
                ip[i]=2;
        }
        target[j] = ip;
    }

    //loop is called to iterate map of inputs.
    //primesinRange function is called for each range and new line is used to format the output as per requirnments
    for(int z = 0; z<count; z++){
        primesInRange(target[z][0],target[z][1]);
        cout<<"\n";
    }
    return 0;
}
```

Segmented Sieve in given Range (Segment)



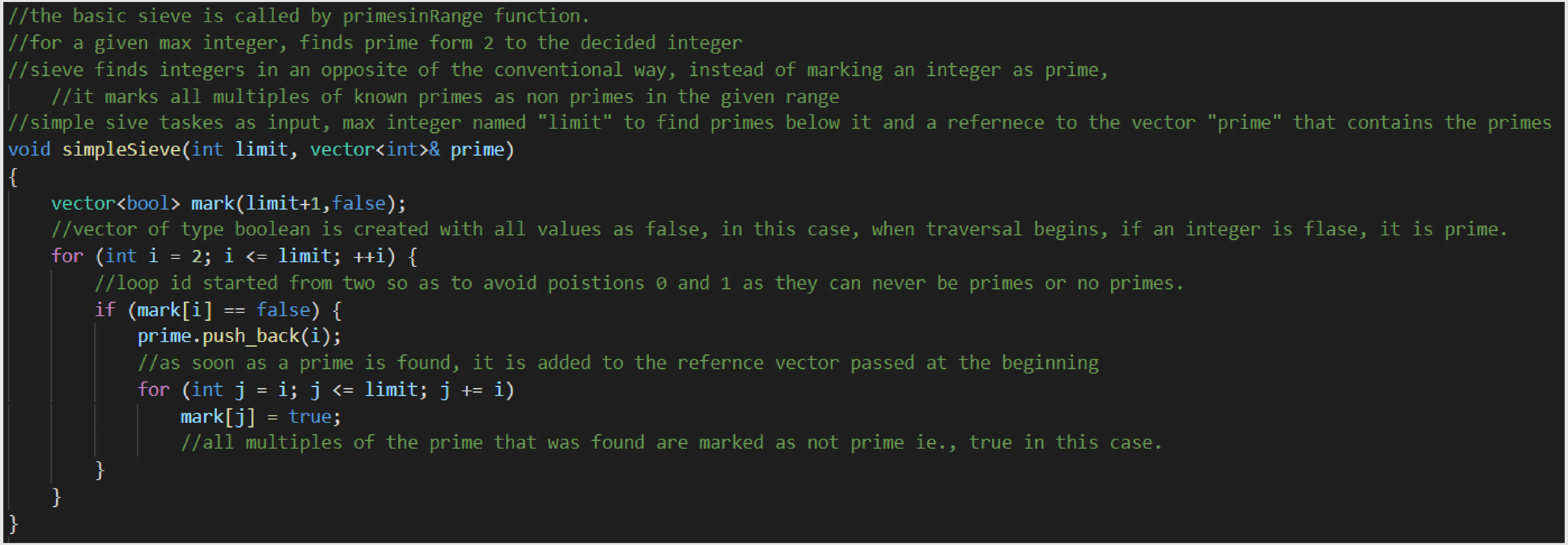
```
 //primesinRange fuction is called from the main function.
 //this function is what adds the segmented part to the normal sieve. the simple sieve function is called from here.
 //it takes as input 2 integers, which has the range of the in which primes are to be found
void primesInRange(int low, int high)
{
    int limit = floor(sqrt(high)) + 1;
    vector<int> prime;
    simpleSieve(limit, prime);
    //the floor of the square root of the high number +1 is taken along wit a integer vector. These two are sent to the simpleseive meathod
    //calling this function updates the prime vector with primes till the limit integer

    int n = high - low + 1;
    vector<bool> mark(n+1,false);
    //new integer is calculated to figure out the size of range in which primes are yet to be found
    //vector of boolean is created with range size and all indexes are marked false. false in this case means prime.

    for (int i = 0; i < prime.size(); i++) {
        int loLim = floor(low / prime[i]) * prime[i];
        //in the loop the known prime numbers are iterated to find the rest of the non primes in the list
        //new int is created so that it is the smallest multiple of prime in the given range
        if (loLim < low)
            loLim += prime[i];
        if(loLim==prime[i])
            loLim += prime[i];
            //the above two cases take care of the cases if in case the int is below the given range or is the prime number i itself
        for (int j = loLim; j <= high; j += prime[i])
            mark[j - low] = true;
            //the loop is started from the lowest multiple to high end of range and is iterated in multiples of the prime i
            //position [j-low] in mark is put to true. j- low as vector mark does not contain the entire range of primes. true here is non prime.
    }

    //mark is used in the given range to print the primes
    for (int i = low; i <= high; i++)
        if (!mark[i - low])
            cout << i << "\n";
            //if the position in mark is not true i.e., false. it is a prime and is printed.
            //new line is used to format the output as per requirnments
}
```

Segmented Sieve in given Range (SimpleSieve)



```
//the basic sieve is called by primesinRange function.
//for a given max integer, finds prime form 2 to the decided integer
//sieve finds integers in an opposite of the conventional way, instead of marking an integer as prime,
    //it marks all multiples of known primes as non primes in the given range
//simple sive taskes as input, max integer named "limit" to find primes below it and a refernece to the vector "prime" that contains the primes
void simpleSieve(int limit, vector<int>& prime)
{
    vector<bool> mark(limit+1,false);
    //vector of type boolean is created with all values as false, in this case, when traversal begins, if an integer is flase, it is prime.
    for (int i = 2; i <= limit; ++i) {
        //loop id started from two so as to avoid poistions 0 and 1 as they can never be primes or no primes.
        if (mark[i] == false) {
            prime.push_back(i);
            //as soon as a prime is found, it is added to the refernce vector passed at the beginning
            for (int j = i; j <= limit; j += i)
                mark[j] = true;
                //all multiples of the prime that was found are marked as not prime ie., true in this case.
        }
    }
}
```

THANK YOU.