Segmented sieve

Segmented Sieve (Background)

Below are basic steps to get idea how Segmented Sieve works

1. Use Simple Sieve to find all primes upto a predefined limit (square root of 'high' is used in below code) and store these primes in an array "prime[]". Basically we call Simple Sieve for a limit and we not only find prime numbers, but also puts them in a separate array prime[].
2. Create an array mark[high-low+1]. Here we need only O(n) space where **n** is number of elements in given range.
3. Iterate through all primes found in step 1. For every prime, mark its multiples in given range [low..high].

```
// C++ program to print print all primes in a range
// using concept of Segmented Sieve
#include <bits/stdc++.h>
using namespace std;

// This functions finds all primes smaller than limit
// using simple sieve of eratosthenes.  It stores found
// primes in vector prime[]
void simpleSieve(int limit, vector<int>& prime)
{
    bool mark[limit + 1];
    memset(mark, true, sizeof(mark));

    mark[0]=false;
    mark[1]=false;

    for (int i = 2; i <= limit; ++i) {
        if (mark[i] == true) {
            // If not marked yet, then its a prime
            prime.push_back(i);
            for (int j = i*2; j <= limit; j += i)
                mark[j] = false;
        }
    }
}

// Finds all prime numbers in given range using
// segmented sieve
void primesInRange(int low, int high)
{
    // Comput all primes smaller or equal to
    // square root of high using simple sieve
    int limit = floor(sqrt(high)) + 1;
    vector<int> prime;
    simpleSieve(limit, prime);

    // Count of elements in given range
    int n = high - low + 1;

    // Declaring boolean only for [low, high]
    bool mark[n + 1];
    memset(mark, true, sizeof(mark));
```

```
    // Use the found primes by simpleSieve() to find
    // primes in given range
    for (int i = 0; i < prime.size(); i++) {
        // Find the minimum number in [low..high] that is
        // a multiple of prime[i] (divisible by prime[i])
        int loLim = floor(low / prime[i]) * prime[i];
        if (loLim < low)
            loLim += prime[i];
        if(loLim==prime[i])
            loLim += prime[i];
        /*  Mark multiples of prime[i] in [low..high]:
            We are marking j - low for j, i.e. each number
            in range [low, high] is mapped to [0, high - low]
            so if range is [50, 100]  marking 50 corresponds
            to marking 0, marking 51 corresponds to 1 and
            so on. In this way we need to allocate space only
            for range  */
        for (int j = loLim; j <= high; j += prime[i])
            mark[j - low] = false;
    }

    // Numbers which are not marked in range, are prime
    for (int i = low; i <= high; i++)
        if (mark[i - low])
            cout << i << "  ";
}

// Driver program to test above function
int main()
{
    int low = 10, high = 100;
    primesInRange(low, high);
    return 0;
}
```