GeeksforGeeks

Related Articles Save for later

Segmented Sieve

Difficulty Level : Hard • Last Updated : 26 Apr, 2021

Given a number n, print all primes smaller than n. For example, if the given number is 10, output 2, 3, 5, 7.

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

A Naive approach is to run a loop from 0 to n-1 and check each number for primeness. A Better Approach is to use Simple Sieve of Eratosthenes.

C

```
// This functions finds all primes smaller than 'limit'
// using simple sieve of eratosthenes.
void simpleSieve(int limit)
{
    // Create a boolean array "mark[0..limit-1]" and
    // initialize all entries of it as true. A value
    // in mark[p] will finally be false if 'p' is Not
    // a prime, else true.
    bool mark[limit];
    for(int i = 0; i<limit; i++) {
      mark[i] = true;
    }
```

```
    // One by one traverse all numbers so that their
    // multiples can be marked as composite.
    for (int p=2; p*p<limit; p++)
    {
        // If p is not changed, then it is a prime
        if (mark[p] == true)
        {
            // Update all multiples of p
            for (int i=p*p; i<limit; i+=p)
                mark[i] = false;
        }
    }

    // Print all prime numbers and store them in prime
    for (int p=2; p<limit; p++)
        if (mark[p] == true)
            cout << p << "  ";
}
```

Java

```
// This functions finds all primes smaller than 'limit'
// using simple sieve of eratosthenes.
static void simpleSieve(int limit)
{

    // Create a boolean array "mark[0..limit-1]" and
    // initialize all entries of it as true. A value
    // in mark[p] will finally be false if 'p' is Not
    // a prime, else true.
    boolean []mark = new boolean[limit];
    Arrays.fill(mark, true);

    // One by one traverse all numbers so that their
    // multiples can be marked as composite.
    for (int p = 2; p * p < limit; p++)
    {
        // If p is not changed, then it is a prime
        if (mark[p] == true)
        {
            // Update all multiples of p
            for (int i = p * p; i < limit; i += p)
                mark[i] = false;
        }
```

```
    }

    // Print all prime numbers and store them in prime
    for (int p = 2; p < limit; p++)
        if (mark[p] == true)
            System.out.print(p + " ");
}

// This code is contributed by rutvik_56.
```

Python3

```python
# This functions finds all primes smaller than 'limit'
# using simple sieve of eratosthenes.
def simpleSieve(limit):

    # Create a boolean array "mark[0..limit-1]" and
    # initialize all entries of it as true. A value
    # in mark[p] will finally be false if 'p' is Not
    # a prime, else true.
    mark = [True for i in range(limit)]

    # One by one traverse all numbers so that their
    # multiples can be marked as composite.
    for p in range(p * p, limit - 1, 1):

        # If p is not changed, then it is a prime
        if (mark[p] == True):

            # Update all multiples of p
            for i in range(p * p, limit - 1, p):
                mark[i] = False

    # Print all prime numbers and store them in prime
    for p in range(2, limit - 1, 1):
        if (mark[p] == True):
            print(p, end=" ")

# This code is contributed by Dharanendra L V.
```

C#

```
// This functions finds all primes smaller than 'limit'
// using simple sieve of eratosthenes.
static void simpleSieve(int limit)
{

    // Create a boolean array "mark[0..limit-1]" and
    // initialize all entries of it as true. A value
    // in mark[p] will finally be false if 'p' is Not
    // a prime, else true.
    bool []mark = new bool[limit];
    Array.Fill(mark, true);

    // One by one traverse all numbers so that their
    // multiples can be marked as composite.
    for (int p = 2; p * p < limit; p++)
    {

        // If p is not changed, then it is a prime
        if (mark[p] == true)
        {

            // Update all multiples of p
            for (int i = p * p; i < limit; i += p)
                mark[i] = false;
        }
    }

    // Print all prime numbers and store them in prime
    for (int p = 2; p < limit; p++)
        if (mark[p] == true)
            Console.Write(p + " ");
}

// This code is contributed by pratham76.
```

Javascript

```
<script>

// This functions finds all primes smaller than 'limit'
// using simple sieve of eratosthenes.
function simpleSieve(limit)
{
```

```
    // Create a boolean array "mark[0..limit-1]" and
    // initialize all entries of it as true. A value
    // in mark[p] will finally be false if 'p' is Not
    // a prime, else true.
    var mark = Array(limit).fill(true);


    // One by one traverse all numbers so that their
    // multiples can be marked as composite.
    for (p = 2; p * p < limit; p++)
    {
        // If p is not changed, then it is a prime
        if (mark[p] == true)
        {
            // Update all multiples of p
            for (i = p * p; i < limit; i += p)
                mark[i] = false;
        }
    }

    // Print all prime numbers and store them in prime
    for (p = 2; p < limit; p++)
        if (mark[p] == true)
            document.write(p + " ");
}

// This code is contributed by todaysgaurav

</script>
```

Problems with Simple Sieve:

The Sieve of Eratosthenes looks good, but consider the situation when n is large, the Simple Sieve faces the following issues.

- An array of size Θ(n) may not fit in memory
- The simple Sieve is not cached friendly even for slightly bigger n. The algorithm traverses the array without locality of reference

Segmented Sieve

The idea of a segmented sieve is to divide the range [0..n-1] in different segments and compute primes in all segments one by one. This algorithm first uses Simple Sieve to find primes smaller than or equal to √(n). Below are steps used in Segmented Sieve.

1. Use Simple Sieve to find all primes up to the square root of 'n' and store these primes in an array "prime[]". Store the found primes in an array 'prime[]'.
2. We need all primes in the range [0..n-1]. We divide this range into different segments such that the size of every segment is at-most √n
3. Do following for every segment [low..high]
   - Create an array mark[high-low+1]. Here we need only O(x) space where **x** is a number of elements in a given range.
   - Iterate through all primes found in step 1. For every prime, mark its multiples in the given range [low..high].

In Simple Sieve, we needed O(n) space which may not be feasible for large n. Here we need O(√n) space and we process smaller ranges at a time (locality of reference)

Below is the implementation of the above idea.

C++

```cpp
// C++ program to print print all primes smaller than
// n using segmented sieve
#include <bits/stdc++.h>
using namespace std;

// This functions finds all primes smaller than 'limit'
// using simple sieve of eratosthenes. It also stores
// found primes in vector prime[]
void simpleSieve(int limit, vector<int> &prime)
{
    // Create a boolean array "mark[0..n-1]" and initialize
    // all entries of it as true. A value in mark[p] will
    // finally be false if 'p' is Not a prime, else true.
    vector<bool> mark(limit + 1, true);

    for (int p=2; p*p<limit; p++)
    {
        // If p is not changed, then it is a prime
        if (mark[p] == true)
        {
            // Update all multiples of p
            for (int i=p*p; i<limit; i+=p)
                mark[i] = false;
        }
    }

    // Print all prime numbers and store them in prime
    for (int p=2; p<limit; p++)
    {
        if (mark[p] == true)
        {
            prime.push_back(p);
            cout << p << " ";
        }
    }
}

// Prints all prime numbers smaller than 'n'
void segmentedSieve(int n)
{
    // Compute all primes smaller than or equal
    // to square root of n using simple sieve
    int limit = floor(sqrt(n))+1;
    vector<int> prime;
```

```
    prime.reserve(limit);
    simpleSieve(limit, prime);

    // Divide the range [0..n-1] in different segments
    // We have chosen segment size as sqrt(n).
    int low = limit;
    int high = 2*limit;

    // While all segments of range [0..n-1] are not processed,
    // process one segment at a time
    while (low < n)
    {
        if (high >= n)
           high = n;

        // To mark primes in current range. A value in mark[i]
        // will finally be false if 'i-low' is Not a prime,
        // else true.
        bool mark[limit+1];
        memset(mark, true, sizeof(mark));

        // Use the found primes by simpleSieve() to find
        // primes in current 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])
            // For example, if low is 31 and prime[i] is 3,
            // we start with 33.
            int loLim = floor(low/prime[i]) * prime[i];
            if (loLim < low)
                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 as false are prime
```

```
        for (int i = low; i<high; i++)
            if (mark[i - low] == true)
                cout << i << " ";

        // Update low and high for next segment
        low = low + limit;
        high = high + limit;
    }
}

// Driver program to test above function
int main()
{
    int n = 100000;
    cout << "Primes smaller than " << n << ":n";
    segmentedSieve(n);
    return 0;
}
```

Java

```
// Java program to print print all primes smaller than
// n using segmented sieve


import java.util.Vector;
import static java.lang.Math.sqrt;
import static java.lang.Math.floor;

class Test
{
    // This methid finds all primes smaller than 'limit'
    // using simple sieve of eratosthenes. It also stores
    // found primes in vector prime[]
    static void simpleSieve(int limit, Vector<Integer> prime)
    {
        // Create a boolean array "mark[0..n-1]" and initialize
        // all entries of it as true. A value in mark[p] will
        // finally be false if 'p' is Not a prime, else true.
        boolean mark[] = new boolean[limit+1];

        for (int i = 0; i < mark.length; i++)
            mark[i] = true;
```

```
        for (int p=2; p*p<limit; p++)
        {
            // If p is not changed, then it is a prime
            if (mark[p] == true)
            {
                // Update all multiples of p
                for (int i=p*p; i<limit; i+=p)
                    mark[i] = false;
            }
        }

        // Print all prime numbers and store them in prime
        for (int p=2; p<limit; p++)
        {
            if (mark[p] == true)
            {
                prime.add(p);
                System.out.print(p + "  ");
            }
        }
    }

    // Prints all prime numbers smaller than 'n'
    static void segmentedSieve(int n)
    {
        // Compute all primes smaller than or equal
        // to square root of n using simple sieve
        int limit = (int) (floor(sqrt(n))+1);
        Vector<Integer> prime = new Vector<>();
        simpleSieve(limit, prime);

        // Divide the range [0..n-1] in different segments
        // We have chosen segment size as sqrt(n).
        int low  = limit;
        int high = 2*limit;

        // While all segments of range [0..n-1] are not processed,
        // process one segment at a time
        while (low < n)
        {
            if (high >= n)
                high = n;

            // To mark primes in current range. A value in mark[i]
            // will finally be false if 'i-low' is Not a prime,
```

```
        // else true.
        boolean mark[] = new boolean[limit+1];

        for (int i = 0; i < mark.length; i++)
            mark[i] = true;

        // Use the found primes by simpleSieve() to find
        // primes in current range
        for (int i = 0; i < prime.size(); i++)
        {
            // Find the minimum number in [low..high] that is
            // a multiple of prime.get(i) (divisible by prime.get(i))
            // For example, if low is 31 and prime.get(i) is 3,
            // we start with 33.
            int loLim = (int) (floor(low/prime.get(i)) * prime.get(i)
            if (loLim < low)
                loLim += prime.get(i);

            /*  Mark multiples of prime.get(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.get(i))
                mark[j-low] = false;
        }

        // Numbers which are not marked as false are prime
        for (int i = low; i<high; i++)
            if (mark[i - low] == true)
                System.out.print(i + "  ");

        // Update low and high for next segment
        low  = low + limit;
        high = high + limit;
    }
}

// Driver method
public static void main(String args[])
{
    int n = 100;
    System.out.println("Primes smaller than " + n + ":");
```

```
        segmentedSieve(n);
    }
`
```

Python3

```python
# Python3 program to print all primes
# smaller than n, using segmented sieve
import math
prime = []

# This method finds all primes
# smaller than 'limit' using
# simple sieve of eratosthenes.
# It also stores found primes in list prime
def simpleSieve(limit):

    # Create a boolean list "mark[0..n-1]" and
    # initialize all entries of it as True.
    # A value in mark[p] will finally be False
    # if 'p' is Not a prime, else True.
    mark = [True for i in range(limit + 1)]
    p = 2
    while (p * p <= limit):

        # If p is not changed, then it is a prime
        if (mark[p] == True):

            # Update all multiples of p
            for i in range(p * p, limit + 1, p):
                mark[i] = False
        p += 1

    # Print all prime numbers
    # and store them in prime
    for p in range(2, limit):
        if mark[p]:
            prime.append(p)
            print(p,end = " ")

# Prints all prime numbers smaller than 'n'
def segmentedSieve(n):

    # Compute all primes smaller than or equal
    # to square root of n using simple sieve
```

```
    limit = int(math.floor(math.sqrt(n)) + 1)
    simpleSieve(limit)

    # Divide the range [0..n-1] in different segments
    # We have chosen segment size as sqrt(n).
    low = limit
    high = limit * 2

    # While all segments of range [0..n-1] are not processed,
    # process one segment at a time
    while low < n:
        if high >= n:
            high = n

        # To mark primes in current range. A value in mark[i]
        # will finally be False if 'i-low' is Not a prime,
        # else True.
        mark = [True for i in range(limit + 1)]

        # Use the found primes by simpleSieve()
        # to find primes in current range
        for i in range(len(prime)):

            # Find the minimum number in [low..high]
            # that is a multiple of prime[i]
            # (divisible by prime[i])
            # For example, if low is 31 and prime[i] is 3,
            # we start with 33.
            loLim = int(math.floor(low / prime[i]) *
                                             prime[i])
            if loLim < low:
                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 j in range(loLim, high, prime[i]):
                mark[j - low] = False

        # Numbers which are not marked as False are prime
        for i in range(low, high):
```

```
            if mark[i - low]:
                print(i, end = " ")

        # Update low and high for next segment
        low = low + limit
        high = high + limit

# Driver Code
n = 100
print("Primes smaller than", n, ":")
segmentedSieve(100)

# This code is contributed by bhavyadeep
```

C#

```
// C# program to print print
// all primes smaller than
// n using segmented sieve
using System;
using System.Collections;

class GFG
{
    // This methid finds all primes
    // smaller than 'limit' using simple
    // sieve of eratosthenes. It also stores
    // found primes in vector prime[]
    static void simpleSieve(int limit,
                            ArrayList prime)
    {
        // Create a boolean array "mark[0..n-1]"
        // and initialize all entries of it as
        // true. A value in mark[p] will finally be
        // false if 'p' is Not a prime, else true.
        bool[] mark = new bool[limit + 1];

        for (int i = 0; i < mark.Length; i++)
            mark[i] = true;

        for (int p = 2; p * p < limit; p++)
        {
            // If p is not changed, then it is a prime
            if (mark[p] == true)
```

```
            {
                // Update all multiples of p
                for (int i = p * p; i < limit; i += p)
                    mark[i] = false;
            }
        }

        // Print all prime numbers and store them in prime
        for (int p = 2; p < limit; p++)
        {
            if (mark[p] == true)
            {
                prime.Add(p);
                Console.Write(p + " ");
            }
        }
    }

    // Prints all prime numbers smaller than 'n'
    static void segmentedSieve(int n)
    {
        // Compute all primes smaller than or equal
        // to square root of n using simple sieve
        int limit = (int) (Math.Floor(Math.Sqrt(n)) + 1);
        ArrayList prime = new ArrayList();
        simpleSieve(limit, prime);

        // Divide the range [0..n-1] in
        // different segments We have chosen
        // segment size as sqrt(n).
        int low = limit;
        int high = 2*limit;

        // While all segments of range
        // [0..n-1] are not processed,
        // process one segment at a time
        while (low < n)
        {
            if (high >= n)
                high = n;

            // To mark primes in current range.
            // A value in mark[i] will finally
            // be false if 'i-low' is Not a prime,
            // else true.
```

```
            bool[] mark = new bool[limit + 1];

            for (int i = 0; i < mark.Length; i++)
                mark[i] = true;

            // Use the found primes by
            // simpleSieve() to find
            // primes in current range
            for (int i = 0; i < prime.Count; i++)
            {
                // Find the minimum number in
                // [low..high] that is a multiple
                // of prime.get(i) (divisible by
                // prime.get(i)) For example,
                // if low is 31 and prime.get(i)
                //  is 3, we start with 33.
                int loLim = ((int)Math.Floor((double)(low /
                            (int)prime[i])) * (int)prime[i]);
                if (loLim < low)
                    loLim += (int)prime[i];

                /* Mark multiples of prime.get(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 += (int)prime[i])
                    mark[j-low] = false;
            }

            // Numbers which are not marked as false are prime
            for (int i = low; i < high; i++)
                if (mark[i - low] == true)
                    Console.Write(i + " ");

            // Update low and high for next segment
            low = low + limit;
            high = high + limit;
        }
    }

    // Driver code
    static void Main()
```

```
    {
        int n = 100;
        Console.WriteLine("Primes smaller than " + n + ":");
        segmentedSieve(n);
    }
}

// This code is contributed by mits
```

Output:

```
Primes smaller than 100:
2 3 5 7 11 13 17 19 23 29 31 37 41
43 47 53 59 61 67 71 73 79 83 89 97
```

Note that time complexity (or a number of operations) by Segmented Sieve is the same as Simple Sieve. It has advantages for large 'n' as it has better locality of reference thus allowing better caching by the CPU and also requires less memory space.
This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course.**

In case you wish to attend **live classes** with experts, please refer **DSA Live Classes for Working Professionals** and **Competitive Programming Live for Students**.

Previous

How is the time complexity of Sieve of Eratosthenes is n*log(log(n))?

Next

Segmented Sieve (Print Primes in a Range)

RECOMMENDED ARTICLES

Page : 1 2

01 Segmented Sieve (Print Primes in a Range)
24, Nov 15

02 Longest sub-array of Prime Numbers using Segmented Sieve

05 Sieve of Atkin
16, Jan 16

06 Sieve of Eratosthenes in 0(n) time complexity
22, Jan 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : Mithun Kumar, BhavyadeepPurswani, aerosayan, kumarayush151, rutvik_56, pratham76, dharanendralv23, scisaif, todaysgaurav

Article Tags : Prime Number, sieve, Mathematical

Practice Tags : Mathematical, Prime Number, sieve

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

Privacy Policy

Contact Us

Copyright Policy

Learn

Algorithms

Data Structures

Languages

CS Subjects

Video Tutorials

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos