Efficient program to print all prime factors of a given number

Given a number n, write an efficient function to print all prime factors of n. For example, if the input number is 12, then output should be “2 2 3″. And if the input number is 315, then output should be “3 3 5 7″.

Following are the steps to find all prime factors.
1) While n is divisible by 2, print 2 and divide n by 2.
2) After step 1, n must be odd. Now start a loop from i = 3 to square root of n. While i divides n, print i and divide n by i, increment i by 2 and continue.
3) If n is a prime number and is greater than 2, then n will not become 1 by above two steps. So print n if it is greater than 2.

```
// Program to print all prime factors
```

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

// A function to print all prime factors of a given number n
void primeFactors(int n)
{
    // Print the number of 2s that divide n
    while (n%2 == 0)
    {
        printf("%d ", 2);
        n = n/2;
    }

    // n must be odd at this point.  So we can skip one element (Note i = i +
    for (int i = 3; i <= sqrt(n); i = i+2)
    {
        // While i divides n, print i and divide n
        while (n%i == 0)
        {
            printf("%d ", i);
            n = n/i;
        }
    }

    // This condition is to handle the case whien n is a prime number
    // greater than 2
    if (n > 2)
        printf ("%d ", n);
}

/* Driver program to test above function */
int main()
{
    int n = 315;
    primeFactors(n);
    return 0;
}
```

Output:

```
3 3 5 7
```

How does this work?
The steps 1 and 2 take care of composite numbers and step 3 takes care of prime numbers. To prove that the complete algorithm works, we need to prove that steps 1 and 2 actually take care of composite numbers. This is clear that step 1 takes care of even numbers. And after step 1, all remaining prime factor must be odd (difference of two prime factors must be at least 2), this explains why i is incremented by 2.
Now the main part is, the loop runs till square root of n not till. To prove that this optimization works, let us consider the following property of composite numbers.
Every composite number has at least one prime factor less than or equal to square root of itself.
This property can be proved using counter statement. Let a and b be two factors of n such that a*b = n. If both are greater than √n, then a.b > √n, * √n, which contradicts the expression "a * b = n".

In step 2 of the above algorithm, we run a loop and do following in loop
a) Find the least prime factor i (must be less than √n,)
b) Remove all occurrences i from n by repeatedly dividing n by i.
c) Repeat steps a and b for divided n and i = i + 2. The steps a and b are repeated till n becomes either 1 or a prime number.

Thanks to **Vishwas Garg** for suggesting the above algorithm. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Find Itinerary from a given list of tickets
- Load Balancing on Servers (Random Algorithm)
- Find number of Employees Under every Employee
- Expected Number of Trials until Success
- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number

Tags: combionatrics, MathematicalAlgo

Me gusta 5 Tweet 4 g+1 6

Writing code in comment? Please use **ideone.com** and share the link here.

82 Comments GeeksforGeeks 1 Login

Recommend 9 Share Sort by Newest

Join the discussion…

doel hazra • 14 days ago

hats off to gaurav rathore . thank you so much .

^ | ˅ • Reply • Share ›

Zu Ba • 16 days ago

There are repeating prime numbers. How it can be modified if it only shows non-repeating prime numbers?

^ | ˅ • Reply • Share ›

Gaurav Rathore ↪ Zu Ba • 15 days ago

You can store the prime number printed in the previous iteration in variable(say prev) and print the current prime number being tested (say cur) if cur!=prev. You can also introduce a exponent count and increment it in each iteration in prev==cur.Later you can print it in exponential form using prev^exp .
In the above code you can achive what you want as follows:

```
void primeFactors(int n)

{

//-----------change 1---------------
int prev=0;

// Print the number of 2s that divide n

while (n%2 == 0)

{
//---------change 2-------------
```

see more

^ | ˅ • Reply • Share ›

doel hazra • 16 days ago

WAP TO ACCEPT A NUMBER AND PRINT ITS FACTORS USING WHILE LOOP . ...HOW TO DO THIS ?

^ | ˅ • Reply • Share ›

Baba • 18 days ago

it is unnecessary printing repeated factor

^ | ˅ • Reply • Share ›

Gaurav Rathore ↪ Baba • 15 days ago

You can store the prime number printed in the previous iteration in variable(say prev) and print the current prime number being tested (say cur) if cur!=prev. You can also introduce a exponent count and increment it in each iteration in prev==cur.Later you can

print it in exponential form using prev^exp .
In the above code you can achive what you want as follows:

```
void primeFactors(int n)

{

//-----------change 1---------------
int prev=0;

// Print the number of 2s that divide n

while (n%2 == 0)

{
//---------change 2-------------
```

see more

• Reply • Share ›

Vatsal • 22 days ago
excedding time limit for n>10^7 ...!
• Reply • Share ›

Sourabh Goel • 22 days ago
Can anyone explain the time complexity of this program?????
1 • Reply • Share ›

karanp • a month ago
625 not working
• Reply • Share ›

Sourabh Goel karanp • 22 days ago
its working. check your code again
• Reply • Share ›

pruthvi raj • a month ago
why to check for if(n>2) in last but one statement, why not if(n>1)? since there will be no way, n value to be 2 at that point as we are already dividing n with 2 in the beginning.
• Reply • Share ›

Menj55 • 2 months ago
Its not working for 4937775. Please update the solution
• Reply • Share ›

RyanAwesomeness ➦ Menj55 • 2 months ago

that is because this program is written with int data type which is range from -32,767 to 32,767. just change everything you see in that code from int to long int which ranges from ...-2,147,483,647 to 2,147,483,647. if you want it to always work no matter how high the number is change everything you see in the code from "int" to "unsigned long long int" which range from 0 to 18,446,744,073,709,551,615

• Reply • Share ›

hayspak • 2 months ago

great dude! hats off!

• Reply • Share ›

angel • 6 months ago

write a program to print square of smallest no. from given 3 nos.

• Reply • Share ›

pruthvi raj • 6 months ago

Just for Confirmation :
Last but one line in primeFactors function (if (n>2)) can be replaced with (if (n>sqrt(n))???
I know both results the same, but just for confirmation am asking.

1 • Reply • Share ›

daghan • 7 months ago

sqrt calculation is costly, it can be cached.
also n /= i is faster than n = n/i

#include<stdio.h>

#include<math.h>

// A function to print all prime factors of a given number n

void primeFactors(int n)

{

int sqrt_n,i;

// Print the number of 2s that divide n

while (n%2 == 0)

{

see more

1 • Reply • Share ›

Kamsin daghan • 2 months ago

sqrt_n = sqrt(n);
for (i = 3; i <= sqrt_n; i += 2)

instead use this

for (i = 3; i <= sqrt(n); i += 2)

because it will run loop "very smaller number of times" as compared to above syntax.

• Reply • Share ›

mahesh • 8 months ago

1.
Write a C program that given an integer ‘n’, prints the number
of integers that are less than or equal to ‘n’ and co-prime to ‘n’

Two integers a and b are said to be relatively prime or co-prime if the only
positive integer that evenly divides both of them is 1. That is, the only
common positive factor of the two numbers is 1. This is equivalent to their
greatest common divisor being 1.

Input Format:

One line containing the value of 'n' , where 1<=n<=10,000

Output Format:

One line containing the number of integers that are co-prime to
n and less than or equal to 'n'

Input

see more

• Reply • Share ›

swaroop mahesh • 7 months ago

What you need is the Euler totient function.

• Reply • Share ›

Mihir • 8 months ago

Efficient code.

2 • Reply • Share ›

Ambuj • 8 months ago

We can do the second step a bit differently:
My version of the solution is as follows(it will print only the unique prime numbers, ex: for number 9 it will print only 3 not 3, 3 but later version can also do that:)
Version 1:(Unique Prime Factor version):

```
void displayPrimeFactor(int number){
int primeFactr[100];
bool isNewPrime;
int count = 0;
int begin =3, end = number;
if(end % 2 == 0){
primeFactr[count++] = 2;
end = end/2;
}
do{
int i = 0; isNewPrime = true;
do{
if(begin % primeFactr[i++] != 0)
```

see more

• Reply • Share ›

jayasurya_j • 10 months ago
is there any efficient way to find the number of factors ?
• Reply • Share ›

Rahul Choudhary jayasurya_j • 10 months ago
Modify the above function as

```
int noOfFactors(int n)
{
int nF=1,k=0;
while (n%2 == 0)
{
k++;
n = n/2;
}

nF*=k+1;

for (int i = 3; i <= sqrt(n); i = i+2)
{
k=0;
while (n%i == 0)
{
```

k++;

see more

1 • Reply • Share ›

Pankaj Dhakre • a year ago

#include <stdio.h>

#include <conio.h>

#include <math.h>

void printPrimeFactors(int num)

{

int prime = 2;

while(num>1)

{

if(num%prime == 0)

{

printf("%d ",prime);

see more

3 • Reply • Share ›

Pankaj Dhakre Pankaj Dhakre • a year ago

This will give the same result without using complicated sqrt().

• Reply • Share ›

Amit Pankaj Dhakre • a year ago

Well this would make more useless iterations over even numbers. And for large numbers, this becomes a problem. Squareroot is calculated in O(log n) time. So this is a good tradeoff against even numbers.

3 • Reply • Share ›

Ayush tomar Pankaj Dhakre • a year ago

This isO(log(n))?

1 • Reply • Share ›

Guest Ayush tomar • a year ago

no...it is O(sqrt(n))...

log(10000)=14 (if base is 2)

log(10000)=14 (if base is 2)...
sqrt(10000)=100
1 • Reply • Share ›

pratik • a year ago
this wont work when we take n = 3
• Reply • Share ›

Vijai pratik • a year ago
it will work... see last line if(n>2) print n...
• Reply • Share ›

Hindhu Jahnavi • a year ago
6 is a num it has 2 3 primefactors...3 is prime factor greater than sqaureroot of 6 noo....
• Reply • Share ›

Vinay Singh Hindhu Jahnavi • a year ago
Please note that 6 is even and hence divisible by 2. We would reach step 2 only when the number is odd, since we are repeatedly dividing the number by two in step 1.
• Reply • Share ›

human • a year ago
is n>2 in last step necessary?
won't n>1 also work?
• Reply • Share ›

duskoKoscica • a year ago
When it comes to prime numbers it is the problem when you handle the big ones, that way you are in place to check a lot possible ones, there are some algorithms to see if the number is prime. But for this task I would sugest to create some data structure and save primes in it. This way you could use this thing manny times. Now I got idea to even save the .... oK NOW I need to put my thinking hat on.
• Reply • Share ›

titanium • a year ago
What is Time Complexity of above algorithm ?
• Reply • Share ›

Steven titanium • a year ago
Worst Case, I think sqrt(n)*log(n).
• Reply • Share ›

titanium Steven • a year ago
Can u explain how you got that ?

• Reply • Share ›

Ayush tomar titanium • a year ago

Steven just like binary search works .it divides the range by 2 and again by 2 . and we get a complexity of O(log(n)) in this case too we are dividing by 2 again and again sqrt(n) is quite obvious

• Reply • Share ›

Himanshu chauhan • a year ago

Run above algo on # 893025... it will fail....

• Reply • Share ›

Steven Himanshu chauhan • a year ago

It worked for me... 3 3 3 3 3 3 5 5 7 7...

• Reply • Share ›

Vinay Singh • a year ago

Please note that sqrt() function is CPU intensive and since it's there within the second for loop, it will be invoked for each iteration! This would defeat the very purpose of using the sqrt() function. It would be better to first evaluate the square root just outside the loop, store it inside a variable and then use that variable in the condition of the for loop (space-time trade-off).

I recently wrote a blog post (http://www.vinaysingh.info/tim... where I did a small benchmark in Java for comparing the prime number generation and got some strange results.

• Reply • Share ›

Deepak Kumar Sahu Vinay Singh • a year ago

If "i <= sqrt(n)" is CPU intensive then how about replacing it with "(i^2) <= n"

• Reply • Share ›

Vinay Singh Deepak Kumar Sahu • a year ago

I don't think that would make a lot of difference because there is no exponentiation operator in C/C++ and the complexity of pow () would O (log n). As I mentioned using the sqrt () function just before the for statement would solve the problem. Something like:

```
limit=(int)sqrt (n);
for (int i = 3; i <= limit; i = i+2){
```

• Reply • Share ›

Person Vinay Singh • a year ago

Old, but for ^2 you can just multiply by it self in constant time. =P i*i -> O(1)

1 • Reply • Share ›

pravesh Vinay Singh • a year ago

but the value of n is changed within the loop

Reply • Share ›

Vinay Singh pravesh • a year ago

Right. I missed that. Now, n/2 would probably be better. I would do some benchmarking and then update.

Reply • Share ›

L • a year ago

Wonderful job dude.. seriously coooooolll!!!!!

Reply • Share ›

Sackri • a year ago

Instead of performing n % 2, and n / 2 in the above function, can't we simply write n & 1 and n >> 1 ?

simply,

```
while( !(n & 1) ) {
// print 2;
n >> 1;
}
```

4 • Reply • Share ›

Jerry Goyal Sackri • a month ago

it won't work for even numbers.

Reply • Share ›

Load more comments

-

GeeksforGeeks

Like You like this.

You and 100,569 others like GeeksforGeeks.

Sayak Naiya Arslan Sahir Jeeva Vanka

- Facebook social plugin

-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms

-
- ## Popular Posts
    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding “extern” keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - Himanshu Dewan

    that's because you wrote sizeof(*arr) making it...

    Output of C Program | Set 26 · 1 minute ago
  - taru sen

    #include<stdio.h> #include<conio.h> int main(){...

    Segregate 0s and 1s in an array · 3 minutes ago
  - Bunty

    # include<stdio.h> # include<stdlib.h> int...

    Implement Queue using Stacks · 4 minutes ago
  - Bunty

    SIMPLE PROGRAM FOR IMPLEMENTATION OF QUEUE...

    Implement Queue using Stacks · 10 minutes ago
  - guy

    i don't understand why we can't find the number...

    Majority Element · 19 minutes ago
  - logic

    no shit...Sherlock!

    Given an array A[] and a number x, check for pair in A[] with sum as x · 28 minutes ago

-