江西理工大学 Jiangxi University of Science and Technology

信息工程学院 School of information engineering

高级算法分析与设计

Advanced Algorithm Analysis and Design

Lecture 04:
Some example and some algorithm

Dr Ata Jahangir Moshayedi

EMAIL: ajm@jxust.edu.cn

Prof Associate ,
School of information engineering Jiangxi university of science and technology, China

Autumn _2022

高级算法分析与设计

Advanced Algorithm Analysis and Design

LECTURE 04: 8 example and 30 algorithm

8一些算法和示例

Agenda

目录

CONTENTS

8 example and 30 algorithm

Example 1:Divisibility （整除）
Example 2: Prime numbers （素数）
Example 3: Express an even number as sum of prime numbers （将偶数表示为素数之和）
Example 4: Program: Prime number below n （程序：n以下的素数）
Example 5: The greatest common divisor of two numbers （两个数的最大公约数）
Example 6: print all prime factors of a given number （打印给定数字的所有素因数）
Example 7: Coin Change Problem （硬币兑换问题）
Example 8: Fibonacci numbers （斐波那契数列）

EXAMPLE

高级算法分析与设计

Advanced Algorithm Analysis and Design

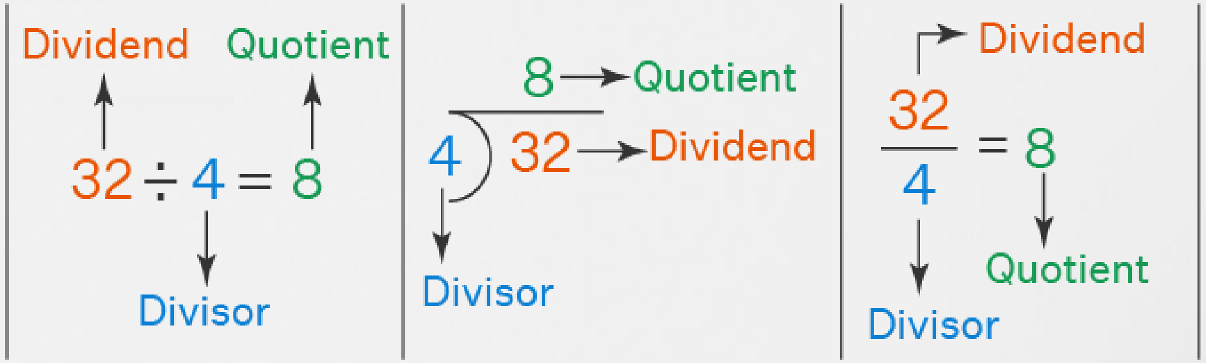
Example 1:Divisibility

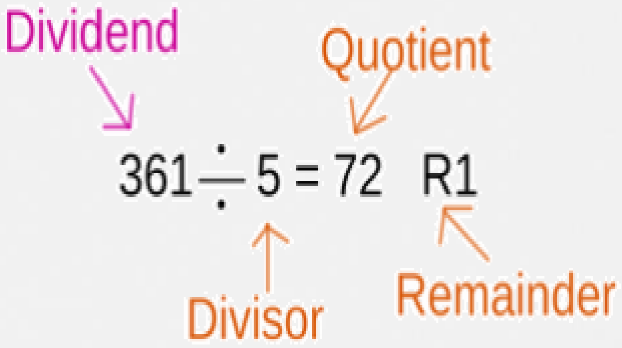
示例1：整除

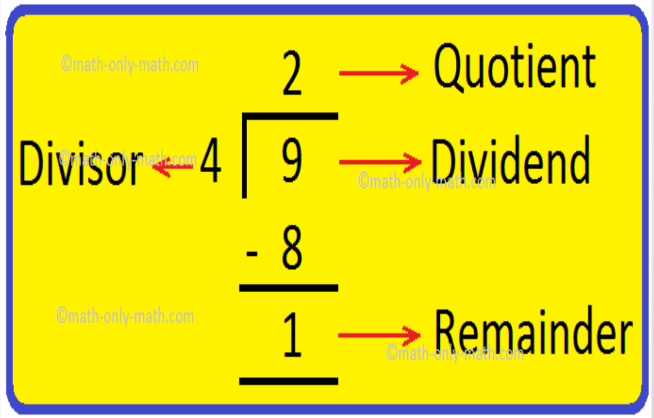
• Divisibility rule （可除性规则）

A **divisibility rule** is a shorthand and useful way of determining whether a given integer is divisible by a fixed divisor without performing the division, usually by examining its digits. Although there are divisibility tests for numbers in any radix, or base, and they are all different, this article presents rules and examples only for decimal, or base 10, numbers. Martin Gardner explained and popularized these rules in his September 1962 "Mathematical Games" column in *Scientific American*.

（可除性规则是一种简写且有用的方法，用于确定给定整数是否可被固定除数整除，而无需执行除法，通常通过检查其数字。尽管对任何基数或基数中的数字都有可除性测试，并且它们都不同，但本文仅针对十进制或以 10 为底的数字提供规则和示例。马丁·加德纳（Martin Gardner）在1962年9月的《科学美国人》(Scientific American）的“数学游戏”专栏中解释并推广了这些规则。）







Dividend = Divisor × Quotient + Remainder （被除数 =除数×商 + 余数）

江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Divisibility
整除性

dividend（被除数） divisor（除数）

30 | 1
30 30 Quotient（商）
0 Remainder（余数）

30 | 2
30 15
0

30 | 3
30 10
0

30 | 4
28 7
2

30 | 5
30 6
0

30 | 6
30 5
0

dividend（被除数） divisor（除数）

30 | 15
30 2 Quotient（商）
0 Remainder（余数）

30 | 16
16 1
14

30 | 17
17 1
13

……

……

.

30 | 29
29 1
1

30 | 30
30 1
0

Program1: divisibility—Version1 可除性——版本1


```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
int main()
{
        int n, i; // number and loop variable
        cout << "Enter a number ====> ";  // get the number from user
        cin >> n; // get the number
        for (i = 1; i <=n; i++)// form 1 to n
                if (n % i == 0)
                        cout << i << "\t"<<n/i<<endl;
        _getch();
   return 0;
}
```

C:\Users\AJM\Desktop\test\bin\Debug\test.exe

```
Enter a number ====> 30
1       30
2       15
3       10
5       6
6       5
10      3
15      2
30      1

Process returned 0 (0x0)   execution time : 4.296 s
Press any key to continue.
```

C:\Users\AJM\Desktop\test\bin\Debug\test.exe

```
Enter a number ====> 17
1       17
17      1

Process returned 0 (0x0)   execution time : 56.880 s
Press any key to continue.
```

Simple but need more time

(简单但需要更多时间)

How to make this process fast.....
怎么使过程加快...

How to make this process fast…………..

Dividend （被除数） divisor （除数）

```
30 | 1
30   30
--
0
```

Quotient （商）

Remainder （余数）

```
30 | 2
30   15
--
0
```

```
30 | 3
30   10
--
0
```

```
30 | 4
28   7
--
2
```

```
30 | 5
30   6
--
0
```

```
30 | 6
30   5
--
0
```

Dividend Divisor （除数）

（被除数）

```
30 | 15
30   2
--
0
```

（商） Quotient

Remainder （余数）

```
30 | 16
16   1
--
14
```

```
30 | 17
17   1
--
13
```

……

……

.

```
30 | 29
29   1
--
1
```

```
30 | 30
30   1
--
0
```

江西理工大学

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

How to make this process fast...............
怎么使过程加快...

- Algo 1: 算法1

You can divide from n to n/2: 1→ n/2

(你可以从 n 到n/2除：1->n/2)

30= 1→ n/2
→(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

For(n=1;n/2;n++)

Algo2: 算法2

In the number 5 the place of divisor and Quotient

(在数字 5 中，除数和商的位置)

30	5
30	6
0	

30	6
30	5
0	

Then we can do from 1→ $\sqrt{n}$

(然后我们可以从 1->$\sqrt{n}$ 做)

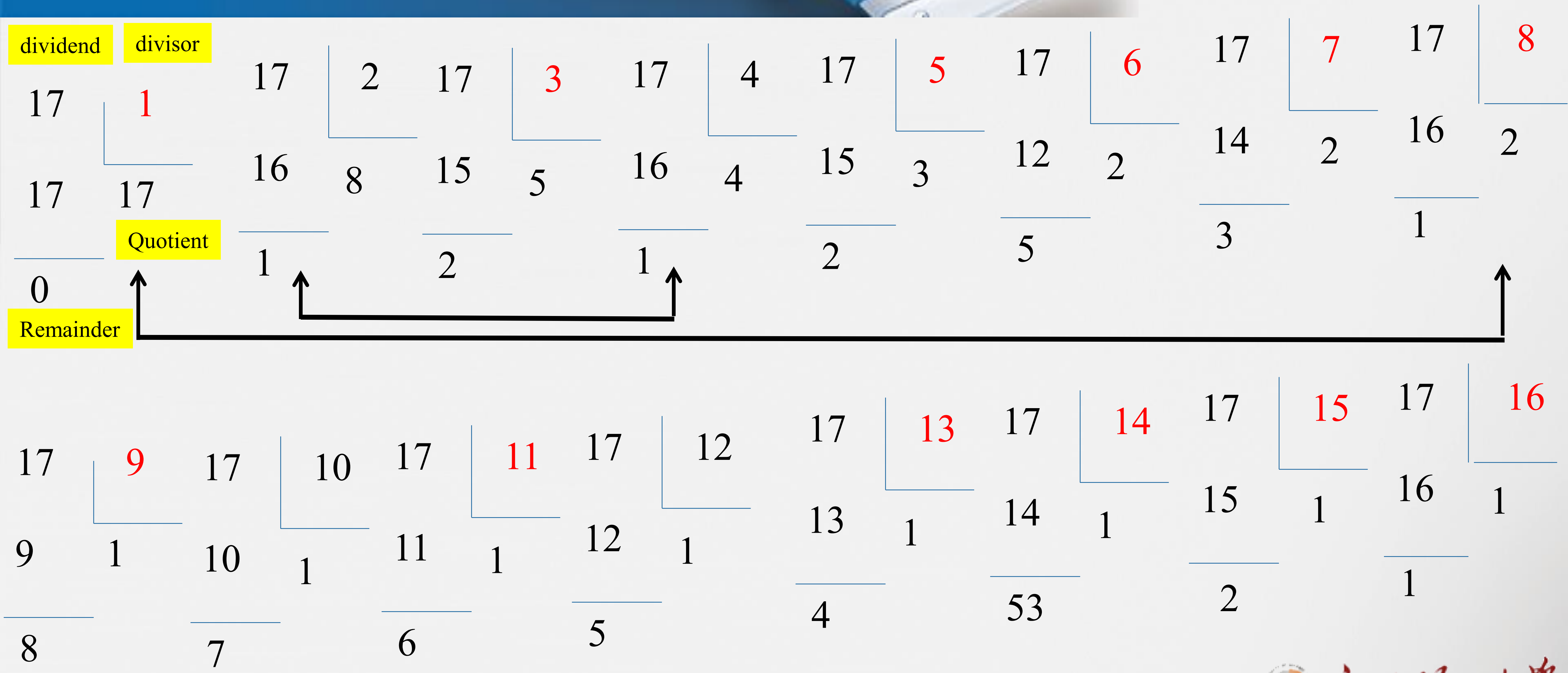
For(n=1;sqrt(n);n++)

√(30)

5.477225575051661134569697828008

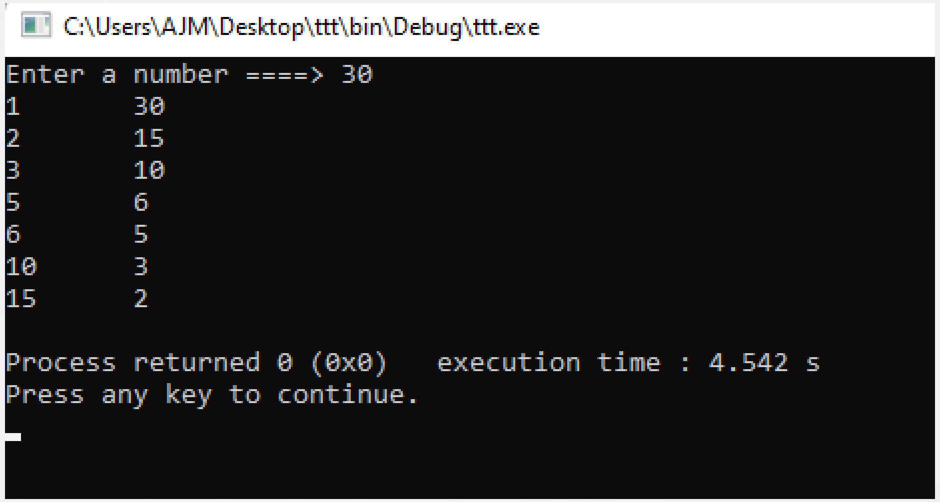
江西理工大学

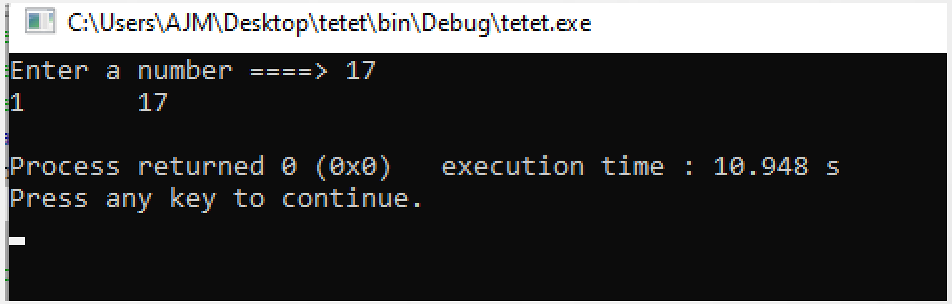
One more example to test
再举一个要测试的例子



Program1: divisibility—Version2
可除性——版本2

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
int main()
{
		int n, i; // number and loop variable
		cout << "Enter a number ====> ";  // get the number from user
		cin >> n; // get the number
		for (i = 1; i <= n/2; i++)// form sgrt
			if (n % i == 0) // shows 1 and divisor
				cout << i << "\t"<<n/i<<endl;
		_getch();
	return 0;
}
```





Program1: divisibility—Version3 可除性——版本3

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
int main()
{
        int n, i; // number and loop variable
        cout << "Enter a number ====> ";  // get the number from user
        cin >> n; // get the number
        for (i = 1; i <= sqrt(n); i++)// form sgrt
                if (n % i == 0) // shows 1 and divisor
                        cout << i << "\t"<<n/i<<endl;
        _getch();
   return 0;
}
```

C:\Users\AJM\Desktop\tes2\bin\Debug\tes2.exe

```
Enter a number ====> 30
1       30
2       15
3       10
5       6

Process returned 0 (0x0)   execution time : 6.798 s
Press any key to continue.
```

C:\Users\AJM\Desktop\tes2\bin\Debug\tes2.exe

```
Enter a number ====> 17
1       17

Process returned 0 (0x0)   execution time : 4.717 s
Press any key to continue.
```

EXAMPLE

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 2: Prime numbers

示例2：素数

江西理工大学

Program2:
Prime numbers
素数

- A prime number is a whole number greater than 1 whose **only** factors are 1 and itself. （素数是大于 1 的整数，其唯一因数是 1 和自身。）
- A factor is a whole number that can be divided evenly into another number. （因子是可以平均分成另一个数字的整数。）
- The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29. （前几个素数是 2、3、5、7、11、13、17、19、23 和 29。）
- Numbers that have more than two factors are called composite numbers. (具有两个以上因子的数字称为合数)
- The number 1 is neither prime nor composite. (数字 1 既不是素数也不是合数。）

Prime Numbers

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

Program2:
Prime numbers 素数

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
        int n, i,c=0;
        // n is the number, i for the loop and c is the counter
         cout << "Enter a number ====> ";
        cin >> n; // get the number
        for (i = 1; i <= n ; i++)
            if (n % i == 0) // if divison was done
              c++;// couter+1
        if (c == 2)
          cout << "is prime";
        else
         cout << "Not prime";
_getch();
    return 0;
}
```

```
C:\Users\AJM\Desktop\YYY\bin\Debug\YYY.exe
Enter a number ====> 22
Not prime
Process returned 0 (0x0)   execution time : 3.067 s
Press any key to continue.
```

```
C:\Users\AJM\Desktop\sdsada\bin\Debug\sdsada.exe
Enter a number ====> 117
Not prime
```

```
C:\Users\AJM\Desktop\sdsada\bin\Debug\sdsada.exe
Enter a number ====> 2562
Not prime
```

Let us save this program as a function
让我们将此程序保存为函数

```
#include <iostream>
#include <conio.h>
using namespace std;

int Prim(int n)
{
   int i, c = 0;
for (int i = 1; i <= n; i++)
if (n%i == 0)
        c++;
        if (c == 2)
            return 1;
        else
            return 0;
}

int main()
{
            int n;
            cout << "Enter a number.. ====> ";
            cin >> n;
            if ( Prim(n))
            cout << n << ":is prime";
                        else
      cout << n << ":is not prime";
            _getch();
   return 0;
}
```

```
C:\Users\AJM\Desktop\prime2\bin\Debug\prime2.exe
Enter a number.. ====> 45
45:is not prime
Process returned 0 (0x0)   execution time : 11.447 s
Press any key to continue.
```

```
C:\Users\AJM\Desktop\prime2\bin\Debug\prime2.exe
Enter a number.. ====> 1250047
1250047:is not prime
```

```
C:\Users\AJM\Desktop\prime2\bin\Debug\prime2.exe
Enter a number.. ====> 5
5:is prime
```

```
C:\Users\AJM\Desktop\prime2\bin\Debug\prime2.exe
Enter a number.. ====> 127
127:is prime
```

5

- We want to optimize this algorithm

（我们想优化这个算法）

- This algorithm is the simplest one which can detect the one as the prime number

（该算法是最简单的算法，可以将 1 检测为素数）

- But there is big problem （但是有很大的问题）
- It will do a lot division and will find the number is not prime
（它会做很多除法，并且会发现数字不是素数）
- As the example check the number 20000
（作为示例，检查数字 20000）
- It can simply find in second step that the number is not prime but will take more time to finish all divisions
- then: HOW WE CAN OPTIMIZE IT?????????

(然后：我们如何优化（它可以简单地在第二步中找到该数字不是素数，但需要更多时间才能完成所有部分）它????????? ）

Optimize the program 优化程序

```
#include <iostream>
#include <conio.h>
#include <math.h>

using namespace std;

int Prim(int n)
{
   int i, c = 0;
if (n<=1) return 0;
for (i = 2; i < sqrt(n) ; i++)
if (n%i == 0)
 return 0;
            return 1;
}

int main()
{
            int n;
            cout << "yes! Enter a number.. ====> ";
            cin >> n;
            if ( Prim(n))
            cout << n << ":is prime";
                        else
      cout << n << ":is not prime";
            _getch();
   return 0;
}
```

//reduce the number of steps and find the prime or not faster （减少步数并更快地找到素数）

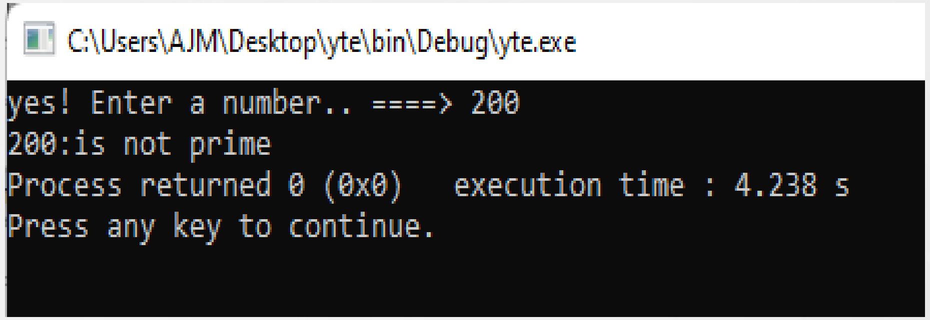
//no need to start from 1 ; as we know are number can be divide by 1 then Start the number from 2

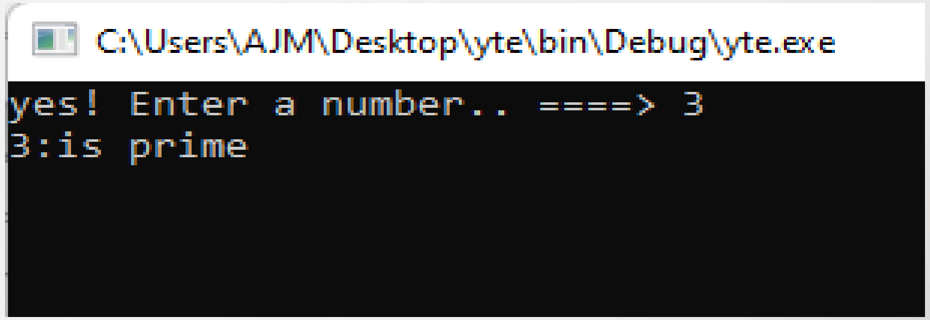
// （不需要从1开始；正如我们所知，数字可以被1除，然后从2开始）

// Devide till n >=2 （做除法直到n>=2）

Check the program and report the error

(检查程序并报告错误)





Even and Odd Numbers
偶数和奇数

Even Numbers（偶数）

- Any integer that can be divided exactly by 2 is an **even number**.
  （任何可以正好除以 2 的整数都是偶数。）
- The last digit is 0, 2, 4, 6 or 8 **（最后一个数字是 0、2、4、6 或 8）**

Odd Numbers（奇数）

- Any integer that **cannot** be divided exactly by 2 is an **odd number**.
  （任何不能完全除以 2 的整数都是奇数。）
- The last digit is 1, 3, 5, 7 or 9.

（最后一个数字是 1、3、5、7 或 9）

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

江西理工大學

Optimize the program 2
优化程序2

```
#include <iostream>
#include <conio.h>
using namespace std;

int Prim(int n)
{
        if (n <= 1) return 0;
        if (n > 2 && n % 2 == 0) return 0; // the number is not even
        for (int i = 3; i *i<= (n); i += 2)
        if (n%i == 0)
        return 0;
 return 1;
}

int main()
{
        int n;
        cout << "opt! Enter a number.. ====> ";
        cin >> n;
        if ( Prim(n))
                cout << n << ":is prime";
        else
                cout << n << ":is not prime";
        _getch();
   return 0;
}
```

```
C:\Users\AJM\Desktop\opt8\bin\Debug\opt8.exe
opt! Enter a number.. ====> 2
2:is prime
```

```
C:\Users\AJM\Desktop\opt8\bin\Debug\opt8.exe
opt! Enter a number.. ====> 2530
2530:is not prime
Process returned 0 (0x0)   execution time : 7.542 s
Press any key to continue.
```

```
C:\Users\AJM\Desktop\opt8\bin\Debug\opt8.exe
opt! Enter a number.. ====> 250
250:is not prime
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 3: Express an even number as sum of prime numbers

示例3：将偶数表示为素数之和

Express an even number as sum of prime numbers
将偶数表示为素数之和

- Given an odd number, we need to express it as the sum of at most three prime numbers.

（给定一个奇数，我们需要将其表示为最多三个素数的总和）

Input: 27
Output: 27 = 3 + 5 + 19

Input: 15
Output: 15 = 2 + 13

- **We have three cases here:** 我们有3个案例

1) When N is a prime number, print the number.
（当N是素数时，打印该数字）

2) When (N-2) is a prime number, print 2 and N-2.

（当（N-2）是素数时，打印 2 和 N-2）

3) Express N as 3 + (N-3). Obviously, N-3 will be an even number (subtraction of an odd from another odd results in even). So, according to Goldbach's conjecture, it can be expressed as the sum of two prime numbers. So, print 3 and other two prime numbers.

3）将 N 表示为 3 + （N-3）。显然，N-3 将是一个偶数（从另一个奇数中减去一个奇数得到偶数）。
因此，根据哥德巴赫猜想，它可以表示为两个素数之和。因此，打印 3 和其他两个质数。

express N as sum of at-most three prime numbers.
将 N 表示为最多三个素数的总和。

Idea 1


```cpp
using namespace std;
// Function to check if a number is prime or not.
bool isPrime(int x)
{
    if (x == 0 || x == 1)
        return false;
    for (int i = 2; i * i <= x; ++i)
        if (x % i == 0)
            return false;
    return true;
}
// Prints at most three prime numbers whose
// sum is n.
void findPrimes(int n)
{
    if (isPrime(n)) // CASE-I
        cout << n << endl;

    else if (isPrime(n - 2)) // CASE-II
        cout << 2 << " " << n - 2 << endl;

    else // CASE-III
    {
        cout << 3 << " ";
        n = n - 3;
        for (int i = 0; i < n; i++) {
            if (isPrime(i) && isPrime(n - i)) {
                cout << i << " " << (n - i);
                break;
            }
        }
    }
}
```

```cpp
// Driver code
int main()
{
    int n;
                cout << "Find me! Enter a number.. ====> ";
                cin >> n;
    findPrimes(n);
    return 0;
}
```

C:\Users\AJM\Desktop\oddtoprime\bin\Debug\oddtoprime.exe

```
Find me! Enter a number.. ====> 27
3 5 19
Process returned 0 (0x0)   execution time : 2.565 s
Press any key to continue.
```

C:\Users\AJM\Desktop\oddtoprime\bin\Debug\oddtoprime.exe

```
Find me! Enter a number.. ====> 154
3 2 149
Process returned 0 (0x0)   execution time : 2.251 s
Press any key to continue.
```

Check the program

检查程序

Time complexity（时间复杂性）：
O(n√n), (√n) to check if the number is prime and n numbers are checked.

（O（n√n），（√n）检查数字是否为素数并检查 n 个数字）

Auxiliary space（辅助空间）：O(1) as no extra space is used.

（O（1），因为没有使用额外的空间。）

江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

even number with prime number
偶数与素数

- An odd number greater than 8 can be written as the summation of non repeated prime number. （大于 8 的奇数可以写为非重复素数的总和。）
- If we remove the repeated number condition expect than 2 we can write all odd number with the summation of prime numbers

（如果我们去掉除2以外的重复数条件，我们可以用素数的总和来写所有奇数）

- Simple idea can be two loop from 1 to the number and check that the number are prime or no and then check that the summation is equal to that number or no.

（简单的想法可以是从 1 到数字的两个循环，
并检查数字是否是素数或否，
然后检查总和是否等于该数字。）

i= 1 2 3 4 5 6… n-1 n
j= 1 2 3 4 5 6… n-1 n
i+j=n

even number with prime number
偶数与素数

Simple idea

```
#include <iostream>
#include <conio.h>
using namespace std;
int Prim(int n)
{
        if (n <= 1) return 0;
        if (n > 2 && n % 2 == 0) return 0; // the number is not even
        for (int i = 3; i *i<= (n); i+= 2)
        if (n%i == 0)
        return 0;
 return 1;
}

int main()
{
        int i,j,n;
        cout << "opt! Enter a number.. ====> ";
        cin >> n;
        for (i = 1;i<=n; i++)
   for (j = 1; j<= (n); j++)

        if ( Prim(i)&&Prim(j)&& i+j==n)
        cout << endl << i<<"\t"<<j;
        _getch();
   return 0;
}
```

```
"C:\Users\AJM\Desktop\New folder\tt2\bin\Debug\tt2.exe"
opt! Enter a number.. ====> 20

3       17
7       13
13      7
17      3
```

```
"C:\Users\AJM\Desktop\New folder\tt2\bin\Debug\tt2.exe"
opt! Enter a number.. ====> 200

3       197
7       193
19      181
37      163
43      157
61      139
73      127
97      103
103     97
127     73
139     61
157     43
163     37
181     19
193     7
197     3
```

Optimization1　优化1

Optimization1

- 1.checking the number 1 is not important because 1 is not prime

  (检查数字1并不重要，因为1不是素数)

- 2.checking the number 2 is not important because 2 is just the even prime number that we have

  (检查数字2并不重要，因为2只是偶数偶数)

- Beside the summation of any number with even number is odd

  (除此之外，任何偶数的和都是奇数)

- Then we can start from 3, 5,7

  (然后我们可以从3、5、7开始)

- Start from 3
- Add n+2

```
int main()
{
    int i,j,n;
    cout << "opt! Enter a number.. ====> ";
    cin >> n;
    for (i = 3;i<=n; i+=2)
    for (j = 3; j<= n; j+=2)

    if ( Prim(i)&&Prim(j)&& i+j==n)
    cout << endl << i<<"\t"<<j;
    _getch();
    return 0;
}
```

Optimization2 优化2

```
#include <iostream>
#include <conio.h>
using namespace std;
int Prim(int n)
{
        if (n <= 1) return 0;
        if (n > 2 && n % 2 == 0) return 0; // the number is not even
        for (int i = 3; i *i<= (n); i+= 2)
        if (n%i == 0)
        return 0;
 return 1;
}

int main()
{
        int i,n;
        cout << "opt! Enter a number.. ====> ";
        cin >> n;
        for (i = 3;i<=n; i+=2)

        if ( Prim(i)&&Prim(n-i))
        cout << endl << i<<"\t"<<n-i;
        _getch();
   return 0;
}
```

Optimization 2

- We can remove the J :
  （我们可以删除J）
- For example, for number 20
  （例如，对于编号20）

We can just check 3,5,7,9
（我们可以检查3,5,7,9）

As 9 is not prime no need to check
（由于9不是素数，无需检查）

Then no need to check 20*20 =400state
（则无需检查20*20=400状态）

```
"C:\Users\AJM\Desktop\New folder\tt3\bin\Debug\tt3.exe"
opt2! Enter a number.. ====> 20

3       17
7       13
13      7
17      3
```

```
"C:\Users\AJM\Desktop\New folder\tt3\bin\Debug
opt2! Enter a number.. ====> 2000

3       1997
7       1993
13      1987
67      1933
127     1873
139     1861
199     1801
211     1789
223     1777
241     1759
277     1723
307     1693
331     1669
337     1663
373     1627
379     1621
421     1579
433     1567
457     1543
541     1459
547     1453
571     1429
577     1423
601     1399
619     1381
673     1327
709     1291
751     1249
769     1231
787     1213
829     1171
877     1123
883     1117
907     1093
937     1063
967     1033
991     1009
1009    991
1033    967
1063    937
1093    907
```

Optimization3 优化3

```
#include <iostream>
#include <conio.h>
using namespace std;
int Prim(int n)
{
        if (n <= 1) return 0;
        if (n > 2 && n % 2 == 0) return 0; // the number is not even
        for (int i = 3; i *i<= (n); i+= 2)
        if (n%i == 0)
        return 0;
 return 1;
}

int main()
{
        int i,n;
        cout << "opt! Enter a number.. ====> ";
        cin >> n;
        for (i = 3;i<n/2; i+=2)

        if ( Prim(i)&&Prim(n-i))
        cout << endl << i<<"\t"<<n-i;
        _getch();
   return 0;
}
```

Optimization3

- We can remove the J :
- For example, for number 20

We can just check 3,5,7,9

As 9 is not prime no need to check

Then no need to check 20*20 =400state

"C:\Users\AJM\Desktop\New folder\opt3\bin\Debug\opt3.exe"

```
opt2! Enter a number.. ====> 20

3       17
7       13
Process returned 0 (0x0)   execution time : 68.450 s
Press any key to continue.
```

"C:\Users\AJM\Desktop\New folder\opt3\bin\Debug\opt3.exe"

```
opt2! Enter a number.. ====> 100

3       97
11      89
17      83
29      71
41      59
47      53
```

"C:\Users\AJM\Desktop\New folder\opt3\bin\Deb... e"

```
opt2! Enter a number.. ====> 2000

3       1997
7       1993
13      1987
67      1933
127     1873
139     1861
199     1801
211     1789
223     1777
241     1759
277     1723
307     1693
331     1669
337     1663
373     1627
379     1621
421     1579
433     1567
457     1543
541     1459
547     1453
571     1429
577     1423
601     1399
619     1381
673     1327
709     1291
751     1249
769     1231
787     1213
829     1171
877     1123
883     1117
907     1093
937     1063
967     1033
991     1009
Process returned 0 (0x0)   execution time : 27.769 s
Press any key to continue.
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 4: Program: Prime number below n

示例4：程序：n以下的素数

Prime number below N / n以下的素数

- Given a number n, print all primes smaller than or equal to n.
- It is also given that n is a small number.

（给定一个数n，打印所有小于或等于n的素数。也可以假定n是一个小数。）

- **For e.g below 50??:**

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50

- **How to find the prime number below n:**

Simple idea: （如何找到n以下的素数）

we have prime function then we can have a loop and find them. （我们有素数函数，然后我们可以有一个循环并找到它们）

Next Idea:

The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 10 million or so .

（当n小于1000万左右时，Eratosthenes筛法是找到所有小于n的素数的最有效方法之一）

Prime number below N/ n以下的素数

Simple idea

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

int Prim(int n){
        if (n <= 1) return 0;
        if (n > 2 && n % 2 == 0) return 0; // the number is not even
        for (int i = 3; i *i<= (n); i += 2)
        if (n%i == 0)
return 0;
return 1;
}
int main()
{
        int n, i;
        cout << "Enter a number ====> ";
        cin >> n;
        for (i = 1; i <= n; i++)
        if ( Prim(i) )
            cout << i << endl;
        _getch();
    return 0;
}
```

"C:\Users\AJM\Desktop\New folder\pr2\bin\Debug\pr2.exe"

```
Enter a number ====> 50
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

sieve method finding the prime number below n （寻找n以下素数的筛法）

- Following is the algorithm to find all the prime numbers less than or equal to a given integer n by the Eratosthene's method: When the algorithm terminates, all the numbers in the list that are not marked are prime.

（下面是通过Eratosthene筛法查找小于或等于给定整数n的所有素数的算法：当算法终止时，列表中未标记的所有数都是素数。）

- Let us take an example when n = 50. So we need to print all prime numbers smaller than or equal to 50.

(让我们以n=50为例。所以我们需要打印所有小于或等于50的素数)

- We create a list of all numbers from 2 to 50.

(我们创建一个从2到50的所有数字的列表)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Prime number below N
n以下的素数

- According to the algorithm we will mark all the numbers which are divisible by 2 and are greater than or equal to the square of it.

（根据算法，我们将标记所有可被2整除且大于或等于其平方的数字。）

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

- Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.

（现在我们转到下一个未标记的数字3，标记所有3的倍数，大于或等于3的平方的数字。）

- We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.

（我们移动到下一个未标记的数字5，标记所有大于或等于5的倍数。）

- We continue this process, till Sqrt of n.

（我们继续这个过程，直到n的平方）

- So the prime numbers are the unmarked ones:

(所以素数是未标记的：

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Sieve Of Eratosthenes_1 （Eratosthenes筛法1）

Fast

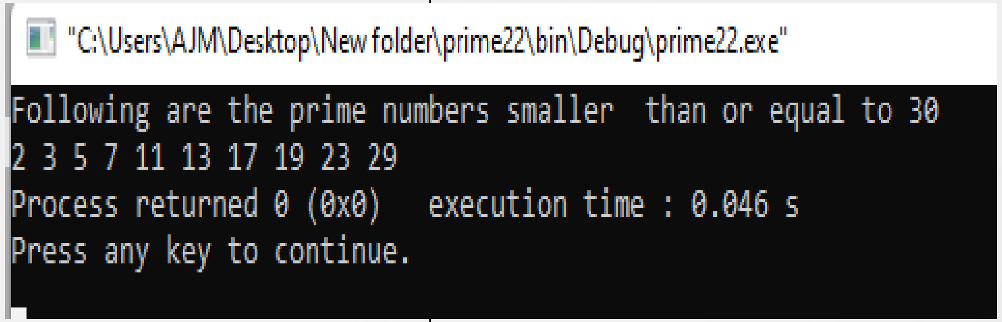
```
// n using Sieve of Eratosthenes
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int n)
{
    // Create a boolean array "prime[0..n]" and initialize
    // all entries it as true. A value in prime[i] will
    // finally be false if i is Not a prime, else true.
    bool prime[n + 1];
    memset(prime, true, sizeof(prime));

    for (int p = 2; p * p <= n; p++) {
        // If prime[p] is not changed, then it is a prime
        if (prime[p] == true) {
            // Update all multiples of p greater than or
            // equal to the square of it numbers which are
            // multiple of p and are less than p^2 are
            // already been marked.
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }
```

```
    // Print all prime numbers
    for (int p = 2; p <= n; p++)
        if (prime[p])
            cout << p << " ";
}

// Driver Code
int main()
{
    int n = 30;
    cout << "Following are the prime numbers smaller "
         << " than or equal to " << n << endl;
    SieveOfEratosthenes(n);
    return 0;
}
```



Time Complexity: *O(n*log(log(n)))*
Auxiliary Space: *O(n)*

Sieve Of Eratosthenes_2（Eratosthenes筛法2）

Faster

```
// stores only halves of odd numbers
// the algorithm is a faster by some constant factors
#include <bitset>
#include <iostream>
using namespace std;
 bitset<500001> Primes;
void SieveOfEratosthenes(int n)
{
    Primes[0] = 1;
    for (int i = 3; i*i <= n; i += 2) {
        if (Primes[i / 2] == 0) {
            for (int j = 3 * i; j <= n; j += 2 * i)
                Primes[j / 2] = 1;
        }
    }
}
```

```
"C:\Users\AJM\Desktop\New folder\pp3\bin\Debug\pp3.exe"
Enter a number ====> 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
Process returned 0 (0x0)   execution time : 2.743 s
Press any key to continue.
```

```
int main()
{
    int n;
    cout << "Enter a number ====> ";
                cin >> n;
    SieveOfEratosthenes(n);
    for (int i = 1; i <= n; i++) {
        if (i == 2)
            cout << i << ' ';
        else if (i % 2 == 1 && Primes[i / 2] == 0)
            cout << i << ' ';
    }
    return 0;
}
```

Sieve Of Eratosthenes_3 （Eratosthenes筛法3）

Faster

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
        int a[2000] = { 1,1,0 },i,j,n;
        cout << "Enter a Number ====> ";
        cin >> n;
        for (i = 1; i*i <= n; i++)
        if (a[i] == 0)
        for (j = i * i; j <= n; j += i)
        a[j] = 1;
        for (i = 1; i <= n; i++)
        if (a[i] == 0)
        cout << i << "\t";
        _getch();
        return 0;
}
```

"C:\Users\AJM\Desktop\New folder\ooop5\bin\Debug\ooop5.exe"

```
Enter a Number ====> 167
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      101     103     107     109     113
127     131     137     139     149     151     157     163     167
```

EXAMPLE

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 5: The greatest common divisor of two numbers

示例5：
两个数的最大公约数

江西理工大学

(GCD) & (HCF)

- The greatest common divisor (GCD) of two or more numbers is the greatest common factor number that divides them, exactly. It is also called the highest common factor (HCF).
（两个或多个数的最大公约数（GCD）是精确划分它们的最大公因数。它也被称为最高公因数（HCF）。）
- For example, the greatest common factor of 15 and 10 is 5, since both the numbers can be divided by 5.

（例如，15和10的最大公因数是5，因为这两个数字都可以除以5。）

  - 15/5 = 3
  - 10/5 = 2
- If a and b are two numbers then the greatest common divisor of both the numbers is denoted by gcd(a, b). To find the gcd of numbers, we need to list all the factors of the numbers and find the largest common factor.
(如果a和b是两个数，那么这两个数的最大公约数用gcd（a，b）表示。要找到数字的gcd，我们需要列出数字的所有因子，并找到最大的公共因子。)

- Suppose, 4, 8 and 16 are three numbers. Then the factors of 4, 8 and 16 are:
（假设4、8和16是三个数字。那么4、8和16的因子是）
  - 4 → 1,2,4
  - 8 → 1,2,4,8
  - 16 → 1,2,4,8,16
- Therefore, we can conclude that 4 is the highest common factor among all three numbers.
（因此，我们可以得出结论，4是所有三个数字中最高的共同因子）

江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

(GCD) & (HCF)

- A simple and old approach is **Euclidean algorithm by subtraction**

  （一种简单而古老的方法是欧几里得减法算法）

- It is is a process of repeat subtraction, carrying the result forward each time until the result is equal to the any one number being subtracted. If the answer is greater than 1, there is a GCD (besides 1). If the answer is 1, there is no common divisor (besides 1), and so both numbers are coprime.

  （这是一个重复减法的过程，每次都将结果向前推进，直到结果等于被减去的任何一个数字。如果答案大于1，则存在GCD（除1外）。如果答案是1，就没有公约数（除了1），所以这两个数都是互质的）

- pseudo code for above approach
  （上述方法的伪代码）：

```
def gcd(a, b):
 if a == b:
 return a
 if a > b:
 gcd(a – b, b)
 else:
 gcd(a, b – a)
```

- At some point one number becomes factor of the other so instead of repeatedly subtracting till both become equal , we check if it is factor of the other .

  （在某一点上，一个数成为另一个数的因子，所以我们不重复减法直到两者相等，而是检查它是否是另一个的因子）

For Example

- suppose a=98 & b=56 a>b so put a= a-b and b remains same. So a=98-56=42 & b= 56 . Since b>a, we check if b%a==0.

（假设a=98&b=56a>b，那么a=a-b和b保持不变。）
（所以a=98-56=42&b=56。由于b>a，我们检查b%a==0。）

- since answer is no we proceed further. Now b>a so b=b-a and a remains same. So b= 56-42 = 14 & a= 42 . Since a>b, we check if a%b==0 .

（既然答案是否定的，我们就继续下去。现在b>a，所以b=b-a和a保持不变。）
（所以b=56-42=14&a=42。由于a>b，我们检查a%b==0。）

- Now answer is yes .So we print smaller among a and b as H.C.F . i.e. 42 is 3 times of 14 so **HCF** is 14 .

（现在答案是肯定的。所以我们在a和b之间打印出较小的H.C.F，即42是14的3倍，所以HCF是14。）

- likewise when a=36 & b=60 ,here b>a so b = 24 & a= 36 but a%b!=0.

（同样，当a=36&b=60时，这里b>a，所以b=24&a=36，但a%b=0）

- Now a>b so a= 12 & b= 24 . and b%a==0.

（现在a>b，所以a=12和b=24。并且b%a = = 0。）

- smaller among a and b is 12 which becomes HCF of 36 and 60 .

（a和b中较小的为12，其HCF为36和60。）

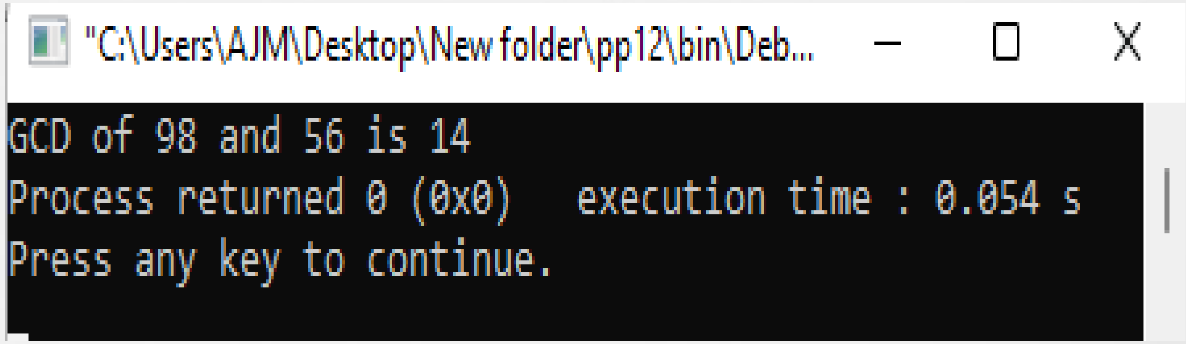
For Example

Simple Solution :

```
// C++ program to find GCD of two numbers
#include <iostream>
using namespace std;
// Function to return gcd of a and b
int gcd(int a, int b)
{
        int result = min(a, b); // Find Minimum of a nd b
        while (result > 0) {
                if (a % result == 0 && b % result == 0) {
                        break;
                }
                result--;
        }
        return result; // return gcd of a nd b
}
// Driver program to test above function
int main()
{
        int a = 98, b = 56;
        cout << "GCD of " << a << " and " << b << " is "
                << gcd(a, b);
        return 0;
}
```

- For finding GCD of two numbers we will first find the minimum of the two numbers and then find the highest common factor of that minimum which is also the factor of the other number.

（为了找到两个数的GCD，我们将首先找到这两个数中的最小值，然后找到该最小值的最高公因数，这也是另一个数的因数。）



Time Complexity : O(min(a,b))
Auxiliary Space: O(1) or constant

Efficient Solution:

```
// C++ program to find GCD of two numbers
#include <iostream>
using namespace std;
// Recursive function to return gcd of a and b
int gcd(int a, int b)
{
if (a == 0) // Everything divides 0
      return b;
   if (b == 0)
      return a;
if (a == b) // base case
       return a;
if (a > b) // a is greater
       return gcd(a-b, b);
   return gcd(a, b-a);
}
// Driver program to test above function
int main()
{
   int a = 98, b = 56;
   cout<<"GCD of "<<a<<" and "<<b<<" is "<<gcd(a, b);
   return 0;
}
```

An **efficient solution** is to use Euclidean algorithm which is the main algorithm used for this purpose.

（一个有效的解决方案是使用欧几里得算法，该算法是用于此目的的主要算法。）

The idea is, GCD of two numbers doesn’t change if smaller number is subtracted from a bigger number

. （这个想法是，如果从一个较大的数字中减去较小的数字，两个数字的GCD不会改变。）

```
"C:\Users\AJM\Desktop\New folder\pr5\bin\Debug\pr5.exe"
GCD of 98 and 56 is 14
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Time Complexity: O(min(a,b))
Auxiliary Space: O(min(a,b))

One more Solution:

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
        int a, b, r;//a,b, r as the reminder
        cout << "Enter 2 Numbers ====> ";
        cin >> a >> b;
// the loop till number become zero
        do {
                r = a%b;// reminder of  and b
                a = b;
                b = r;
        } while (r != 0);
        cout << a;
        _getch();
        return 0;
}
```

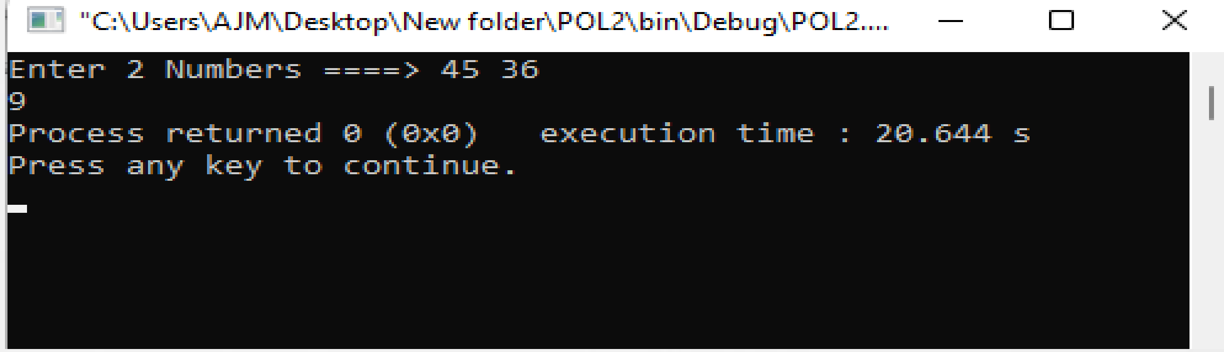
```
"C:\Users\AJM\Desktop\New folder\poee\bin\Debug\poee.exe"
Enter 2 Numbers ====> 45 36
9
Process returned 0 (0x0)   execution time : 16.590 s
Press any key to continue.
```

ConsoleApplication6.cpp

Return Solution:

```
#include "stdafx.h"
#include<iostream>
#include<conio.h>
using namespace std;

int BMM(int a, int b)
{
        if (a%b != 0)
        BMM(b, a%b);
        else
           return b;
}
int main()
{
        int a, b, r;
        cout << "Enter 2 Numbers ====> ";
        cin >> a >> b;
        cout << BMM(a, b);
        _getch();
        return 0;
}
```



EXAMPLE

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 6: print all prime factors of a given number

示例 6：打印给定数字的所有素因数

print all prime factors of a given number
打印给定数字的所有素因数

- Given a number **n**, write an efficient function to print all prime factors of **n**.

(给定一个数字n，编写一个有效的函数来打印n的所有素因子。)

12	2
6	2
3	3

For example, if the input number is 12, then the output should be “2 2 3”.

（例如，如果输入数字是12，那么输出应该是“2 2 3”。）

315	3
105	3
35	5
7	7

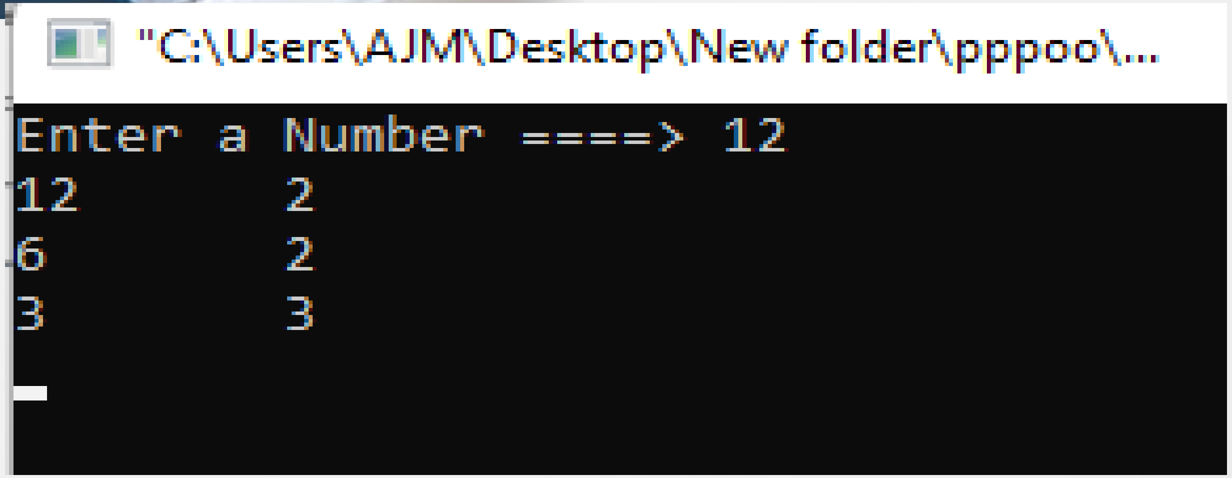
And if the input number is 315, then the output should be “3 3 5 7”.

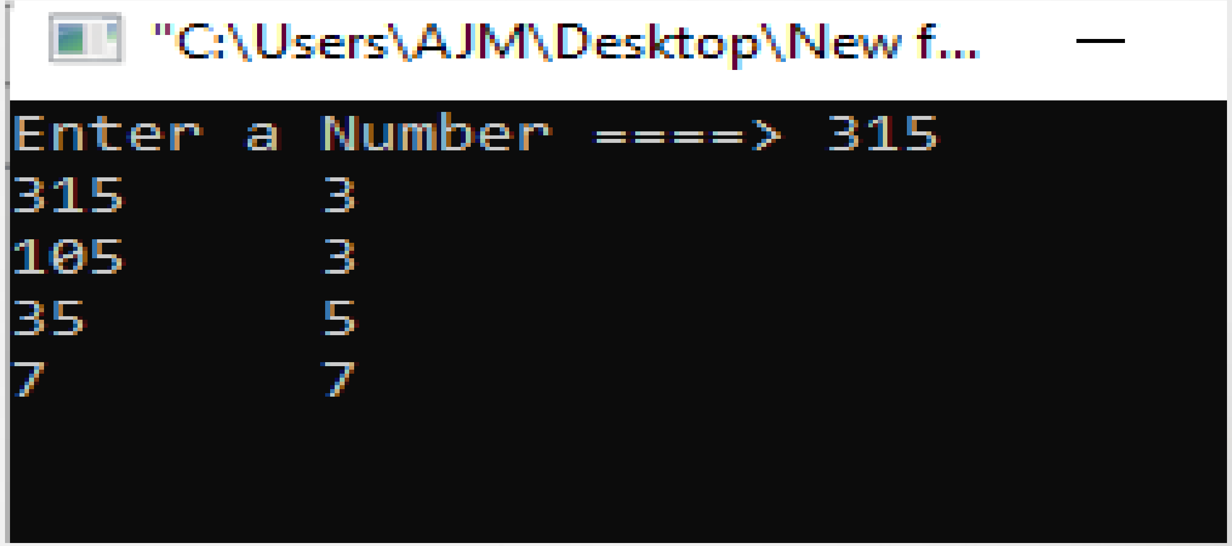
（如果输入数字为315，则输出应为“3 3 5 7”。）

print all prime factors of a given number
打印给定数字的所有素因数

```
#include "stdafx.h"
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
            int a, b=2, r;
            cout << "Enter a Number ====> ";
            cin >> a ;
            while (a > 1)
            {
if (a%b == 0)
    {
    cout << a << "\t" << b << endl;
    a = a / b;
}
else
    b++;
            }
  _getch();
   return 0;
}
```





ConsoleApplication8.cpp

Return Solution:

```
#include "stdafx.h"
#include<iostream>
#include<conio.h>
using namespace std;
void Tajzeh(int a, int b)
{
 if (a > 1)
  if (a%b == 0)
                  {
                   cout << a << "\t" << b << endl;
                   Tajzeh( a / b,b);
                  }
 else
 Tajzeh(a,b+1);
}
int main()
{
                 int a;
                 cout << "Enter a Number ====> ";
                 cin >> a;
                 Tajzeh(a, 2);
                 _getch();
    return 0;
}
```

```
"C:\Users\AJM\Desktop\New folder\prog1\bin\Debug\prog1.exe"
Enter a Number ====> 288
288     2
144     2
72      2
36      2
18      2
9       3
3       3

Process returned 0 (0x0)   execution time : 5.579 s
Press any key to continue.
```

EXAMPLE

高级算法分析与设计

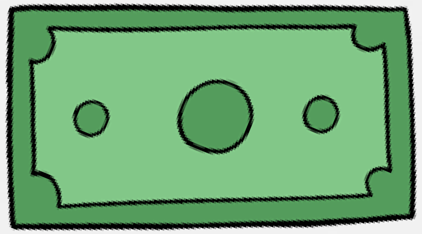
Advanced Algorithm Analysis and Design

Example 7: Coin Change Problem

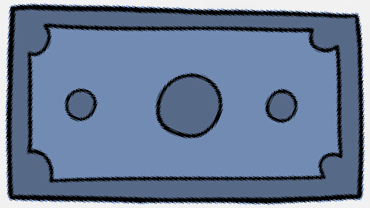
示例7：硬币兑换问题

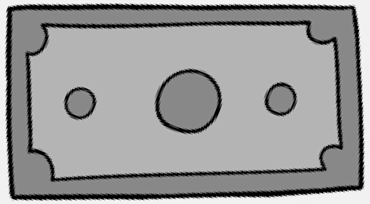
江西理工大学

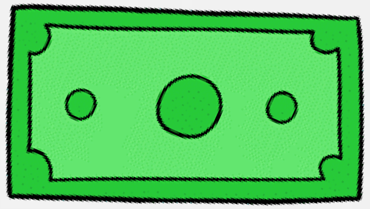
Example 7: Coin Change Problem
示例7：硬币兑换问题

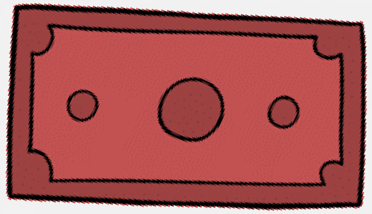


50000

A  10000 → 5*10000

B  5000 → 10*5000

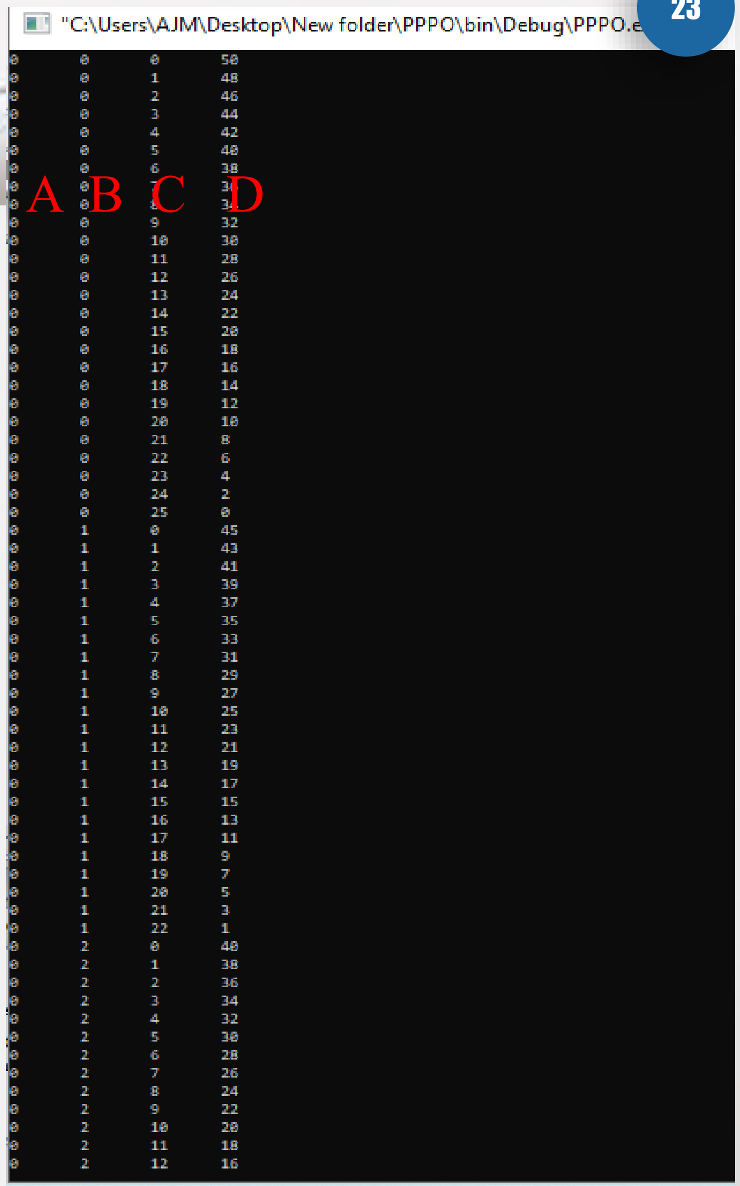
C  2000 → 25*2000

D  1000 → 50*1000

If((A*10000)+(B*5000)+(C*2000)+(D*1000))=50000

Example 7: Coin Change Problem
示例7：硬币兑换问题

```cpp
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
 int a, b, c, d;
 for (a = 0; a <= 5; a++)
 for (b = 0; b <= 10; b++)
 for (c = 0; c <= 25; c++)
  for (d = 0; d <= 50; d++)
    {
   if （ a*10 + b*10 +c*25 +d*50 == 50）
  cout <<"A:"<< a << "\t"<<"B:" << b << "\t" <<"C:"<< c << "\t" << "D:"<<d << endl;
                  }
                 _getch();
   return 0;
}
```



Example 7: Coin Change Problem
示例7：硬币兑换问题

One more Solution:

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
 int a, b, c, d;
 for (a = 0; a <= 5; a++)
 for (b = 0; b <= 10-a*2; b++)
 for (c = 0; c <= 25-a*5-b*2.5; c++) {
 d = 50 - a * 10 - b * 5 - c * 2;
  cout <<"A:"<< a << "\t"<<"B:" << b << "\t" <<"C:"<< c << "\t" << "D:"<<d << endl;
                    }
                   _getch();
    return 0;
}
```

"C:\Users\AJM\Desktop\New folder\wewe\bin\Debug\wewe.exe"

```
A:0	B:0	C:0	D:50
A:0	B:0	C:1	D:48
A:0	B:0	C:2	D:46
A:0	B:0	C:3	D:44
A:0	B:0	C:4	D:42
A:0	B:0	C:5	D:40
A:0	B:0	C:6	D:38
A:0	B:0	C:7	D:36
A:0	B:0	C:8	D:34
A:0	B:0	C:9	D:32
A:0	B:0	C:10	D:30
A:0	B:0	C:11	D:28
A:0	B:0	C:12	D:26
A:0	B:0	C:13	D:24
A:0	B:0	C:14	D:22
A:0	B:0	C:15	D:20
A:0	B:0	C:16	D:18
A:0	B:0	C:17	D:16
A:0	B:0	C:18	D:14
A:0	B:0	C:19	D:12
A:0	B:0	C:20	D:10
A:0	B:0	C:21	D:8
A:0	B:0	C:22	D:6
A:0	B:0	C:23	D:4
A:0	B:0	C:24	D:2
A:0	B:0	C:25	D:0
A:0	B:1	C:0	D:45
A:0	B:1	C:1	D:43
A:0	B:1	C:2	D:41
A:0	B:1	C:3	D:39
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 8: Fibonacci numbers

示例8：斐波那契数列

Example 8: Fibonacci numbers
示例8：斐波那契数列

- The Fibonacci numbers are the numbers in the following integer sequence.

（斐波那契数是以下整数序列中的数字。）

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

- In mathematical terms, the sequence **Fn** of Fibonacci numbers is defined by the recurrence relation

（在数学术语中，斐波那契数列Fn由递推关系定义）

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2} \quad n>2$$

- Given a number n, print n-th Fibonacci Number.

（给定一个数字n，打印第n个斐波那契数）

- Input : n = 2 Output : 1
- Input : n = 9 Output : 34

1	1	2	3	5
8	13	21	34	55
89	144	233	377	610
987	1597	2584	4181	6765

a	b	C=a+b
1	1	2
1	2	3
2	3	5
3	5	8
3	8	

江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

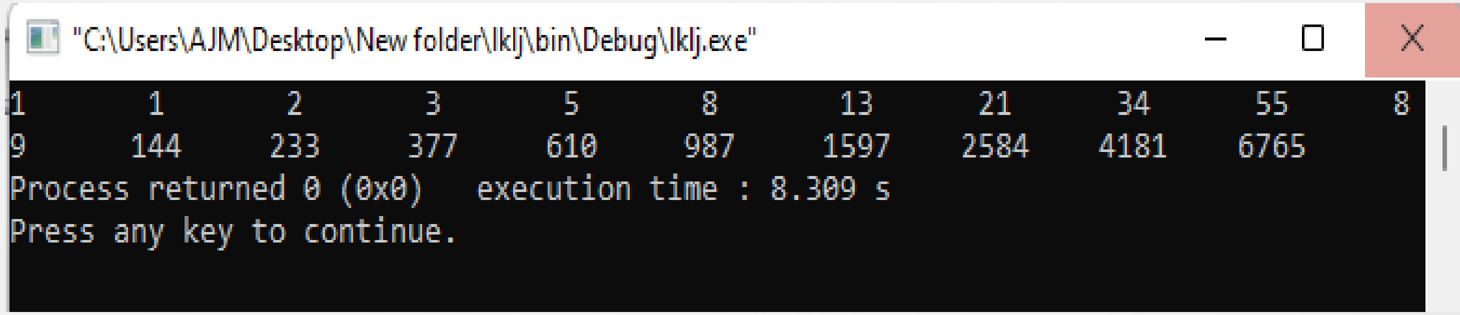
Example 8: Fibonacci numbers
示例8：斐波那契数列

Simple Solution:

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
		int a, b, c, i;
		a = b = 1;
		cout << a << "\t" << b << "\t";
		for (i = 2; i <20; i++)
		{
			c = a + b;
			cout << c << "\t";
			a = b;
			b = c;
		}
		_getch();
	return 0;
}
```



```
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
        int f[40] = {1,1},i;
        for (i = 2; i < 40; i++)
                f[i] = f[i - 1] + f[i - 2];
        for (i = 0; i <40; i++)
                cout << endl << f[i];
        _getch();
        return 0;
}
```

On more way

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
        int f[40] = {1,1},i;
        for (i = 2; i < 40; i++)
                f[i] = f[i - 1] + f[i - 2];
        for (i = 39; i >=0; i--)
                cout << endl << f[i];
        _getch();
        return 0;
}
```

Back to first But need more memory

Example 8: Fibonacci numbers using Recursion 使用递归的斐波那契数

On more way

```
#include "stdafx.h"
#include<iostream>
#include<conio.h>
using namespace std;
int Fiboo(int n)
{
        if (n <= 2)
        return 1;
        else
        return Fiboo(n - 1) + Fiboo(n - 2);
}

int main()
{
        cout << Fiboo(45);
        _getch();
    return 0;
}
```

- As long as the number become big the time will increase
  (只要数量变大，时间就会增加)
- 2^(45) state

Fibonacci Series using Recursion 使用递归的斐波那契级数

On more Solution:

- A simple method that is a direct recursive implementation mathematical recurrence relation is given above.

（上面给出了一种直接递归实现数学递归关系的简单方法。）

```
#include<iostream>
#include<conio.h>
using namespace std;
int fib(int n)
{
        if (n <= 1)
        return n;
        return fib(n - 1) + fib(n - 2);
}

int main()
{
        int n;
        cout << "Enter a Number ====> ";
        cin >> n;
        cout << fib(n);
        _getch();
        return 0;
}
```

"C:\Users\AJM\Desktop\New folder\aatar\bin\Debug\aatar.exe"

```
Enter a Number ====> 9
34
Process returned 0 (0x0)   execution time : 5.190 s
Press any key to continue.
```

Time Complexity: 时间复杂性

- **Exponential,** as every function calls two other functions.

  （指数级，因为每个函数都调用另外两个函数。）

- If the original recursion tree were to be implemented, then this would have been the tree but now for n times the recursion function is called

  （如果要实现最初的递归树，那么这将是树，但现在在递归函数被调用了n次）

- Original tree for recursion

  （递归的原始树）

```
                      fib(5)
                  /             \
            fib(4)                fib(3)
          /        \            /       \
     fib(3)      fib(2)      fib(2)    fib(1)
    /     \      /    \      /    \
 fib(2)  fib(1) fib(1) fib(0) fib(1) fib(0)
 /    \
fib(1) fib(0)
```

Optimized tree for recursion for code above （上面代码的递归优化树）

fib(5)
fib(4)
fib(3)
fib(2)
fib(1)

Extra Space: O(n) if we consider the function call stack size, otherwise O(1).

额外空间：如果我们考虑函数调用堆栈大小，则为O（n），否则为O（1）。

Method 2:
(Use Dynamic Programming)
// C++ program for Fibonacci Series
// using Dynamic Programming
方法2：（使用动态编程）//斐波那契级数的C++程序//使用动态编程

```
#include<iostream>
#include<conio.h>
using namespace std;
class GFG{
    public:
int fib(int n)
{
   // Declare an array to store
   // Fibonacci numbers.
   // 1 extra to handle
   // case, n = 0
   int f[n + 2];
   int i;

   // 0th and 1st number of the
   // series are 0 and 1
   f[0] = 0;
   f[1] = 1;

   for(i = 2; i <= n; i++)
   {
     //Add the previous 2 numbers
     // in the series and store it
     f[i] = f[i - 1] + f[i - 2];
   }
   return f[n];
   }
};
```

```
// Driver code
int main ()
{
   GFG g;
   int n;
               cout << "Enter a Number ====> ";
               cin >> n;
   cout << g.fib(n);
   return 0;
}
```

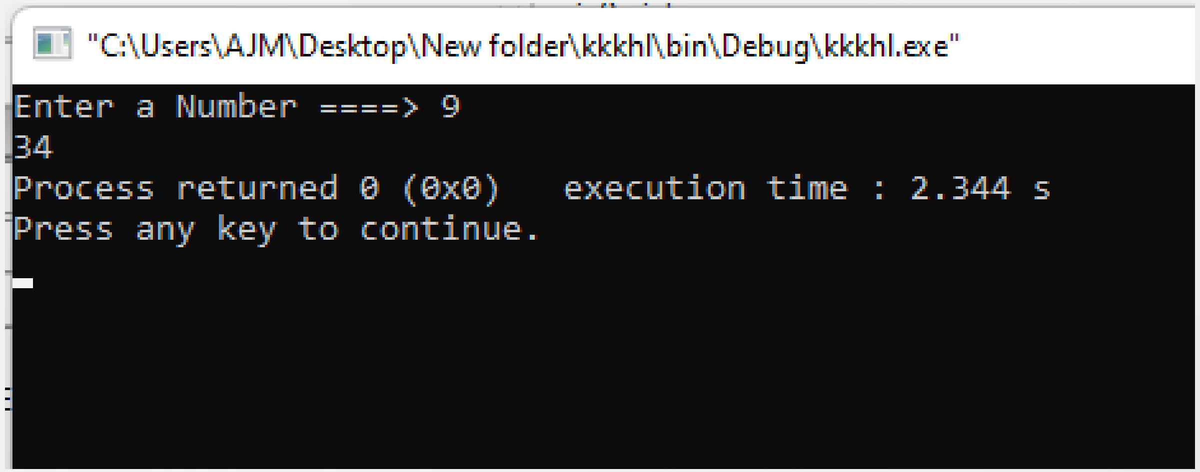
"C:\Users\AJM\Desktop\New folder\KJLJ\bin\Debug\KJLJ.exe"

```
Enter a Number ====> 9
34
Process returned 0 (0x0)   execution time : 3.147 s
Press any key to continue.
```

Method 3: (Space Optimized Method 2)
方法3： （空间优化方法2）

- We can optimize the space used in method 2 by storing the previous two numbers only because that is all we need to get the next Fibonacci number in series.

（我们可以通过存储前两个数字来优化方法2中使用的空间，因为这就是我们需要得到下一个斐波那契数列的全部内容。）




```
#include <iostream>
#include<bits/stdc++.h>
using namespace std;
int fib(int n)
{
        int a = 0, b = 1, c, i;
        if( n == 0)
                return a;
        for(i = 2; i <= n; i++)
        {
        c = a + b;
        a = b;
        b = c;
        }
        return b;
}
```

```
// Driver code
int main()
{
        int n;
        cout << "Enter a Number ====> ";
        cin >> n

        cout << fib(n);
        return 0;
}
```

Reference
参考

- https://afteracademy.com/blog/time-and-space-complexity-analysis-of-algorithm

江西理工大学

Jiangxi University of Science and Technology

信息工程学院

School of information engineering

高级算法分析与设计

Advanced Algorithm Analysis and Design

THANK YOU

“The beauty of research is that you never know where it’s going to lead.”

RICHARD ROBERTS
Nobel Prize in Physiology or Medicine 1993

“BE HUMBLE. BE HUNGRY.

AND ALWAYS BE THE

HARDEST WORKER

IN THE ROOM.”