Problem 1

Uva686 - Goldbach's Conjecture (II)

（哥德巴赫猜想）

Time: 3 seconds

Problem Descriptions (1/2)

- **For any even number n greater than or equal to 4, there exists at least one pair of prime numbers p_1 and p_2 such that $n = p_1 + p_2$.**
- **This conjecture has not been proved nor refused yet. No one is sure whether this conjecture actually holds. However, one can find such a pair of prime numbers, if any, for a given even number.**

Problem Descriptions (2/2)

- The problem here is to write a program that reports the number of all the pairs of prime numbers satisfying the condition in the conjecture for a given even number.
- A sequence of even numbers is given as input. Corresponding to each number, the program should output the number of pairs mentioned above.
- Notice that we are interested in the number of essentially different pairs and therefore you should not count (p_1, p_2) and (p_2, p_1) separately as two different pairs.

I/O

- **Input**
  - **An integer is given in each input line. You may assume that each integer is even, and is greater than or equal to 4 and less than 2^{15}. The end of the input is indicated by a number 0.**
- **Output**
  - **Each output line should contain an integer number. No other characters should appear in the output.**

Example

- **Input**

  6

  10

  12

  0

- **Output**

  1

  2

  1

6=3+3 (1case)

10=3+7, 10=5+5 (2 cases)

12=5+7 (1 cases)

Brute Force
Time Complexity $O(n^2)$

- Given a even number *n*

```
main ( )
{    for (i=2; i<=n/2 ;i++)                    O(n)
     {
          ret=check_prime(i)+check_prime(n-i)
          if (ret==2) count++;
     }}

boolean check_prime(k)                         O(n)
{     for (i=2; i<k ;i++)
               check whether (k mod i == 0) return t
               else return false
 }
```

Brute Force
Time Complexity *O(nlogn)*

- **Given a even number *n***

```
main ( )
{    for (i=2; i<=n/2 ;i++)                O(n)
     {
          ret=check_prime(i)+check_prime(n-i)
          if (ret==2) count++;
     }}

boolean check_prime(k)                     O(logn)
{     for (i=2; i<sqrt(k) ;i++)
            check whether (k mod i == 0) return t
            else return false
 }
```

Time Complexity

- **n cases**
  - $O(n^2) \times \underbrace{O(n)}_{}$ *n cases*
  - **Total:** $O(n^3)$

- **n cases**
  - $O(n\log n) \times O(n)$ *n cases*
  - **Total:** $O(n^2 \log n)$

Prime Generation: Sieve of Eratosthenes(1/4)

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

$2^{15}=65536$

Prime Generation: Sieve of Eratosthenes(2/4)

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

$2^{15}=65536$

Prime Generation: Sieve of Eratosthenes(3/4)

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

$2^{15}=65536$

Prime Generation: Sieve of Eratosthenes(4/4)

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

$2^{15}=65536$

Prime Generation: Sieve of Eratosthenes

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

$2^{15}=65536$

0	1	2	3	4	5	6	7	8	9
2	3	5	7	11	13	17	19	23	29

```
bool prime[2^15];

void sieve_eratosthenes()
{
    for (int i=0; i<2^15; i++)
        prime[i] = true;

    prime[0] = false; prime[1] = false;

    for (int i=2; i<2^15; i++)
        if (prime[i])
            for (int j=i+i; j<2^15; j +=i)
                prime[j] = false;
```

Generate a Prime Number List

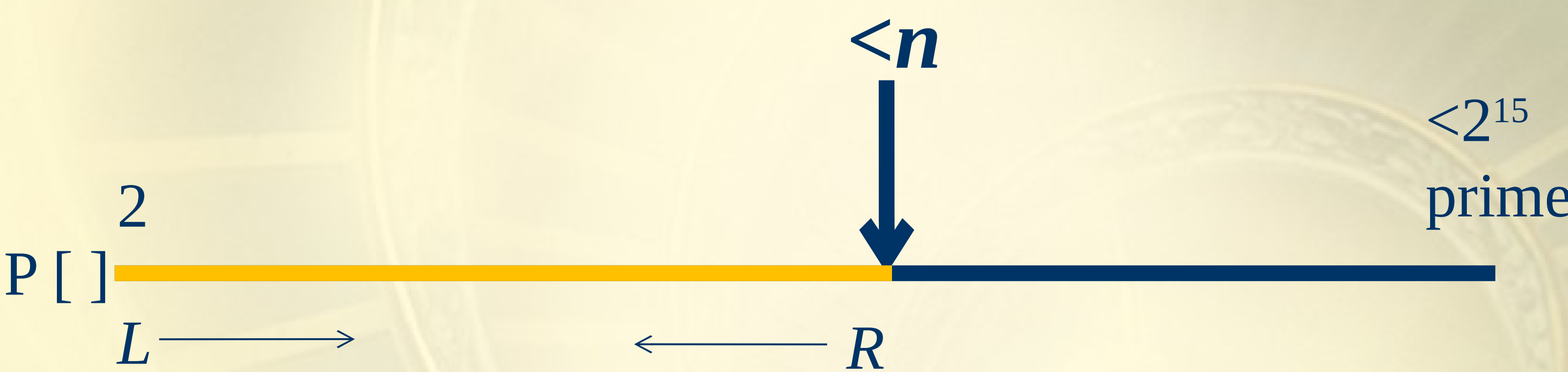
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11
~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21
~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~	31
~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~	41

$2^{15}=65536$

```
bool prime[2^15];

void sieve_eratosthenes()
{
    for (int i=0; i<2^15; i++)
        prime[i] = true;

    prime[0] = false; prime[1] = false;

    for (int i=2; i<2^15; i++)
        if (prime[i])
            for (int j=i*i; j<2^15; j+=i)
                prime[j] = false;
```

Binary Search the Bound

Given *n*




```
If (P[L]+P[R]==n)
    count++;
```

Example

Given *30*

<30

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**

L

R

30-29=1

Example

Given *30*

<30

0	1	2	3	4	5	6	7	8	9
2	3	5	7	11	13	17	19	23	29

L

R

30-23=7

Example

Given *30*

<30

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**

L

R

30-19=11

Example

Given *30*

<30

0	1	2	3	4	5	6	7	8	9
2	3	5	7	11	13	17	19	23	29

L *R*

30-17=13

Example

Given *30*

<*30*

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**

R *L*

Time Complexity

- **Prime Number List:**
  - $O(n^2)$
- **Search for the bound**
  - $O(\log n)$
- **Find the answer**
  - $O(n)$
- **Total:** $O(n^2)+\{O(\log n)+O(n)\}\times\underbrace{O(n)}_{n\ cases}$
  - $O(n^2)$

C++ Library: vector<T>

- vector 型別是以容器（Container） 模式為基準設計的，也就是說，基本上它有 begin()，end()，size()，max_size()，empty() 以及 swap() 這幾個方法。

- 存取元素的方法
  ① vec[i] - 存取索引值為 i 的元素參照。（索引值從零起算，故第一個元素是 vec[0]。）
  ② vec.at(i) - 存取索引值為 i 的元素的參照，以 at() 存取會做陣列邊界檢查，如果存取越界將會拋出一個例外，這是與 operator[] 的唯一差異。
  ③ vec.front() - 回傳 vector 第一個元素的參照。
  ④ vec.back() - 回傳 vector 最尾元素的參照。
- 新增或移除元素的方法
  ① vec.push_back() - 新增元素至 vector 的尾端，必要時會進行記憶體配置。
  ② vec.pop_back() - 刪除 vector 最尾端的元素。
  ③ vec.insert() - 插入一個或多個元素至 vector 內的任意位置。
  ④ vec.erase() - 刪除 vector 中一個或多個元素。
  ⑤ vec.clear() - 清空所有元素。
- 取得長度 / 容量
  ① vec.size() - 取得 vector 目前持有的元素個數。
  ② vec.empty() - 如果 vector 內部為空，則傳回 true 值。
  ③ vec.capacity() - 取得 vector 目前可容納的最大元素個數。這個方法與記憶體的配置有關，它通常只會增加，不會因為元素被刪減而隨之減少。
- 重新配置／重設長度
  ① vec.reserve() - 如有必要，可改變 vector 的容量大小（配置更多的記憶體）。在眾多的 STL 實做，容量只能增加，不可以減少。
  ② vec.resize() - 改變 vector 目前持有的元素個數。
- (Iterator)
  ① vec.begin() - 回傳一個 Iterator，它指向 vector 第一個元素。
  ② vec.end() - 回傳一個 Iterator，它指向 vector 最尾端元素的下一個位置（請注意：它不是最末元素）。
  ③ vec.rbegin() - 回傳一個反向 Iterator，它指向 vector 最尾端元素的。
  ④ vec.rend() - 回傳一個 Iterator，它指向 vector 的第一個元素。

Java Class: BigInteger

- 函數解析字串 "16,263,054,952,801,281,548"
  而這個數字已經遠遠超過 Long 的最大值 "9,223,372,036,854,775,807"

- 傳遞 Value 到大整數中，使用 String
  ① BigInteger(String val) : Translates the decimal String representation of a BigInteger into a BigInteger.
  ② BigInteger(String val, int radix) : Translates the String representation of a BigInteger in the specified radix into a BigInteger.

- 傳遞 Value 到大整數中，使用 String

```
String bigIntStr = "16263054952801281548";
BigInteger a = new BigInteger(bigIntStr);
System.out.printf("%s > %d\n", a, Long.MAX_VALUE);
System.out.printf("'%s' binary = %s\n", a, a.toString(2));
```

- 接著如果你要對 " 大 " 數字進行加減乘除，不能使用直覺的 "+-*/", 而必須透過 BigInteger 類別上面的方法：

```
BigInteger btwo = new BigInteger("2");
System.out.printf("%s+1=%s\n", a, a.add(BigInteger.ONE));
System.out.printf("%s-1=%s\n", a, a.subtract(BigInteger.ONE));
System.out.printf("%s*2=%s\n", a, a.multiply(btwo));
System.out.printf("%s/2=%s\n", a, a.divide(btwo));
```

```
16263054952801281548+1=16263054952801281549
16263054952801281548-1=16263054952801281547
16263054952801281548*2=32526109905602563096
16263054952801281548/2=8131527476400640774
```

- 而常用的 pow() 函數也可以使用 BigInteger 完成：

```
System.out.printf("2^100=%s\n", btwo.pow(100));
```

```
2^100=1267650600228229401496703205376
```