数论

- 数论，是专门研究整数的纯数学的分支，而整数的基本元素是素数（也称质数），所以数论的本质是对素数性质的研究。
- 数论被高斯誉为“数学中的皇冠”。按研究方法来看，数论大致可分为初等数论和高等数论。而初等数论是用初等方法研究的数论，它的研究方法本质上说，就是利用整除性质，主要包括整除理论、同余理论、连分数理论。
- 信息学竞赛中的数论主要涉及素数、约数、同余和数论函数等相关知识。

求下数组中第2大的数。

3	1	2	5	4	7	6

temp=nums[low]=3,以3为枢轴点，进行快排。

第一次，low=0,high=6,temp=3,经过快排后的结果：

6	7	4	5	3	2	1

此时partition排序后获得的index=4，4>2，那么第2大的数位于枢轴点的左边。

第二次，只针对上图中左半边进行排序。

temp=nums[low]=6,以6为枢轴点进行第2次快排。

6	7	4	5	3	2	1

第2次，low=0，high=4-1=3，temp=6；

排序后的结果：

7	6	4	5	3	2	1

此时快排后的index=1,1=2-1，所以枢轴点对应的就是第k大的数。

结束！

1 基本概念

整数集合： $Z=\{\ldots,-2,-1,0,1,2,\ldots\}$

自然数集合： $N=\{0,1,2,\ldots\}$

整除： 若 a=bk，其中a，b，k都是整数，则b整除a，记做 b|a。

约数： 若b|a且b>=0，则称b是a的约数（因数），a是b的倍数。

- 1整除任何数，任何数都整除0。
- 若a|b，a|c，则a|(b+c)，a|(b-c)。
- 若a|b，则对任意整数c，a|(bc)。

传递性： 若a|b,b|c，则a|c。

1 素数与合数

素数： a>1 且除了1和本身外不能被其它数整除的数。

合数： a>1 且不是素数的数称为合数。

其他整数（0,1负整数）既不是素数也不是合数。

素数有无穷多个，但分布比较稀疏，不大于n的素数约有n/ln(n)个。

2 素数判定（试除法）

- 若 n 是一个合数，则 n 至少有 1 个素因子。因此其中最小的素因子一定不大于 $\sqrt{n}$ 。
- 如果 n 是合数，则一定可以为分解 $a \times b$ 的形式，其中 $a \leq b, a \neq 1, b \neq n$，如 $18 = 2 \times 9, 18 = 3 \times 6$。因 $a \times a \leq a \times b = n$ ，则可得：$a \leq \sqrt{n}$ 。
- 可得判断依据：如果 $2 \sim \sqrt{n}$ 中有 n 的约数，则 n 是合数，否则 n 是素数。

2.1 素数判定（试除法）

- 若 n 是一个合数，则 n 至少有 1 个素因子。因此其中最小的素因子一定不大于 $\sqrt{n}$ 。
- 如果 n 是合数，则一定可以为分解 $a \times b$ 的形式，其中 $a \le b, a \ne 1, b \ne n$，如 $18 = 2 \times 9, 18 = 3 \times 6$。因 $a \times a \le a \times b = n$ ，则可得：$a \le \sqrt{n}$ 。
- 可得判断依据：如果 $2 \sim \sqrt{n}$ 中有 n 的约数，则 n 是合数，否则 n 是素数。

```
1       bool isPrime(int n){
2               if (n==1) return false;
3               else{
4                       for(int i=2;i*i<=n;i++)
5                               if (n % i == 0) return false;
6                       return true;
7               }
8       }
```

2.2 朴素筛选法

对于序列中的每个合数，一定可以写成$p*x$的形式，p是序列左边较小的数，x是倍数，枚举倍数x, 把$p*x$标记为合数，这就是筛选法。

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2.2 朴素筛选法

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

2.2 朴素筛选法

在筛选过程中，第一轮执行n/2次，第二轮执行n/3次，第三轮执行n/4次....

时间复杂度为$O(\frac{n}{2}+\frac{n}{3}+\frac{n}{4}+....)=O(n*(\frac{1}{2}+\frac{1}{3}+...))$

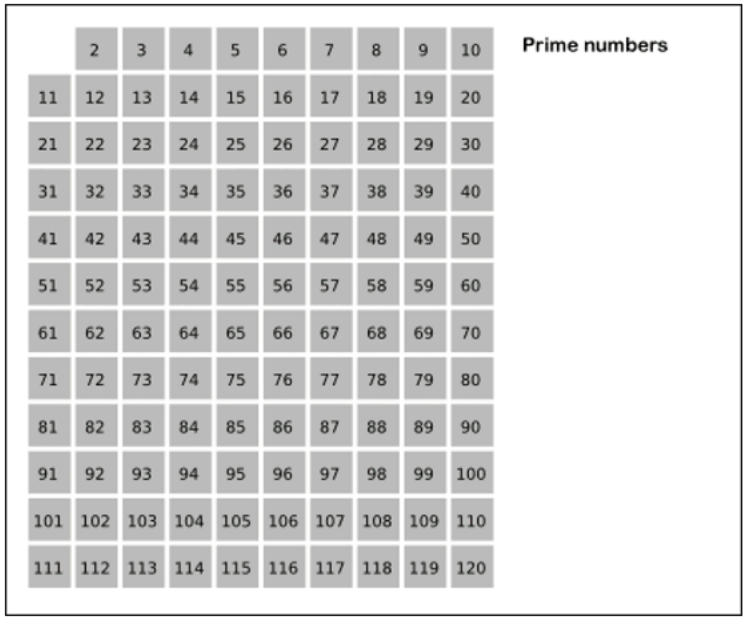
$=O(n*logn)$

每个i都需要遍历：

```
2  //朴素的筛选法
3  void get_primes2(){
4      for(int i=2;i<=n;i++){
5          if(!st[i]) primes[cnt++]=i;//把素数存起来
6          for(int j=i;j<=n;j+=i){
7          //不管是合数还是质数，都用来筛掉后面它的倍数
8              st[j]=true;
9          }
10     }
11 }
```

2.3 埃氏(Eratosthenes)筛法

在朴素筛选法中，对每个数都进行筛选，显然存在大量重复的筛选，如筛选过2后，2的倍数就不需要再筛选。这就是埃氏筛选法。



	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
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

2.3 埃氏(Eratosthenes)筛法

埃氏筛选法只筛选素数的倍数：

时间复杂度为 $O(\frac{n}{2}+\frac{n}{3}+\frac{n}{5}+\frac{n}{7}+\frac{n}{11}.....)$

$=O(n*loglogn)\approx O(n)$

2.3 埃氏(Eratosthenes)筛法

遍历每一个素数即可：

```
5   int isprime[100005];
6   void prime(int n){
7       int i,j;
8       for(int i=2;i<=n;i++)
9           isprime[i]=1;//初始都是素数
10      for(int i=2;i<=n;i++){
11          if(isprime[i]){
12              for(j=i*i;j<=n;j+=i){
13                  isprime[j]=0;
14              }
15          }
16      }
17  }
```

[1444] 埃氏筛选求素数

```
const int SIZE = 1e7+5;

int prime[SIZE]; // 第i个素数
bool is_prime[SIZE];    //true表示i是素数

int slove(int n)
{
    int p = 0;
    for(int i = 0; i <= n; i++)
        is_prime[i] = true;  //初始化都是素数
    is_prime[0] = is_prime[1] = false; //0,1不是素数
    for(int i = 2; i <= n; i++)
    {
        if(is_prime[i]) //
        {
            prime[p++] = i; //计算素数的个数,也记录下了素数
            for(int j = 2 * i; j <= n; j += i)       // 除掉了i的倍数的数字
                is_prime[j] = false;
        }
    }
    return p;
}
```

```
int main()
{
    int n;
    cin >> n;
    int res = slove(n);
    for(int i = 0; i < res; i++)
        printf("%d\n",prime[i]);
}
```

2.4 线性筛选（欧拉筛选）

埃氏筛选中，以n=50为例，30这个数被筛了3次，分别是：

$$2 * 15(p = 2)$$
$$3 * 10(p = 3)$$
$$5 * 10(p = 5)$$

如何去掉这些重复的筛选？每个合数只会被它的最小质因子筛掉，每个合数只会被筛一次。

这就是“快速线性筛选法”，也称为欧拉筛选。快速线性筛选法没有冗余，不会重复筛除一个数，几乎是线性的。

10^7时比埃氏筛法快一倍

10^6时与埃氏筛法差不多

欧拉筛法（线性筛）

- 枚举 2~n 中的每一个数 i：
  - 如果 i 是素数则保存到素数表中；
  - 利用 i 和素数表中的素数 prime[j] 去筛除 i*prime[j] ，为了确保i*prime[j] 只被素数 prime[j] 筛除过这一次，要确保 prime[j] 是i*prime[j] 中最小的素因子，即 i 中不能有比 prime[j] 还要小的素因子。

核心：i只会被最小质数因子筛掉

欧拉筛法（线性筛）

1. 如果i素数的话，素数i乘以不大于i的素数，这样筛除的数是不会重复的（线性筛选的理论基础）。筛出的数都是 n = p1 * p2 的形式，且p1 和 p2 不相等；如：

i = 2: n = 2;
i = 3: 2 * 3,3 * 3;
i = 5: 2 * 5,3 * 5,5 * 5

不会有值相等

2. 如果i是合数，此时i可以表示成递增素数相乘 i = p1 * p2 * ... * pn，$pi \le pj(i <= j)$ ，p1 也是最小的系数。

线性筛 n=50 的筛选过程图示：

	素数表	筛除的数		素数表	筛除的数
2	{2}	{4}	***13***	{2, 3, 5, 7, 11, 13}	{26, 39}
3	{2, 3}	{6, 9}	***14***	{2, 3, 5, 7, 11, 13}	{28}
4	{2, 3}	{8}	***15***	{2, 3, 5, 7, 11, 13}	{30, 45}
5	{2, 3, 5}	{10, 15, 25}	***16***	{2, 3, 5, 7, 11, 13}	{32}
6	{2, 3, 5}	{12}	***17***	{2, 3, 5, 7, 11, 13, 17}	{34}
7	{2, 3, 5, 7}	{14, 21, 35, 49}	***18***	{2, 3, 5, 7, 11, 13, 17}	{36}
8	{2, 3, 5, 7}	{16}	***19***	{2, 3, 5, 7, 11, 13, 17, 19}	{38}
9	{2, 3, 5, 7}	{18, 27}	***20***	{2, 3, 5, 7, 11, 13, 17, 19}	{20}
10	{2, 3, 5, 7}	{20}	***21***	{2, 3, 5, 7, 11, 13, 17, 19}	{42}
11	{2, 3, 5, 7, 11}	{22, 33}	***22***	{2, 3, 5, 7, 11, 13, 17, 19}	{44}
12	{2, 3, 5, 7, 11}	{24}	***...***	...	...

欧拉筛法（线性筛）

```
1  void get_primes(){
2      for(int i=2;i<=n;i++){
3          if(!st[i]) primes[cnt++]=i;
4          for(int j=0;primes[j]*i<=n;j++){
5              st[primes[j]*i]=true;//用最小质因子去筛合数
6              if(i%primes[j]==0)
7                  break;
8          }
9      }
10 }
```

1)当i%primes[j]!=0时,说明此时遍历到的primes[j]不是i的质因子，那么只可能是此时的primes[j]<i的最小质因子,所以primes[j]*i的最小质因子就是primes[j];

2)当有i%primes[j]==0时,说明i的最小质因子是primes[j],因此primes[j]*i的最小质因子也就应该是prime[j]，之后接着用st[primes[j+1]*i]=true去筛合数时，就不是用最小质因子去更新了,因为i有最小质因子primes[j]<primes[j+1],此时的primes[j+1]不是primes[j+1]*i的最小质因子,此时就应该退出循环，避免之后重复进行筛选。

[1634] 线性筛素数

```
4   #define m 5800000
5   #define m1 100000000+10
6   int n,len,a[m];
7   bool vis[m1];
8   int main()
9   {
10      scanf("%d",&n);
11      for(int i = 2;i <= n;i++)
12      {
13          if(!vis[i])
14              a[++len] = i;
15          for(int j = 1;(j <= len )&&(i * a[j] <= n);j++)
16          {
17              vis[a[j] * i] = true;
18              if(!(i % a[j]))
19                  break;
20          }
21      }
22      printf("%d",len);
23      return 0;
24  }
```

素数筛法

```
int m,p[10000],r[10000];//p存质数 r存次幂
void div(int n)
{
    for(int i=2;i*i<=n;i++)
        if(n%i==0)//能分解
        {
            p[++m]=i;
            while(n%i==0)n/=i,r[m]++;
        }
    if(n>1)p[++m]=n,r[m]=1;//如果还有一个大于sqrt(n)的质数
    for(int i=1;i<=m;i++) //输出
      cout<<p[i]<<' '<<r[i]<<endl;
}
```

[3934]Prime Distance

给定两个整数 L,R，求闭区间［L,R］中相邻两个质数差值最小的数对与差值最大的数对。当存在多个时，输出靠前的素数对。

$1 \leqslant L < R < 2^{31}$ (1<=L<R<=2147483647)， $R-L \leqslant 10^6$ 。

- 性质1：若一个数 `n` 是一个合数，必然存在 `2` 个因子 $d, \frac{n}{d}$，假设d <= $\frac{n}{d}$，则d <= $\sqrt{n}$，因此必然存在一个小于等于 $\sqrt{n}$的因子

- 性质2：若x∈[L,R],且x是合数，则一定存在P <= $\sqrt{2^{31}-1}$ (< 50000)，使得 `P` 能整除 `x`，其中 `P < x`.

步骤

- 1、找出1 ~ $\sqrt{2^{31}-1}$ (< 50000)中的所有质因子
- 2、对于1 ~ 50000 中每个质数 `P` ，将 `[L,R]` 中所有 `P` 的倍数筛掉(至少 `2` 倍)
  - 找到大于等于 `L` 的最小的 `P` 的倍数P_0，找下一个倍数时只需要 `+= P` 即可

如何求出在区间 $[\mathrm{L},\mathrm{R}]$ 中最小的 p 的倍数

1. $\lceil \frac{L}{p} \rceil \times p \quad \Rightarrow$ `ceil((double)L / p) * p`
2. $\lfloor \frac{L+p-1}{p} \rfloor \times p \quad \Rightarrow$ `(L + p - 1) / p * p`

分析

由于L和R的范围很大，所以，直接通过素数筛跑出答案会超时，但是，L和R的区间<=1e6，因此我们可以考虑求出[L,R]中的所有素数。

如果直接求，时间复杂度为[R-L] *sqrt(x),也会超时。还是需要考虑筛选法，但常规筛选法是从[2,N]全区域筛选，也会超时，此时我们要思考如何缩小筛选的范围？

我们可以先通过素数筛跑出2~sqrt（R）的所有素数，对于每个素数a，把[L，R]中所有能被a整除的数（i*a）（合数）标记，所有未被标记的数就是素数，把相邻的2个素数比较，找出差最小和最大的。

```cpp
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
typedef long long LL;
//sqrt(2^31 - 1)
const int N = 1e6 + 10;
```

```cpp
bool st[N];
int primes[N], cnt;
void get_primes(int n) {
    memset(st, 0, sizeof st);
    cnt = 0;
    for (int i = 2; i <= n; ++ i) {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] * i <= n; ++ j) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

```
23 int main() {
24     int l, r;
25     while (~scanf("%d%d", &l, &r)) {
26         get_primes(50000);
27
28         //把[l,r]区间内所有的合数用他们的最小质因子筛掉
29         memset(st, 0, sizeof st);
30         for (int i = 0; i < cnt; ++ i) {
31             LL p = primes[i];
32             for (LL j = max(2 * p, (l + p - 1) / p * p); j <= r; j += p)
33                 st[j - l] = true;
34         }
35
36         //剩下的所有的都是素数了
37         cnt = 0;
38         for (int i = 0; i <= r - l; ++ i)
39             if (!st[i] && i + l > 1)
40                 primes[cnt ++ ] = i + l;
```

Report Window

```
if (cnt < 2) printf("There are no adjacent primes.\n");
else {
    //计算间隔
    int minp = 0, maxp = 0;
    for (int i = 0; i + 1 < cnt; ++ i) {
        int d = primes[i + 1] - primes[i];
        if (d < primes[minp + 1] - primes[minp]) minp = i;
        if (d > primes[maxp + 1] - primes[maxp]) maxp = i;
    }
    printf("%d,%d are closest, %d,%d are most distant.\n",
    primes[minp], primes[minp + 1],
    primes[maxp], primes[maxp + 1]);
}
```

2 因式分解

- 表述：若整数 $N \geq 2$，那么 N 一定可以惟一地表示为若干素数的乘积。形如

$$N = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k} \quad (p_i\text{为素数}, r_i \geq 0)$$

18 = 2 * 3 * 3 = 2^1 * 3^2

45 = 3 * 3 * 5

.....

显而易见，对于一个合数N，一定存在两个数，使其相乘为N，这两个数就是N的因子。

若这两个数不为素数，则可以继续将这两个数分解，直至得到素数。

综上所述，一个合数N，是由多个素因子相乘得到。

2 因式分解

- 表述：若整数 $N \geq 2$，那么 N 一定可以惟一地表示为若干素数的乘积。形如

$$N = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k} \quad (p_i\text{为素数}, r_i \geq 0)$$

18 = 2 * 3 * 3 = 2^1 * 3^2

45 = 3 * 3 * 5

.....

显而易见，对于一个合数N，一定存在两个数，使其相乘为N，这两个数就是N的因子。

若这两个数不为素数，则可以继续将这两个数分解，直至得到素数。

综上所述，一个合数N，是由多个素因子相乘得到。

1 [3261]质因数分解

已知正整数n是两个不同的质数的乘积，试求出较大的那个质数。

对于100%的数据，$6 \leqslant n \leqslant 2*10^9$。

```
1     for(int i=2;i*i <= n;i++)
2         if(n%i == 0) {
3             cout << n/i << endl;
4             break;
5         }
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int main ()
{
    int n;
    bool flag=0;
    cin>>n;
    for(int i=2;i<=sqrt(n);++i)
        if(n%i==0)
        {
            flag=1;
            cout<<n/i<<endl;
            break;
        }
    if(!flag)
        cout<<1<<endl;
    return 0;
}
```

2 [2658]质因数分解2

给定 n 个正整数 ai，将每个数分解质因数，并按照质因数从小到大的顺序输出每个质因数的底数和指数。

第一行包含整数 n。 接下来 n 行，每行包含一个正整数 ai。

对于每个正整数 ai，按照从小到大的顺序输出其分解质因数后，每个质因数的底数和指数，每个底数和指数占一行。

每个正整数的质因数全部输出完毕后，输出一个空行。

$1 \leqslant n \leqslant 100 \quad 2 \leqslant ai \leqslant 2 \times 10^9$

x 的质因子最多只包含一个大于 根号x 的质数。如果有两个，这两个因子的乘积就会大于 x，矛盾。

i 从 2 遍历到 根号x。 用 x / i，如果余数为 0，则 i 是一个质因子。

s 表示质因子 i 的指数，x /= i 为 0，则 s++， x = x / i 。

最后检查是否有大于 根号x 的质因子，如果有，输出。

```
int main()
{
    int n;
    cin >> n;
    while (n -- )
    {
        int x;
        cin >> x;
        divide(x);
    }

    return 0;
}
```

```
void divide(int x)
{      //i <= x / i:防止越界，速度大于 i < sqrt(x)
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)//i为底数
        {
            int s = 0;//s为指数
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;//输出
        }
    //如果x还有剩余，单独处理
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}
```

3 [1329] 质数因子分解

在数学课上，老师给同学们讲解了如何对一个正整数n进行质数因子分解，如60=2*2*3*5，其质数因子为2、2、3、5。课后，老师给同学们布置了一道习题：求一个正整数的质数因子分解形式中，各质数因子之和（如果质数因子重复的话，则必须多次计算；注意1不是质数）。例如60的质数因子之和为12。请你通过编程的方法帮同学们解答这道习题。

输入格式

每个测试数据n占一行，

$2 \leqslant n \leqslant 2 \times 10^9$

4 [4329]: Sherlock and his girlfriend

夏洛克有了一个新女朋友。情人节快到了，他想送给她一些珠宝。他买了n件珠宝。第i件的价格等于i+1，即珠宝的价格为2,3,4，。。。。n+1。

沃森给夏洛克一个挑战，让他给这些珠宝上色，如果其中一件的价格是另一件价格的素因子，那么两件的颜色就不能一样。

此外，沃森要求他尽量减少使用不同颜色的数量。

输入一行包含单个整数n（1≤N≤1000000）- 珠宝的数量。

输出的第一行应该包含一个整数k，即在给定约束条件下可用于为珠宝件上色的最小颜色数。

下一行由n个空格分隔的整数（介于1和k之间）组成，这些整数按照价格上涨的顺序指定每件物品的颜色。

如果有多种方法可以使用k颜色为工件着色，则可以输出其中任何一种。

分析

题目说如果一件珠宝的价格是另一件珠宝价格的素因子，那么两件珠宝就不能有相同的颜色，两者颜色不同。那么一个无论多大的非素数都可以分解为多个因子，所以题目的本质只是让我们区别素数和非素数，所以最多的颜色种类也只有两种，至于素数是1还是非素数是1，都是可以过的。

- n 个点，标号 $2..n+1$，给这些点染色，要求若 a 是 b 的素因子，则 a 和 b 的颜色不同求一种颜色数最少的方案，$n \le 1000$。

- 分析：
- 注意到这是二分图，一边是素数，一边是合数。
- 把素数都染成 1，合数都染成 2 即可。

```cpp
#include<cstring>
using namespace std;
bool isprime(int x) {
    for (int i = 2; i*i <= x; i++) {
        if (x % i == 0)
            return false;
        //判断这个数是否为素数
        }
        return true;
}

int main()
{
    int n;
    cin >> n;
    if (n == 1) {
        cout << "1" << endl;
        cout << "1" << endl;
    }
    else if (n == 2) {
    //n=1和n=2对应2和2，3他们都是素数所以只
    //有一种颜色情况，需要拿出来说明
        cout << "1" << endl;
        cout << "1 1" << endl;
    }
    else {
        cout << "2" << endl;
        for (int i = 1; i <= n; i++) {
            if (isprime(i+1)) cout << "1 ";
            else cout << "2 ";
    //当n>2时无论n多大，都只有素数非素数两种情况
        }
    cout << endl;
    }
```

3 筛选法

常规**试除法**，通常写一个函数对要判断的整数x，时间复杂度为$O(\sqrt{x})$. 如果要判断n的数是否是素数，时间复杂度为$O(n*\sqrt{x})$。

如果能够实现准备好素数表就可以帮助我们更有效地求解素数的相关问题。筛选法可以快速列举给定范围内的所有素数。其时间复杂度不大于$O(n*logn)$。

[7549]最强素数

素数41能写成连续6个素数之和：41=2+3+5+7+11+13。

现在要求n以内的素数中，能表示为最多连续素数之和的那个数，如果有多个答案，请输出最大的那个素数。

输入仅一行，一个整数n。100%的数据，1<=n<=1000000

输出就一个整数，为所求的能表示为最多连续素数和的那个素数。

输入样例 100

输出样例 41

分析

题目要求找[1,n]范围内能表示为最多个连续素数和的素数并输出（如果有多个输出最大的）。

首先筛选出一定范围内所需要的素数并存储，然后在找出不大于 n 的最大素数下标 x，暴力枚举所有prime[0]~prime[x]里所有组合，就可以得到答案。

求出所有素数

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN=1000005;
int p[MAXN];//=0 是素数  =1不是
int prime[MAXN];//存所有的素数
int cnt;//表示素数的个数
int q[MAXN];//q[i]表示长度为i是的最大值

void allPrime(){
    for(int i=2;i<=sqrt(MAXN);i++){
        if(p[i])continue;
        for(int j=i*2;j<=MAXN;j=j+i)
            p[j]=1;
    }
    for(int i=2;i<MAXN;i++){
        if(p[i]==0)
            prime[cnt++]=i;
    }
}
```

统计最长和

```
20  int main(){
21      int n,x;
22      scanf("%d",&n);
23      allPrime();
24      for(x=0;x<cnt;x++)//找出不大于n的最大素数
25          if(prime[x]>n)break;
26      int sum,maxlen=0;
27      for(int i=0;i<x;i++){
28          //枚举所有的组合
29          sum=0;//当前组合的和
30          for(int j=i;j<x;j++){//把i~j之间的所有素数求和
31              sum=sum+prime[j];
32              if(sum>n)break;
33              if(p[sum]==0){//sum是素数
34                  q[j-i+1]=max(sum,q[j-i+1]);//最初长度为j-i+1是的最大值
35                  maxlen=max(maxlen,j-i+1);
36              }
37          }
38      }
39      printf("%d",q[maxlen]);
```

筛法优化素因数分解-1

- 利用埃氏筛法可以快速实现素因数分解，只要在判定素数时记录下每个数值的最小素因数即可。算法实现如下：

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1000000
bool prime[MAXN];
int Minfac[MAXN];
int res[MAXN];
void init(int n){//初始化
    prime[0] = prime[1] = false;//0和1都不是素数
    Minfac[0] = Minfac[1] = -1;//两数都没有素因子
        for( int i=2;i<=n;++i){
            prime[i] = true;
            Minfac[i] = i;
        }
}
```

筛法优化素因数分解-2

```
16  void eratos(int n){
17      for( int i=2;i*i<=n;++i){
18          if(prime[i]==true){//如果是质数
19              for( int j=i*i;j<=n;j+=i){
20                  prime[j] = false;
21                  //将其倍数设为false
22                  if(Minfac[j]==j){
23                      //如果之前没有找打这个数的最小素因子
24                      Minfac[j] = i;
25                          //i就是它的最小素因子
26                  }
27              }
28          }
29      }
30  }
```

筛法优化素因数分解-3

```
32 void factor(int x){
33     int i=1;
34     while(x>1){
35         res[i++]=Minfac[x];//将最小的素因子加入数组
36         x /= Minfac[x];//除掉它
37     }
38     i--;
39     cout<<"x=";
40     for(int j=1;j<=i;j++){
41         if(j==i) cout<<res[j];
42         else cout<<res[j]<<"*";
43     }
44 }
45 int main(){
46     int n;
47     cin>>n;
48     init(n);eratos(n);factor(n);
49     return 0;
50 }
```