Наибольшим общим делителем **(НОД)** чисел ***a*** и ***b*** называется самое большое натуральное число ***d,*** являющееся делителем для каждого из чисел ***a*** и ***b.***

Наибольший общий делитель чисел ***a*** и ***b*** будем обозначается одним из символов – ***НОД(a; b).***

Один из способов нахождения ***НОД*** вытекает непосредственно из определения: действительно, можно было бы сначала найти все делители числа ***a***, затем – числа ***b.*** После этого отобрать общие делители и выбрать из них наибольший. Такой путь, однако, был бы весьма нерациональным, особенно для больших чисел.

Другой, наиболее рациональный, способ нахождения ***НОД*** был предложен ***Евклидом*** (III в. до н.э.).

Алгоритм Евклида. Если ***a*** разделить с остатком на $b \neq 0$, затем разделить с остатком ***b*** на полученный остаток, затем разделить с остатком первый остаток на второй и т.д., то последний, отличный от нуля, остаток равен ***НОД(a; b).***

Если ***НОД*** чисел равен ***единице***, то числа называются ***взаимно простыми.***

Пусть ***a и b*** – произвольные натуральные числа. Если ***m*** – общее кратное чисел ***a и b***, то, по транзитивности отношения делимости, каждое из чисел ***2m, 3m, 4m, ...*** также является общим кратным чисел ***a и b.*** Среди натуральных общих кратных, по принципу наименьшего числа, существует наименьшее общее кратное.

Наименьшим общим кратным (НОК) чисел ***a и b*** называется наименьшее натуральное число ***m,*** такое что *оно* делится на каждое из чисел ***a и b.***

Наименьшее общее кратное чисел ***a и b*** будем обозначать одним из символов – ***НОК(a;b)***

Любое общее кратное двух чисел делится на их наименьшее общее кратное.

НОК и НОД двух натуральных чисел ***a и b*** связаны соотношением

$$ab = НОД(a; b) \cdot НОК(a; b).$$

НОК двух взаимно простых чисел равно их произведению.

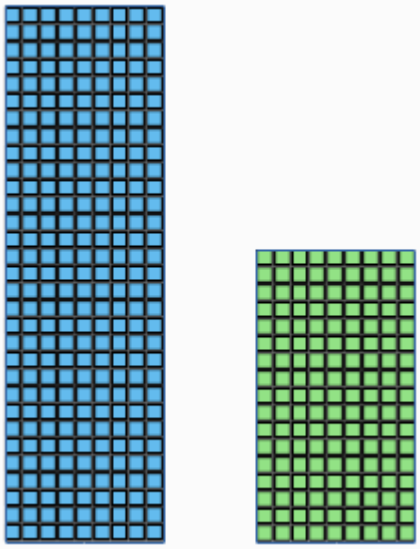
НОД и НОК любого конечного множества чисел вычисляется по принципу: ***НОД(a,в,c)=НОД(НОД(a,в),c) и НОК(a,в,c)=НОК(НОК(a,в),c)***

Алгоритм расчета наибольшего общего делителя

Даны два целых числа **A** и **B**, их наибольший общий делитель — такое число C, что на него **делится без остатка и A, и B.**

Вспомним **алгоритм Евклида** и предлагаю попытаться визуализировать задачу.

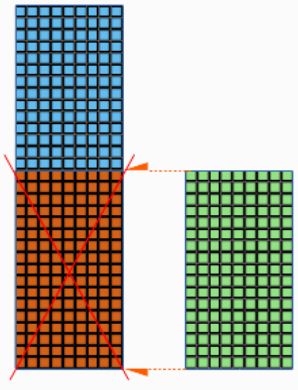
Целое число — это количество чего-либо неделимого. На следующей картинке два числа показаны в виде прямоугольников, под **значением числа** можно понимать количество «блоков в прямоугольнике». Показано схематично.

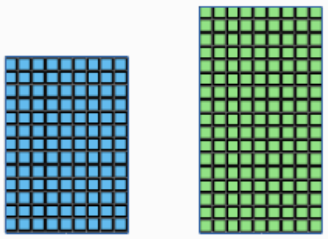
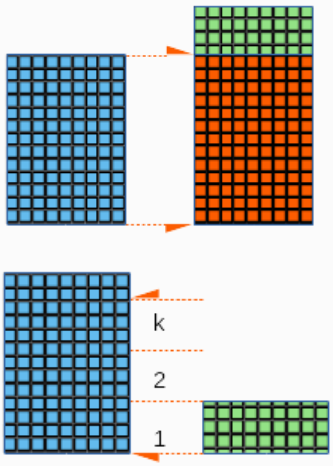


Эффективный алгоритм расчета НОД строится на следующих наблюдениях (постарайтесь их «почувствовать»):

1. если A делится на B без остатка — то НОД(A, B) = B;
2. любое число, которое делит оба числа A и B, делит также и A-B, поэтому
   НОД(A, B) <= НОД (A — B, B);
   То есть уменьшение числа A на значение B не повлияет на результат вычисления НОД;
3. мы можем воспользоваться предыдущим пунктом несколько (t) раз — если A = B*t + r для целых чисел t и r — то НОД(A, B) = НОД(r, B).

Из второго пункта следует идея следующего алгоритма поиска НОД: *Отнимать от большего меньшее, пока числа не станут равны. Полученное число и является наибольшим общим делителем.* Такой алгоритм будет работать значительно быстрее, чем полный перебор, но и его можно улучшить — посмотрим визуализацию (исходное состояние показано выше):

Действие	Иллюстрация
Вычитаем из большего меньшее.	

Новое исходное состояние (можно повторить предыдущий шаг)	
Вычитаем из большего меньшее. Новое исходное состояние. На этот раз можно **многократно** вычитать число A из числа B (отношение порядка между ними не изменится).	

В какой-то момент числа окажутся равны и мы получим результат. Этот момент обязательно настанет. В крайнем случае, когда оба числа станут равны единице (потому что это ей кратны любые целые числа).

Из последней иллюстрации видно, что многократное вычитание можно заменить на получение остатка от деления (об этом же говорит третье «наблюдение»).

Тогда алгоритм можно записать на псевдокоде следующим образом:

```
1.  наибольший_общий_делитель(a, b) {
2.    если a делится на b без остатка то - верни b;
3.    если b делится на a без остатка то - верни a;
4.
5.    если a > b - то верни наибольший_общий_делитель(a mod b, b);
6.    иначе верни наибольший_общий_делитель(a, b mod a);
7.  }
```

2 Алгоритм расчета наименьшего общего кратного

Наименьшее общее кратное двух целых чисел A и B есть наименьшее натуральное число, которое делится на A и B без остатка.

Чтобы лучше понять о чем речь — предлагаю такую **геометрическую интерпретацию**: значения A и B задают длины отрезков. НОК — это длина другого отрезка, который можно составить как из целого количества отрезков A, так и отрезков B:

Для любых чисел мы можем найти общее кратное

C = A*B, однако, оно не всегда будет **наименьшим**. Примитивный алгоритм вычисления НОК мог бы заключаться в переборе всех чисел от max(A, B) до A*B. Однако, это не самое эффективное решение. На самом деле, если длина отрезка A = 4, а B = 3, то перебирать надо все отрезки, кратные 4, т.е. max(A, B).

Обратите снимание, что если A и B взаимнопростые (иными словами НОД(A, B) = 1) — то НОК(A, B) = A*B. Если же у этих чисел есть делители d0, d1, ..., dn, то их **общими кратными** будут числа: (A*B)/d0, (A*B)/d1, … (A*B)/dn. Значит, чтобы найти **наименьшее** общее кратное, нужно найти **наибольший из делителей**:

```
наименьшее_общее_кратное(a, b) {
  верни (A*B)/наибольший_общий_делитель(a, b);
}
```

Реализуем алгоритмы на языке C++.

Первый (не самый эффективный вариант) вычисления НОД через многократное вычитание меньшего числа из большего:

```
unsigned int greatest_common_divisor(unsigned int a, unsigned int b) {
  if (a == b)
    return a;
  if (a > b)
    return greatest_common_divisor(a-b, b);
  return greatest_common_divisor(a, b-a);
}
```

Более эффективный алгоритм вычисления НОД, использующий остаток от деления:

```
unsigned int greatest_common_divisor(unsigned int a, unsigned int b) {
  if (a % b == 0)
    return b;
  if (b % a == 0)
    return a;

  if (a > b)
    return greatest_common_divisor(a%b, b);
  return greatest_common_divisor(a, b%a);
}
```

Функция вычисления НОК:

```
unsigned int least_common_multiple(unsigned int a, unsigned int b)
{
  return (a*b)/greatest_common_divisor(a,b);
}
```

Вычислить НОД можно и без рекурсии.

```
#include <iostream>
using namespace std;
int main() {
  int y, x;
  cin >> x >> y;
  while (x != y) {
    if (x>y) {
      x = x-y;
    }
    else {
      y = y-x;
    }
  }
  cout << x;
}
```

Еще одна **реализация алгоритма Евклида** использует функцию ***swap:***

```
int NOD(int a, int b) {
    if (a < b) {
        swap(a, b);
    }
    while (a % b != 0) {
        a = a % b;
        swap(a, b);
    }
    return b;
}
```

Функция ***swap*** выполняет обмен значений двух переменных. Она есть в стандартной библиотеке. Если в данный момент не понятно, как она работает, то замените ее на:

```
int t = a;
a = b;
b = t;
```

Задача:

Найти наименьшее общее кратное для всех элементов массива — минимальное число, которое делится на все элементы массива без остатка. Также, найти НОД всех элементов массива.

Решение:

Вот тут приведены алгоритмы расчета НОК и НОД для двух чисел. Ясно, что наиболее эффективный алгоритм расчета НОК двух чисел — это произведение чисел поделить на их НОД. По содержимому статьи ясно что НОК(a1, a2, a3, ... aN) равен НОК(НОК(НОК(A1, A2), A3)..., AN). Таким образом, для расчета НОК массива чисел надо многократно расчитывать НОД двух чисел.

```c
#include <stdio.h>
#include <stdlib.h>

void read_array(int n, int** values) {
    for (int i = 0; i < n; ++i) {
        printf("values[%d] = ", i);
        scanf("%d", &((*values)[i]));
    }
}

void print_array(int n, int* values) {
    for (int i = 0; i < n; ++i) {
        printf("values[%d] = %d\n", i, values[i]);
    }
}

void swap(int* a, int* b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int gcd(int a, int b) {
    if (a < b) {
        swap(&a, &b);
    }
    while (a % b != 0) {
        a = a % b;
        swap(&a, &b);
    }
    return b;
}

int gcd_n(int n, int* values) {
    if (n == 0)
        return -1;
    if (n == 1)
        return values[0];
    int gcd_value = gcd(values[0], values[1]);
    for (int i = 2; i < n; ++i) {
        gcd_value = gcd(gcd_value, values[i]);
    }
    return gcd_value;
}

```

```
46.  int lcm(int a, int b) {
47.      return (a*b)/gcd(a, b);
48.  }
49.
50.  int lcm_n(int n, int* values) {
51.      if (n == 0)
52.          return -1;
53.      if (n == 1)
54.          return values[0];
55.      int lcm_value = lcm(values[0], values[1]);
56.      for (int i = 2; i < n; ++i) {
57.          lcm_value = lcm(lcm_value, values[i]);
58.      }
59.      return lcm_value;
60.  }
61.
62.  int main() {
63.      int n;
64.      int *values;
65.
66.      printf("n: ");
67.      scanf("%d", &n);
68.
69.      values = malloc(sizeof(int) * n);
70.
71.      read_array(n, &values);
72.
73.      printf("lcm: %d", lcm_n(n, values));
74.
75.      free(values);
76.      return 0;
77.  }
```

Решите задачи:

а. **acmp** № 14
б. **acmp №** 394