Задача A. Проверка на простоту

Задача была разобрана на лекции. Код для решения.

```
for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
        cout << "composite";
        return 0;
    }
}
```

Задача B. Количество делителей

Задача была разобрана на лекции. Код для решения.

```
for (int i = 1; i * i <= x; ++i) {
    if (x % i == 0) {
        cnt++;
        if (i != x / i) {
            cnt++;
        }
    }
}
```

Задача C. Разложение на простые

Задача была разобрана на лекции. Код для решения.

```
for (int i = 2; i * i <= n; i++) {
    while (n % i == 0) {
        n /= i;
        cout << i << " ";
    }
}
if (n > 1) {
    cout << n;
}
cout << "prime";
```

Задача D. Разложение на простые++

Задача была разобрана на лекции. Код для решения.

```
for (int i = 2; i * i <= n; i++) {
    int cnt = 0;
    while (n % i == 0) {
        n /= i;
        cnt++;
    }
    // i^cnt
}
if (n > 1) {
    // n^1
}
```

Задача E. Представление чисел

Задача была разобрана в видеолекции

Задача F. Алгоритм Евклида

Наибольшим общим делителем (англ. greatest common divisor) целых неотрицательных чисел a и b называется наибольшее число x, которое делит одновременно и a, и b.

Алгоритм Евклида находит gcd двух чисел a и b за $\mathcal{O}(\log \min(a, b))$, основываясь на следующей несложной формуле:

- a, если $b = 0$
- $gcd(b, a - b)$ иначе

Здесь предполагается, что $a > b$.

Докажем корректность этой формулы:

Если $g = gcd(a, b)$ делит и a, и b, то их разность $(a - b)$ тоже будет делиться на g.

Никакой больший делитель d числа b не может делить число $(a - b)$: если $d > g$, то d не может делить a, а значит и не делит $(a - b)$.

Прямая рекурсивная реализация:

```
int gcd(int a, int b) {
    if (a < b)
        swap(a, b);
    if (b == 0)
        return a;
    else
        return gcd(b, a - b);
}
```

Задача G. Сложить две дроби

$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + c \cdot b}{b \cdot d} = \frac{a \cdot \frac{HOK(b,d)}{b} + c \cdot \frac{HOK(b,d)}{d}}{HOK(b,d)}$

Задача H. Целые точки на отрезке

Если (x_1, y_1) и (x_2, y_2) — целочисленные концы отрезка, то количество целочисленных точек на нем равно НОД $(|x_2 - x_1|, |y_2 - y_1|) + 1$.

Задача I. МегаНОД

Задача была разобрана на лекции. Код для решения.

```
for (int i = 0; i < n; ++i) {
    int y;
    cin >> y;
    x = gcd(x, y);
}
```

Задача J. До последней стружки

Предположим, что имеется x заготовок; подсчитаем количество втулок, которое можно получить из этих заготовок. На первом этапе получится x втулок и x единиц стружки; из них в свою очередь можно получить x/m новых втулок (целочисленное деление). Значит, на втором этапе имеем $s = x + x / m$ втулок и $x / m + x \,\%\, m$ единиц стружки. (Число $x \,\%\, m$ равно остатку от деления x на m.) На следующем этапе рассуждения повторяются: из $x_1 = x / m$ единиц стружки снова получаем x_1 / m втулок и такое же количество единиц стружки, то есть число втулок увеличится на x_1 / m. Таким образом, для подсчёта числа бронзовых втулок из x заготовок можно использовать процедуру:

```
long long g(long long x) {
    long long s = x;
    while (x >= m)    {
        s += x / m;
        x = x / m + x % m;
    }
    return s;
}
```

Как найти необходимое количество заготовок? Например, можно воспользоваться бинарным поиском по ответу. Идея этого алгоритма состоит в следующем. Будем искать нужное количество заготовок в виде неизвестного числа x на отрезке $[1; n]$. Выберем в качестве начального значения середину этого отрезка, то есть число $x = n/2$. Если этого количества заготовок недостаточно, будем продолжать дальнейший поиск на отрезке $[n/2; n]$, иначе — на отрезке $[1; n/2]$. На каждом шаге выбираем середину текущего отрезка и сравниваем количество втулок, которое можно получить (процедура g), с требуемым количеством. Если полученных втулок меньше, передвинем *левую* границу искомого отрезка в эту середину, иначе — передвинем *правую* границу. Продолжаем этот процесс до тех пор, пока разность между левой и правой границей текущего отрезка больше 1.

Сложность этого алгоритма — $O(\log n \cdot \log m)$.