A. Конфеты

(Автор задачи — *Киндер М.И.*)

Подсчитаем количество конфет, которое должно быть одинаковым в каждой коробке. Для этого найдём общее количество конфет во всех коробках $sum = a_1 + a_2 + \ldots + a_n$ и разделим на число коробок n. Если значение этого *среднего арифметического* — целое, то есть остаток от деления sum на n равен нулю, то Карлсону ничего съедать и перекладывать не придётся. В общем случае, остаток $eat := sum \bmod n$ определяет число конфет, которое нужно съесть Карлсону. Теперь оставшиеся конфеты можно разложить поровну между коробками. Для этого из каждой коробки, где число конфет a_i больше среднего арифметического $mean := sum \operatorname{div} n$, нужно переложить $a_i - mean$ конфет в коробки с меньшим числом конфет. Общее число перекладываемых конфет равно сумме по всем таким i.

Приведём основной фрагмент кода на языке Pascal:

```
eat := sum mod n;
mean := sum div n;

sum := 0;
for i := 1 to n do
    if a[i] > mean then Inc(sum, a[i] – mean);

write(eat, ' ', sum – eat);
```

B. Похожие числа

(Автор задачи — *Киндер М.И.*)

Для чисел a и b подсчитаем количество вхождений каждой цифры m от 0 до 9. Обозначим их через $d[a,m]$ и $d[b,m]$ соответственно. Для каждой цифры m от 0 до 9 осталось сравнить значения $d[a,m]$ и $d[b,m]$. Если для какой-то цифры m они совпадают, степень похожести увеличиваем на единицу.

Приведём основной фрагмент кода на языке Pascal:

```
readln(n);
for i := 1 to 2 do
    for j := 1 to n do begin
        read(m);
        Inc(d[i,m]);
    end;

sum := 0;
for i := 0 to 9 do
    if d[1,i] > d[2,i] then Inc(sum, d[2,i]) else Inc(sum, d[1,i]);

write(sum);
```

C. Штабеля

(Автор задачи — *Киндер М.И.*)

Все числа каждого набора отсортируем по возрастанию. Теперь осталось проверить, что каждое число набора не меньше суммы всех предыдущих. Если это условие выполняется для всех k чисел набора (кроме первого), выводим yes; иначе — выводим no.

Оценим сложность этого алгоритма. Для сортировки массива из k элементов понадобится $O(k \log k)$ операций, для проверки «прочности» штабеля нужно еще $O(k)$ операций. Действительно, когда проверяется условие «прочности» для очередного числа $weight[i+1]$ набора, нам нужно вычислить сумму $sum[i]$ всех стоящих слева от $weight[i+1]$ чисел, причём $sum[i]$ получается из предыдущего значение суммы $sum[i-1]$ добавлением единственного слагаемого $weight[i]$. Значит, для проверки условия $sum[i] \le weight[i+1]$ на очередном шаге потребуется всего две дополнительные операции. Поэтому проверка прочности штабеля (после сортировки чисел) потребует еще $O(k)$ операций. Поскольку таких наборов n, общая сложность алгоритма $O(n \cdot k \log k)$.

Приведём основной фрагмент кода на языке Pascal:

```
flag := true;
sum := weight[1];
for i := 2 to k do
    if weight[i] < sum then begin
       flag := false; break;
       end
    else Inc(sum, weight[i]);
if flag then writeln('yes') else writeln('no');
```

D. Цепочка вычитаний

(Автор задачи — *Киндер М.И.*)

Прежде всего, отметим, что на любом шаге среди чисел ровно одно нечётное. Вначале — это число 1. На очередном шаге вычитаются либо два чётных числа, либо чётное и нечётное числа, так что количество нечётных снова остаётся прежним (и равным одному). Поэтому последнее оставшееся число всегда будет нечётным. Значит, если требуемое число k — чётное, выводим –1.

Покажем теперь, как получить *любое нечётное* число $k < 2^n$.

Предположим, что мы уже умеем получать *все нечётные* $k < 2^{n-1}$ с помощью чисел 1, 2, 2^2, … , 2^{n-1}, и пусть теперь задан набор 1, 2, 2^2, …, 2^{n-1}, 2^n. Рассмотрим нечётные числа в промежутке от 1 до 2^n, отметим его середину — число 2^{n-1}.

Если заданное нечётное $k < 2^{n-1}$, то по предположению его можно получить с помощью чисел 1, 2, 2^2, … , 2^{n-1}. Но тогда его можно получить и с помощью исходного набора 1, 2, 2^2, …, 2^{n-1}, 2^n. Для этого достаточно на первом шаге заменить пару чисел 2^{n-1} и 2^n их разностью, равной $2^n - 2^{n-1} = 2^{n-1}$, и мы вновь приходим к набору 1, 2, 2^2, … , 2^{n-1}, из которого число k уже получается.

Если же нечётное $k > 2^{n-1}$, то заменим его на число $k' = 2^n - k < 2^{n-1}$, которое — опять же по предположению — можно получить из набора 1, 2, 2^2, … , 2^{n-1}. Поэтому сначала с помощью $(n-1)$ операций вычитания получаем число $k' = 2^n - k$, а затем последней операцией заменяем пару чисел 2^n и k' их разностью $2^n - k' = 2^n - (2^n - k) = k$.

Осталось правильно реализовать указанный алгоритм. Это можно сделать, например, с помощью стека.

Председатель жюри М.И. Киндер