Муниципальный этап всероссийской олимпиады школьников по информатике
Москва, 11 декабря 2016 г.

Решения заданий для 9–11 классов

Ограничение по времени работы программы во всех задачах: 1 секунда.
Каждая задача оценивается в 100 баллов.

Задача 1. Выборы в США

Условие

Выборы президента США проходят по непрямой схеме. Упрощённо схема выглядит так. Сначала выборы проходят по избирательным округам, на этих выборах голосуют избиратели (то есть все граждане, имеющие право голоса). Затем голосование проходит в коллегии выборщиков, на этих выборах каждый избирательный округ представлен одним выборщиком, который голосует за кандидата, победившего на выборах в данном избирательном округе. Кандидатов в президенты несколько, но реально борьба разворачивается между двумя кандидатами от основных партий, поэтому для победы в выборах кандидату нужно обеспечить строго больше половины голосов в коллегии выборщиков. Но для того, чтобы выборщик проголосовал за данного кандидата, необходимо, чтобы в его избирательном округе этот кандидат также набрал строго больше половины голосов избирателей. Известны случаи (например, в 2016 году), когда из-за такой непрямой избирательной системы в выборах побеждал кандидат, за которого проголосовало меньше избирателей, чем за другого кандидата, проигравшего выборы.

Пусть коллегия выборщиков состоит из N человек, то есть имеется N избирательных округов. Каждый избирательный округ, в свою очередь, состоит из K избирателей. Определите наименьшее число избирателей, которое могло проголосовать за кандидата, одержавшего победу в выборах.

Программа получает на вход два целых числа N и K ($1 \le N \le 10^3$, $1 \le K \le 10^6$) и должна вывести одно целое число – искомое количество избирателей.

Пример входных и выходных данных

Ввод	**Вывод**	**Примечание**
5 3	6	Чтобы данный кандидат получил большинство в коллегии выборщиков, необходимо, чтобы 3 из 5 выборщиков проголосовали за него, то есть кандидат должен одержать победу в 3 округах. Каждый округ состоит из 3 избирателей, поэтому для победы в округе необходимо набрать 2 голоса в данном округе.

Решение

Подсчитаем, какое количество выборщиков должно поддержать победившего кандидата. Для этого достаточно разобрать случаи чётного и нечётного значения N. Несложно заметить, что в обоих случаях ответом будет выражение (N // 2 + 1), где «//» – операция целочисленного деления в Python (аналог операции `div` в Паскале). Для проверки этого выражения достаточно вычислить его для чётного и для нечётного N.

Аналогично, в каждом округе, в котором победил данный кандидат, за него должно проголосовать (K // 2 + 1) избиратель из общего числа избирателей K в данном округе. А в остальных округах, в которых данный кандидат проиграет, за него никто не проголосует. Ответом будет произведение (N // 2 + 1) × (K // 2 + 1).

Приведём пример решения на языке Python (100 баллов).

```
n = int(input())
k = int(input())
print((n // 2 + 1) * (k // 2 + 1))
```

Задача 2. Преобразование дроби

Дана правильная рациональная несократимая дробь a / b. С этой дробью выполняется следующая операция: к числителю и знаменателю дроби прибавляется 1, после чего дробь сокращается. Определите, можно ли при помощи таких операций из дроби a / b получить другую правильную дробь c / d.

Программа получает на вход четыре целых числа a, b, c, d, причём $0 < a < b \le 10^5$, $0 < c < d \le 10^5$, числа a и b взаимно простые, числа c и d взаимно простые, $a / b \ne c / d$.

Программа должна вывести одно натуральное число – сколько описанных операций нужно применить, чтобы из дроби a / b получить дробь c / d. Если это сделать невозможно, программа должна вывести число 0.

Примеры входных и выходных данных

Ввод	Вывод	Примечание
1 3 2 3	2	Дана дробь 1 / 3. После первой операции получается дробь 2 / 4, которая сокращается до 1 / 2. После второй операции получается дробь 2 / 3.
2 3 1 3	0	Получить из дроби 2 / 3 дробь 1 / 3 невозможно.

Система оценивания

Решение, правильно работающее только для случаев, когда все числа не превосходят 100, будет оцениваться в 60 баллов.

Решение

В условии задачи сказано, что $a < b$. Можно заметить, что в результате прибавления к числителю и знаменателю дроби числа 1 значение дроби увеличивается (так как $(a + 1) / (b + 1) > a / b$, это можно доказать раскрыв скобки в эквивалентном неравенстве $(a + 1)\, b > (b + 1)\, a$). Таким образом, после каждой применённой операции значение дроби будет возрастать.

Также необходимо заметить, что если применить операцию много раз, то числитель и знаменатель дроби можно сделать сколь угодно большими, а разница между числителем и знаменателем не увеличивается. Поэтому если применить операцию много раз, то в результате дробь можно сделать сколь угодно близкой к единице (в математике говорят, что предельное значение дроби равно 1, если число операций стремится к бесконечности).

Поэтому в результате применения таких операций исходная дробь когда-нибудь станет больше или равна той дроби, которую необходимо получить. Если исходная дробь станет строго больше необходимой дроби, то в дальнейшем они уже никогда не станут равны.

Решение задачи заключается в том, что к дроби нужно применять описанную операцию, подсчитывая количество произведённых операций. После каждой операции находить наибольший общий делитель числителя и знаменателя, делить числитель и знаменатель дроби на их наибольший общий делитель. Этот процесс продолжается до тех пор, пока дробь не станет больше или равна второй данной дроби. Если получили результат равный второй дроби, то программа выводит количество выполненных операций, иначе

программа выводит число 0.

Приведём пример решения на языке Python (100 баллов).

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

a = int(input())
b = int(input())
c = int(input())
d = int(input())

ans = 0
while a * d < c * b:
    a += 1
    b += 1
    m = gcd(a, b)
    a //= m
    b //= m
    ans += 1
if a == c and b == d:
    print(ans)
else:
    print(0)
```

В этом решении `def gcd(a, b)` – это объявление функции `gcd`, которая вычисляет наибольший общий делитель чисел *a* и *b* при помощи алгоритма Евклида. В этой функции берётся остаток от деления числа *a* на число *b* при помощи операции взятия остатка «%» (аналог операции `mod` в Паскале). Отметим, что если реализовать алгоритм Евклида при помощи операций вычитания, а не взятия остатка от деления, то такая реализация будет работать существенно медленнее. Пример медленной реализации алгоритма Евклида:

```
def gcd(a, b):
    while a != 0 and b != 0:
        if a > b:
            a = a – b
        else:
            b = b - a
    return a + b
```

Например, если $a = 1$, $b = 10^5$, то такая реализация будет вычитать из числа *b* по 1, и суммарно будет выполнено 10^5 вычитаний, в то время как реализация алгоритма Евклида с использованием остатка от деления вместо вычитаний выполнит одну операцию взятия остатка. Поэтому решения, в которых алгоритм Евклида реализован при помощи вычитаний, не проходят тесты с большими входными числами из-за ограничений по времени работы программы и набирают неполный балл.

Аналогично неполный балл набирают решения, в которых поиск наибольшего общего делителя чисел *a* и *b* осуществляется не при помощи алгоритма Евклида, а перебором всех возможных общих делителей от 2 до $\min(a, b)$.

Задача 3. Инопланетный геном

Геном жителей системы Тау Кита содержит 26 видов оснований, для обозначения которых будем использовать буквы латинского алфавита от A до Z, а сам геном записывается строкой из латинских букв. Важную роль в геноме играют пары соседних оснований, например, в геноме «ABBACAB» можно выделить следующие пары оснований: AB, BB, BA,

AC, CA, AB.

Степенью близости одного генома другому геному называется количество пар соседних оснований первого генома, которые встречаются во втором геноме.

Вам даны два генома, определите степень близости первого генома второму геному.

Программа получает на вход две строки, состоящие из заглавных латинских букв. Каждая строка непустая, и её длина не превосходит 10^5.

Программа должна вывести одно целое число – степень близости генома, записанного в первой строке, геному, записанному во второй строке.

Пример входных и выходных данных

Ввод	**Вывод**	**Примечание**
ABBACAB BCABB	4	Следующие пары оснований первого генома встречаются во втором геноме: AB, BB, CA, AB. Обратите внимание на то, что пара AB в первом геноме встречается два раза, поэтому и подсчитана в ответе два раза.

Система оценивания

Решение, правильно работающее только для случаев, когда длины данных строк не превосходят 100, будет оцениваться в 60 баллов.

Решение

Можно реализовать простое «наивное» решение – переберём циклом по первой строке все пары соседних символов в этой строке, каждую пару найдём во второй строке. Если пара нашлась, то увеличим значение ответа на 1 и выведем в конце количество найденных пар.

Пример решения на языке Паскаль.

```
var s1, s2: string;
    i, ans: longint;
begin
    readln(s1);
    readln(s2);
    ans := 0;
    for i := 1 to length(s1) - 1 do
        if pos(copy(s1, i, 2), s2) <> 0 then
            ans := ans + 1;
    writeln(ans)
end.
```

В этом решении при помощи функции `copy` из первой строки извлекается два соседних символа, а потом при помощи функции `pos` осуществляется проверка, содержатся ли эти два символа во второй строке.

Пример аналогичного решения на языке Python.

```
s1 = input()
s2 = input()
ans = 0
for i in range(len(s1) - 1):
    if s1[i:i + 2] in s2:
        ans += 1
print(ans)
```

В этом решении пара соседних символа извлекается при помощи среза `s1[i:i + 2]`, а поиск пары символов в строке `s2` осуществляется при помощи оператора `in`.

Приведенные выше решения набирают 60 баллов, так как проверка каждой пары первой строки осуществляется путём просмотра всей второй строки. Такое решение имеет

сложность O(nm), где n – длина первой строки, m – длина второй строки и из-за неэффективности проходит только те тесты, в которых длина строк небольшая.

Для того, чтобы набрать большее число баллов, необходимо придумать более эффективный способ проверки того, содержится ли данная пара во второй строке. Для этого необходимо каким-то образом сначала обработать вторую строку и сохранить информацию о том, какие пары символов встречаются во второй строке в какой-либо структуре данных.

Можно использовать массив строк, в котором будут сохраняться все пары символов из второй строки. Для примера из условия, в котором вторая строка BCABB, получится массив ["BC", "CA", "AB", "BB"]. При этом если какая-либо пара уже встречается в этом массиве, добавлять ее не нужно, чтобы размер массива был невелик. Максимальная длина этого массива k равна количеству возможных пар символов, то есть $k = 26 \times 26 = 676$. Для поиска данной пары в массиве воспользуемся линейным поиском, то есть будем просматривать весь массив с начала, пока не найдется данная пара или не встретится конец строки. Построение такого массива имеет сложность O(mk). Далее аналогично каждая пара символов первой строки проверяется на вхождение в построенный массив, что также имеет сложность O(nk). Общая сложность такого алгоритма O((n+m) k), и такие решения набирают 80 баллов.

Приведём пример такого решения на языке Паскаль (80 баллов), использующего массив для хранения пар.

```
var s1, s2, pair: string;
    i, j, ans, npairs: longint;
    pairs: array[1..26 * 26] of string;
begin
    readln(s1);
    readln(s2);
    npairs := 0;
    ans := 0;
    for i := 1 to length(s2) - 1 do
    begin
        pair := copy(s2, i, 2);
        j := 1;
        while (j <= npairs) and (pair <> pairs[j]) do
            j := j + 1;
        if j > npairs then
        begin
            npairs := npairs + 1;
            pairs[npairs] := pair
        end
    end;
    for i := 1 to length(s1) - 1 do
    begin
        pair := copy(s1, i, 2);
        j := 1;
        while (j <= npairs) and (pair <> pairs[j]) do
            j := j + 1;
        if j <= npairs then
            ans := ans + 1
    end;
    writeln(ans)
end.
```

Аналогичное решение на языке Python (80 баллов) гораздо короче, так как линейный поиск в списке можно сделать стандартной операций `in`:

```
s1 = input()
s2 = input()
```

```
pairs = []
ans = 0
for i in range(len(s2) - 1):
   pair = s2[i:i + 2]
   if pair not in pairs:
       pairs.append(pair)
for i in range(len(s1) - 1):
   pair = s1[i:i + 2]
   if pair in pairs:
       ans += 1
print(ans)
```

Это решение легко превращается в решение, набирающее 100 баллов, если заменить структуру данных, в которой хранятся все пары строки `s2` на `set`. Пример решения на языке Python (100 баллов), использующего `set` для хранения пар.

```
s1 = input()
s2 = input()
pairs = set()
for i in range(len(s2) - 1):
    pairs.add(s2[i:i + 2])
ans = 0
for i in range(len(s1) - 1):
    if s1[i: i + 2] in pairs:
        ans += 1
print(ans)
```

`set` – это структура данных «множество», в которой можно хранить уникальные элементы с возможностью быстрого добавления, удаления и поиска элементов. В языке Python структура `set` реализована при помощи технологии хеширования, поэтому все операции со структурой `set` выполняются за O(1) в среднем. Поэтому сложность такого решения O($n + m$). Похожая структура данных есть и в библиотеке STL языка C++.

В языке Паскаль структуры данных set нет (кроме как в современных реализациях PascalABC.Net можно использовать структуру set of string), но можно самостоятельно реализовать требуемую структуру, используя идею битового массива. Для этого создадим двумерный массив `pairs: array['A'..'Z', 'A'..'Z'] of boolean`, в котором будем записывать значение True для тех пар символов, которые встречаются во второй строке. Каждая пара символов соответствует элементу массива, первый индекс которого является первым символом пары, второй индекс – вторым символом пары. Например, если во второй строке встретилась пара 'AB', то необходимо выполнить пометку `pairs['A', 'B'] := True`. Тем самым добавление одной пары символов в данное множество, а также проверка, входит ли данная пара в множество, будет выполняться за O(1).

Пример решения на языке Pascal (100 баллов) с использованием битового массива для хранения пар:

```
var s1, s2: string;
    i, ans: longint;
    pairs: array['A'..'Z', 'A'..'Z'] of boolean;
begin
    readln(s1);
    readln(s2);
    ans := 0;
    for i := 1 to length(s2) - 1 do
        pairs[s2[i], s2[i + 1]] := True;
    for i := 1 to length(s1) - 1 do
        if pairs[s1[i], s1[i + 1]] then
```

```
            ans := ans + 1;
    writeln(ans)
end.
```

Также отметим, что в компиляторе Free Pascal по умолчанию для хранения строк используется тип `ShortString`, в котором длина строки ограничена 255 символами. Для того, чтобы решение на языке Free Pascal набрало больше 60 баллов, необходимо использовать тип данных `AnsiString`, или включить в текст программы опцию компилятора `{$H+}`.

Задача 4. Танец

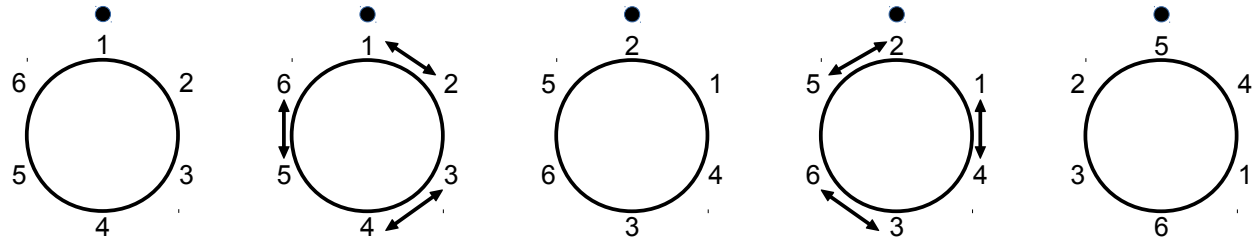
Для исполнения большого танца в круг выстроилось N танцоров (N чётное). Пронумеруем танцоров числами от 1 до N начиная от подиума по часовой стрелке. На каждом шаге танца танцоры разбиваются на пары (пару образуют два соседних по кругу танцора), и танцоры в паре меняются местам, причём на первом и всех последующих нечётных шагах танцор, стоящий в начале круга, образует пару с танцором, стоящим рядом с ним по часовой стрелке. Также пару образуют два танцора, следующие за ними по часовой стрелке, и т. д. На втором шаге и всех шагах с чётными номерами танцор, стоящий в начале круга, образует пару с танцором, стоящим рядом с ним против часовой стрелки. Два танцора, следующие за ними против часовой стрелки, также образуют пару и т. д.

На рисунке изображена начальная расстановка для $N = 6$ танцоров и два следующих шага танца. Расположение подиума отмечено точкой.

Определите, кто будет стоять рядом с танцором номер P через K шагов.

Программа получает на вход три целых числа N, P, K, записанные в отдельных строках. Первое число N – количество танцоров в кругу, N чётное. Второе число P – номер танцора, $1 \le P \le N$. Третье число K – количество сделанных шагов после начала танца, $1 \le K$. Максимальные значения для N и K приведены в разделе «Система оценивания».

Программа должна вывести два целых числа в порядке возрастания – номера танцоров, которые будут стоять рядом с танцором номер P после K шагов танца.



Начальная расстановка | После первого шага | После второго шага

Пример входных и выходных данных

Ввод	Вывод	Примечание
6 5 2	2 4	Рисунок выше соответствует этому примеру.

Система оценивания

Каждый тест к этой задаче (кроме теста из условия) оценивается в 5 баллов. В таблице приведены ограничения на N и K для всех тестов.

Номера тестов	Количество баллов за все тесты группы	Ограничение на N	Ограничение на K
1	0	Пример из условия	
2–7	30	$N \le 100$	$K \le 100$
8–9	10	$N \le 100$	$K \le 10^9$
10–13	20	$N \le 10^5$	$K \le 10^5$
14–15	10	$N \le 10^5$	$K \le 10^9$
16–21	30	$N \le 10^9$	$K \le 10^9$

Решение

Даже по приведённой выше таблице видно, что в этой задаче есть много различных вариантов решений, отличающихся эффективностью, способных набрать различное число баллов в зависимости от ограничений.

Самое простое решение задачи – моделирование перемещений всех танцоров. Расположение танцоров сохраняется в массиве, на каждом шаге соседние элементы массива меняются местами, этот процесс повторяется K раз. Это моделирование имеет сложность O(NK), проходит тесты 2–7 и набирает 30 баллов. Приведём пример такого решения на языке Python (30 баллов).

```
n = int(input())
p = int(input())
k = int(input())
a = [i + 1 for i in range(n)]
for step in range(1, k + 1):
    if step % 2 == 1:
        for i in range(0, n, 2):
            a[i], a[i + 1] = a[i + 1], a[i]
    else:
        for i in range(0, n, 2):
            a[i], a[i - 1] = a[i - 1], a[i]
i = a.index(p)
ans1 = a[(i - 1) % n]
ans2 = a[(i + 1) % n]
print(min(ans1, ans2), max(ans1, ans2))
```

Прокомментируем это решение. Элементы списка в Python индексируются с 0, поэтому мы заполняем список значениями `a[i] = i + 1`. Далее в зависимости от чётности номера шага запускается либо цикл обмена значения `a[i]` с `a[i + 1]`, либо с `a[i - 1]`. Здесь нужно аккуратно обработать случай, когда на чётном шаге элемент `a[0]` меняется с `a[n - 1]`. Это можно сделать либо отдельной командой, либо, как в приведённом решении, используется особенность языка Python, в которой элементы списка также индексируются с конца отрицательными числами: -1, -2, -3 и т.д. То есть элемент с индексом -1 является последним элементом списка, поэтому можно этот случай отдельно не рассматривать.

После окончания моделирования найдём в полученном списке элемент, который равен P, сделаем это при помощи стандартного метода `index` в списке. Далее рассмотрим его соседей – элементы с индексами $i + 1$ и $i - 1$. Но при этом можно выйти за границы списка, если i окажется крайним значением 0 или $n - 1$. Поэтому возьмем остаток от деления

значений $i + 1$ и $i - 1$ на n. Здесь нужно быть аккуратным, при реализации решения на языках C++, Паскаль и других – остаток от деления числа -1 на n в этих языках будет равен -1, что приведёт к ошибке. Поэтому вместо взятия остатка от деления можно сделать две проверки на случай равенства значения i нулю или $n - 1$. Также нам нужно вывести два значения в порядке возрастания, поэтому сначала выведем наименьшее из двух чисел, затем наибольшее.

Чтобы написать решение, набирающее большее число баллов, необходимо изучить структуру расположения танцоров после каждого шага танца. Сделаем это для $N = 8$ танцоров и запишем результат в виде таблицы.

Шаг танца	Порядок танцоров							
Начало	1	2	3	4	5	6	7	8
1	2	1	4	3	6	5	8	7
2	7	4	1	6	3	8	5	2
3	4	7	6	1	8	3	2	5
4	5	6	7	8	1	2	3	4
5	6	5	8	7	2	1	4	3
6	3	8	5	2	7	4	1	6
7	8	3	2	5	4	7	6	1
8	1	2	3	4	5	6	7	8

Можно заметить, что танцоры с нечётными номерами (1, 3, 5, 7) на каждом шаге движутся вправо, а танцоры с чётными номерами (2, 4, 6, 8) всегда движутся влево. Поэтому ровно через N шагов каждый танцор возвращается в исходное место, и дальнейшие передвижения начинают повторяться.

Поэтому если количество выполненных шагов k заменить на остаток от деления k на n, то решение станет быстрее. Для этого можно в предыдущее решение добавить одну строчку `k = k % n` сразу после считывания k. Это решение проходит тесты 2–9 и набирает 40 баллов. Сложность такого решения уже не зависит от величины K и составляет $O(N \min(K, N))$. Также можно взять остаток от деления не на n, а на $n / 2$ – через $n / 2$ шагов взаимное расположение танцоров будет таким же, как в начале, но смещённым на половину круга.

В тестах 10–13 решение сложности $O(NK)$ уже не уложится в ограничение по времени в 1 секунду. В этом случае можно моделировать перемещения не всех танцоров, а только следить за номерами соседей данного танцора P. До начала танца соседями танцора P были $P - 1$ и $P + 1$. На каждом следующем шаге танца номера соседей увеличиваются на 2 если P нечётное или уменьшаются на 2, если P чётное. Также нужно учитывать, что номера танцоров могут быть только числами от 1 до N, что можно сделать при помощи отдельных проверок: если номер танцора стал больше N, то вычтем из номера N, а если стал меньше 1, то прибавим к номеру N. В конце также необходимо вывести сначала наименьшее, потом наибольшее из двух номеров.

Пример решения на языке Python (50 баллов, проходит тесты 2–7 и 10–13).

```
n = int(input())
p = int(input())
k = int(input())
ans1 = p - 1
if ans1 < 1:
    ans1 += n
ans2 = p + 1
if ans2 > n:
```

```
    ans2 -= n
for step in range(k):
    if p % 2 == 1:
        ans1 += 2
        ans2 += 2
        if ans1 > n:
            ans1 -= n
        if ans2 > n:
            ans2 -= n
    else:
        ans1 -= 2
        ans2 -= 2
        if ans1 < 1:
            ans1 += n
        if ans2 < 1:
            ans2 += n
print(min(ans1, ans2), max(ans1, ans2))
```

Сложность этого решения O(*K*). Если в этом решении также добавить присваивание `k = k % n`, то получится решение сложности O(min(*K*, *N*)), которое проходит тесты 2–15 и набирает 70 баллов.

Для того, чтобы получить решение на полный балл, нужно заменить цикл длины *K*, в котором номера соседних танцоров увеличиваются или уменьшаются на 2, на одно увеличение или уменьшение номеров танцоров на 2 *K*. После этого нужно взять остаток от деления номеров танцоров на *N* и учесть случай, когда после взятия остатка получилось число 0 (в этом случае нужно заменить номер танцора на *N*).

Пример решения на языке Python (100 баллов).

```
n = int(input())
p = int(input())
k = int(input())
ans1 = p - 1
ans2 = p + 1
if p % 2 == 1:
    ans1 += 2 * k
    ans2 += 2 * k
else:
    ans1 -= 2 * k
    ans2 -= 2 * k
ans1 %= n
ans2 %= n
if ans1 == 0:
    ans1 = n
if ans2 == 0:
    ans2 = n
print(min(ans1, ans2), max(ans1, ans2))
```

При реализации такого решения на языках Pascal или C++ нужно учитывать, что при взятии остатка от деления отрицательного числа на *N* результат будет отрицательным. Поэтому на этих языках остаток от деления нужно брать следующим образом:

```
ans1 = (ans1 % N + N) % N
```

Задача 5. Распродажа

В магазине проходит новогодняя распродажа – цены всех товаров снижены на 25 %. Оказалось, что первоначально все цены делились на 4, поэтому после снижения цен все цены

также выражаются целым числом.

Товаровед вечером перед распродажей снял ценники со всех товаров и напечатал для каждого товара ещё один ценник со сниженной ценой. Он оставил все ценники на столе, рассчитывая утром их развесить. Но, придя утром в магазин, он обнаружил, что уборщица смешала все ценники вместе, и теперь ему нужно отделить старые ценники от новых. Помогите ему решить эту задачу.

Первая строка входных данных содержит общее количество ценников N, $2 \le N \le 10^5$, N – чётное число. Следующие N строк содержат целые положительные числа, не превосходящие 10^9, идущие в порядке неубывания по одному в строке – числа, записанные на всех ценниках (как старых, так и новых). Гарантируется, что входные данные корректны, то есть решение существует.

Программа должна вывести $N / 2$ целых чисел в порядке неубывания – стоимости товаров после понижения цен.

Пример входных и выходных данных

Ввод	**Вывод**	**Примечание**
6 30 40 42 45 56 60	30 42 45	До распродажи цены товаров были 40, 56, 60, после снижения цены на эти товары стали равны 30, 42, 45.

Система оценивания

Решение, правильно работающее только для случаев, когда $N \le 100$, а значения всех цен (старых и новых) не превосходят 1000 и различны, будет оцениваться в 30 баллов.

Решение, правильно работающее только для случаев, когда $N \le 100$, а значения всех цен не превосходят 1000, будет оцениваться в 60 баллов.

Решение

Будем идти по списку цен от наименьшей цены к наибольшей. Очередное встреченное значение цены (обозначим его `price`) будем считать ценой какого-то товара после снижения цен, выведем его. Теперь посчитаем стоимость этого товара до понижения цен, оно равно `price / 3 * 4`. Нужно найти среди цен соответствующее значение и удалить его из списка. Количество баллов, которое набирает такое решение, зависит от того, как реализовано удаление из списка и от используемых структур данных.

Если хранить все цены в массиве (или в `list` в языке Python), то удаление элемента из массива приводит к сдвигу всей части массива, то есть выполняется за O(N). Можно не удалять элементы из массива, а вместо этого присваивать элементу массива, в котором хранится найденное значение цены товара до понижения значение 0. Тогда во внешнем цикле будем пропускать нулевые значения элементов массива – они соответствуют ценам товаров до понижения, то есть они были «удалены» ранее. Приведём пример такого решения на языке Python (60 баллов).

```
n = int(input())
prices = [int(input()) for i in range(n)]
for i in range(n):
    if prices[i] != 0:
        price = prices[i]
        print(price)
        price = price // 3 * 4
        j = 0
```

```
        while prices[j] != price:
            j += 1
        prices[j] = 0
```

В этом решении для поиска цены товара до повышения используется линейный поиск по всем элементам массива, общая сложность решения O(N^2).

Сложность решения можно уменьшить до O(N), если воспользоваться методом «двух указателей». Заметим, что мы перебираем цены товаров в порядке неубывания, поэтому цены товаров до понижения также будут следовать в порядке неубывания. Это означает, что поиск цены товара до понижения следует осуществлять не с начала массива, а с того места, где остановился поиск предыдущей цены товара до понижения. То есть значение переменной j нужно не сбрасывать на 0 перед каждым началом цикла, а продолжать поиск с того значения j, на котором остановился предыдущий поиск. Для исправления решения нужна минимальная правка – инициализацию j = 0 нужно перенести в начало программы.

Решение на языке Python (100 баллов).

```
n = int(input())
prices = [int(input()) for i in range(n)]
j = 0
for i in range(n):
    if prices[i] != 0:
        price = prices[i]
        print(price)
        price = price // 3 * 4
        while prices[j] != price:
            j += 1
        prices[j] = 0
```

Возможны и другие варианты решения с использованием эффективных структур данных, позволяющих быстро искать и удалять элементы. Структура `set` для решения на полный балл не подойдёт, так как в `set` все хранимые значения различные, а по условию задачи возможны и одинаковые решения. Можно использовать структуру данных «словарь», также называемая «ассоциативный массив» (`dict` в языке Python, `map` в языке C++), в которой каждому ключу соответствует некоторое значение. Ключами будут цены товаров (как до понижения, так и после понижения), а значениями – сколько раз встречается такая цена. После считывания данных построим такой словарь `count`, затем будем проходить по списку цен от наименьшей к наибольшей. Для каждого значения цены `price` проверяем соответствующее ей значение в словаре. Если это значение не равно 0, то выведем эту цену и удалим две стоимости (после понижения и соответствующую ей стоимости до понижения), что заключается в уменьшении на 1 двух элементов словаря: `count[price]` и `count[price / 3 * 4]`.

Решение на языке Python (100 баллов).

```
n = int(input())
prices = [int(input()) for i in range(n)]
count = dict()
for price in prices:
    if price in count:
        count[price] += 1
    else:
        count[price] = 1
for price in prices:
    if count[price] > 0:
        print(price)
        count[price] -= 1
        count[price // 3 * 4] -= 1
```

В языке C++ вместо словаря можно использовать структуру данных multiset, которая позволяет хранить повторяющиеся значения. В этом случае нужно просто все считанные числа добавить в структуру `multiset`, затем пройти по структуре, удаляя цены товаров до понижения, которые соответствуют найденным ценам товара после понижения. В приведённом ниже решении из структуры `multiset` удаляется наименьшее значение (итератор на это значение возвращает метод `begin()`) и соответствующее ему значение цены до повышения (в строке `prices.erase(prices.find(price / 3 * 4))`).

Решение на языке C++ (100 баллов).

```
#include <iostream>
#include <set>
using namespace std;
int main() {
    int n;
    cin >> n;
    multiset<int> prices;
    for (int i = 0; i != n; ++i) {
        int price;
        cin >> price;
        prices.insert(price);
    }
    while (!prices.empty()) {
        int price = *prices.begin();
        cout << price << '\n';
        prices.erase(prices.begin());
        prices.erase(prices.find(price / 3 * 4));
    }
   return 0;
}
```