Задача A.

Пусть с тех пор, как Вите было n лет, а его брату m, прошло x лет.

Решение 1, математическое. Тогда сегодня Вите $n+x$ лет, а его брату $m+x$. По условию $k(n+x)=m+x$, значит $kn+kx=m+x$, то есть $(k-1)x=m-kn$. Таким образом $x=(m-kn)/(k-1)$. Если $m-kn$ не делится на $k-1$, или $m-kn \leqslant 0$, то описанной ситуации произойти не могло, следует вывести -1. Иначе выводим ответ: $n+x$.

Приведем код такого решения на языке Python.

```
n = int(input())
m = int(input())
k = int(input())

if m <= k * n or (m - k * n) % (k - 1) != 0:
    print(-1)
else:
    x = (m - k * n) // (k - 1)
    print(n + x)
```

Решение 2, программистское. Переберем значение x, начиная с 1. Для очередного вариант значения x проверим, выполняется ли равенство $k(n+x)=m+x$. Если это равенство выполнено, то выводим $n+x$. Осталось понять, до какой верхней границы имеет смысл перебирать x. Заметим, что если $x>m$, то $k(n+x) \geqslant 2(n+x)>2x>m+x$, поэтому равенство $k(n+x)=m+x$ выполниться не может. Таким образом, достаточно перебрать x от 1 до m. Если ни один x не подошёл, выводим -1.

Приведем код такого решения на языке Python.

```
n = int(input())
m = int(input())
k = int(input())

for x in range(1, m + 1):
    if k * (n + x) == m + x:
        print(n + x)
        exit()

print(-1)
```

Задача B.

Заметим, что в одном наборе вложенных друг в друга матрёшек не может быть двух матрёшек одинакового размера. Действительно, в каждую матрёшку непосредственно внутрь неё вкладывается меньшая по размеру, а значит внутри каждой все матрёшки строго меньшие её по размеру, снаружи — строго большие. Таким образом, ответ не может быть больше количества различных размеров матрёшек. С другой стороны, если взять по одной матрёшке каждого имеющегося в наличии размера, то их все можно будет вложить друг в друга, как и требуется по условию.

Итак, ответ — количество различных размеров матрёшек, которые есть у Маши. Осталось понять, как определить это количество.

Решение 1. Сначала отсортируем массив размеров матрёшек. Подходит любая сортировка со временем работы $O(n^2)$ и быстрее, например сортировка пузырьком или выбором, а также встроенная в стандартную библиотеку. Теперь, просматривая массив размеров, мы обнаруживаем новый размер матрёшки в первом элементе, а также когда видим значение, не равное предыдущему.

Код программы на паскале с использованием сортировки пузырьком.

```
program matryoska;

var
    a: array [1..1000] of longint;
    i, j, n, t, ans: longint;
begin
    read(n);
    for i := 1 to n do
        read(a[i]);
    for i := 1 to n do
        for j := 1 to n - i do
            if a[j] > a[j + 1] then begin
                t := a[j]; a[j] := a[j + 1]; a[j + 1] := t;
            end;

    ans := 0;
    for i := 1 to n do
        if (i = 1) or (a[i] <> a[i - 1]) then
            inc(ans);

    writeln(ans);
end.
```

Код программы на языке Python с использованием встроенной сортировки.

```
n = int(input());
a = list(map(int, input().split()))
a.sort()

ans = 0
for i in range(n):
    if i == 0 or a[i] != a[i - 1]:
        ans += 1
print(ans)
```

Решение 2. Также количество различных элементов можно посчитать следующим образом. Заведем массив *cnt* размером 10 000, изначально заполненный нулями. При считывании массива размеров матрёшек, будем увеличивать на один элемент с номером a_i в массиве *cnt*. Тогда в конце значение $cnt[i]$ будет содержать число различных матрёшек размера i. Осталось посчитать количество ненулевых элементов массива *cnt*.

Приведем пример кода на паскале.

```
program matryoska2;

const
    MAX = 10000;

var
    i, k, n, ans: longint;
    cnt: array [1..MAX] of longint;

begin
    readln(n);
    for i := 1 to n do
    begin
        read(k);
        inc(cnt[k]);
    end;

    ans := 0;
    for i := 1 to MAX do
        if cnt[i] > 0 then inc(ans);
    writeln(ans);
end.
```

Задача C.

Попробуем простое решение: найдем все числа Фибоначчи от первого до R-го, сложим их в массив и посчитаем, сколько из чисел в интервале от L до R делятся на 3. Напишем такое решение, например, на паскале.

```
program fib3;

var
    fib: array [1..100000] of longint;
    L, R, i, ans: longint;

begin
    read(L, R);

    fib[1] := 1;
    fib[2] := 2;
    for i := 3 to R do
        fib[i] := fib[i - 1] + fib[i - 2];

    ans := 0;
    for i := L to R do
        if fib[i] mod 3 = 0 then
            ans := ans + 1;
    writeln(ans);
end.
```

Программа работает на небольших тестах, но выдает неверный ответ на больших тестах (например, на 12 тесте жюри, если отправить её на проверку). В чём же дело? Посмотрев на содержимое массива `fib`, видно, что числа Фибоначчи растут очень быстро и происходит переполнение типа `longint`. Даже использование 64-битного типа не решает проблему, числа растут слишком быстро.

Что если попробовать написать решение на языке Python, где ограничений по размеру чисел, которые может содержать целочисленный тип, нет? Попробуем.

```
L = int(input())
R = int(input())

fib = [0] * (R + 1)
fib[1] = 1
fib[2] = 2
for i in range(3, R + 1):
    fib[i] = fib[i - 1] + fib[i - 2]

ans = 0
for i in range(L, R + 1):
    if fib[i] % 3 == 0:
        ans += 1
print(ans)
```

К сожалению, эта программа тоже не проходит тесты жюри, на этот раз на больших тестах превышается ограничение по времени — получаются слишком длинные числа, для которых сложение и взятие остатка по модулю 3 работает слишком медленно.

Таким образом, вычислить все числа Фибоначчи с номерами от L до R и для каждого из них проверить, делится ли оно на три, не получается. Нужно применить какой-то другой подход.

Арифметика остатков. Первый подход основывается на следующей математической идее. Вместо того, чтобы считать сами числа Фибоначчи, будем считать их остатки от деления на три. Тогда переполнения не произойдет, и числа длинными становиться не будут, поэтому после соответствующей модификации любая из двух предыдущих программ легко решит задачу. Исправим, например, программу на паскале. Получим следующее решение.

```
program fib3;

var
    fib: array [1..100000] of longint;
    L, R, i, ans: longint;

begin
    read(L, R);

    fib[1] := 1;
    fib[2] := 2;
    for i := 3 to R do
        fib[i] := (fib[i - 1] + fib[i - 2]) mod 3;

    ans := 0;
    for i := L to R do
        if fib[i] = 0 then
            ans := ans + 1;
    writeln(ans);
end.
```

Поиск закономерности. Попробуем зайти с другой стороны. Выпишем несколько первых чисел Фибоначчи и выделим те из них, которые делятся на 3. Получим последовательность 1, 2, **3**, 5, 8, 13, **21**, 34, 55, 89, **144**, 233, ...

Легко видеть, что, начиная с третьего, каждое четвертое число делится на три. Таким образом, число Фибоначчи делится на 3, если его номер даёт остаток 3 по модулю 4.

Докажем это. Последовательность чисел Фибоначчи по модулю 3 начинается так: 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, ... Заметим, что $F_1 \equiv F_9 \pmod 3$ и $F_2 \equiv F_{10} \pmod 3$. Значит дальше остатки по модулю 3 будут повторяться с периодом 8. Среди первых 8 чисел на 3 делятся только числа Фибоначчи с номерами 3 и 7.

Итак, можно просто посчитать количество чисел, которые дают остаток 3 по модулю 4 на отрезке от L до R.

```
L = int(input())
R = int(input())

ans = 0
for i in range(L, R + 1):
    if i % 4 == 3:
        ans += 1
print(ans)
```

Наконец, воспользуемся поводом рассказать об еще одном полезном приёме, который можно (хотя, как мы видели, и не обязательно) применить в этой задаче.

Пусть нам требуется посчитать чисел с каким-либо свойством на отрезке от L до R. Тогда можно посчитать количество таких чисел на отрезке от 1 до R и на отрезке от 1 до $L - 1$, и вычесть из первого значения второе. В случае этой задачи получаем такое, совсем простое, решение.

```
L = int(input())
R = int(input())

print((R + 1) // 4 - L // 4)
```

Задача D.

Чтобы решить эту задачу, полезно сначала посмотреть, что происходит в привычной нам десятичной системе счисления. Выпишем несколько первых чисел, которые заканчиваются хотя бы на два нуля: 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, ... Легко понять, что начало этого ряда совпадает с началом натурального ряда, просто в конец каждого числа дописано по два нуля. Таким образом, i-е такое число просто равно $100i$.

Обобщим теперь это рассуждение на произвольное основание системы счисления и произвольное число нулей. Так как искомое число заканчивается на n нулей в k-ичной системе счисления, оно делится на k^n. Поэтому ответом является i-е натуральное число, делящееся на k^n, то есть $i \cdot k^n$.

Решение задачи на языке Python.

```
k = int(input())
n = int(input())
i = int(input())

print(i * k**n)
```

Задача E.

Это пример «задачи на реализацию», где в условии написано, что требуется сделать, необходимо лишь реализовать описанное.

Для каждого ученика будем хранить структуру, состоящую из трех чисел: номера ученика, номер его класса и его суммарный балл по всем отборочным турам. После этого с помощью написанной вручную или встроенной в язык сортировки отсортируем массив таких структур в порядке убывания суммарного балла. Отберем первую команду, взяв в неё четырех лучших, затем пройдём по ученикам не из 11 класса и отберём четырёх лучших, которые не попали в первую команду. Заметим, что это всегда возможно, так как по условию гарантируется, что есть хотя бы 8 учеников не из 11 класса.

Не забудем перед выводом отсортировать номера участников внутри каждой сборной в порядке возрастания.

Приведем пример программы на языке Python.

```
n = int(input())
alls = []
no11 = []
for i in range(n):
    x = list(map(int, input().split()))
    p = (sum(x[1:]), x[0], i + 1)
    alls.append(p)
    if p[1] != 11:
        no11.append(p)

alls.sort(reverse=True)
no11.sort(reverse=True)

res = [[], []]
for i in range(4):
    res[0].append(alls[i][2])

i = 0
while len(res[1]) < 4:
    if not no11[i][2] in res[0]:
        res[1].append(no11[i][2])
    i += 1

res[0].sort()
res[1].sort()

print('␣'.join(map(str, res[0])))
print('␣'.join(map(str, res[1])))
```