位移寄存器（registers）

工程师 Christopher 在开发一款新的计算机处理器。

这个处理器可以访问 m 个不同的 b 位存储单元（本题中 $m = 100$ 且 $b = 2000$）。它们被称作**寄存器**，编号从 0 到 $m - 1$。我们把这些寄存器记为 $r[0], r[1], \ldots, r[m-1]$。每个寄存器都是 b 个比特的数组，这些比特从 0 （最右的比特） 到 $b - 1$ （最左的比特）编号。对所有的 i $(0 \leq i \leq m - 1)$ 和 j $(0 \leq j \leq b - 1)$，我们将寄存器 i 的第 j 个比特记为 $r[i][j]$。

对所有的比特序列 $d_0, d_1, \ldots, d_{l-1}$ （具有某个长度 l），该序列的**整数值**等于 $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$。我们说**存储在某个寄存器中的整数值**就是寄存器中比特序列的整数值，也就是说，该整数值为 $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$。

该处理器有 9 种类型的**指令**，可以用来修改寄存器中的比特。每条指令操作一个或多个寄存器，并将其输出存储到其中的一个寄存器。下面我们用 $x := y$ 表示一个修改 x 的值并将其变成 y 的操作。每种类型的指令所做的操作描述如下：

- $move(t, y)$：将寄存器 y 中的比特数组拷贝到寄存器 t。对所有的 j $(0 \leq j \leq b - 1)$，设置 $r[t][j] := r[y][j]$。
- $store(t, v)$：设置寄存器 t 等于 v，这里 v 是某个 b 个比特的数组。对于所有的 j $(0 \leq j \leq b - 1)$，设置 $r[t][j] := v[j]$。
- $and(t, x, y)$：取寄存器 x 和 y 的按位与，并将结果存到寄存器 t 中。对于所有的 j $(0 \leq j \leq b - 1)$，如果 $r[x][j]$ 和 $r[y][j]$**同时**为 1 则设置 $r[t][j] := 1$，否则设置 $r[t][j] := 0$。
- $or(t, x, y)$：取寄存器 x 和 y 的按位或，并将结果存到寄存器 t 中。对于所有的 j $(0 \leq j \leq b - 1)$，如果 $r[x][j]$ 和 $r[y][j]$**至少有一个**为 1 则设置 $r[t][j] := 1$，否则设置 $r[t][j] := 0$。
- $xor(t, x, y)$：取寄存器 x 和 y 的按位异或，并将结果存到寄存器 t 中。对于所有的 j $(0 \leq j \leq b - 1)$，如果 $r[x][j]$ 和 $r[y][j]$**恰好有一个**为 1 则设置 $r[t][j] := 1$，否则设置 $r[t][j] := 0$。
- $not(t, x)$：取寄存器 x 的按位非，并将结果存到寄存器 t 中。对于所有的 j $(0 \leq j \leq b - 1)$，设置 $r[t][j] := 1 - r[x][j]$。
- $left(t, x, p)$：左移寄存器 x 中的所有比特 p 位，并将结果存到寄存器 t 中。将寄存器 x 中的比特左移 p 位的结果，是一个包含 b 个比特的数组 v。对于所有的 j $(0 \leq j \leq b - 1)$，如果

$j \geq p$ 则 $v[j] = r[x][j-p]$，否则 $v[j] = 0$。对所有的 j $(0 \leq j \leq b-1)$，设置 $r[t][j] := v[j]$。

- $right(t, x, p)$：右移寄存器 x 中的所有比特 p 位，并将结果存到寄存器 t 中。将寄存器 x 中的比特右移 p 位的结果，是一个包含 b 个比特的数组 v。对于所有的 j $(0 \leq j \leq b-1)$，如果 $j \leq b-1-p$ 则 $v[j] = r[x][j+p]$，否则 $v[j] = 0$。对所有的 j $(0 \leq j \leq b-1)$，设置 $r[t][j] := v[j]$。

- $add(t, x, y)$: 将寄存器 x 和 y 中的整数值加起来，并将结果存到寄存器 t 中。加法是在模 2^b 下做的。正式一些来说，设 X 是操作前存在寄存器 x 中的整数值，而 Y 是操作前存在寄存器 y 中的整数值。设 T 为操作后存在寄存器 t 中的整数值。如果 $X + Y < 2^b$，设置 t 中的比特使得 $T = X + Y$。否则，设置 t 中的比特使得 $T = X + Y - 2^b$。

Christopher 希望你用这个新处理器解决两种任务。任务的类型用整数 s 来表示。对所有类型的任务，你需要创建一个**程序**，其为上文所定义的指令构成的序列。

程序的**输入**包括 n 个整数 $a[0], a[1], \ldots, a[n-1]$，而每个整数都有 k 个比特，也就是说，$a[i] < 2^k$ $(0 \leq i \leq n-1)$。在程序执行前，输入的所有的数都依次存储在寄存器 0 中，使得对所有的 i $(0 \leq i \leq n-1)$，k 比特序列 $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ 的整数值等于 $a[i]$。注意 $n \cdot k \leq b$。寄存器 0 中所有其他的比特（其下标在 $n \cdot k$ 和 $b-1$ 之间，包括 $n \cdot k$ 和 $b-1$），以及其他所有寄存器中的所有比特，都初始化为 0。

执行某个程序就是按序执行其所包含的指令。在最后一条指令执行完毕后，程序的**输出**将根据寄存器 0 中比特最终的值计算出来。具体来说，输出是 n 个整数 $c[0], c[1], \ldots, c[n-1]$ 的序列，这里对所有 i $(0 \leq i \leq n-1)$ 来说，$c[i]$ 都是寄存器 0 中比特 $i \cdot k$ 到 $(i+1) \cdot k - 1$ 所构成的序列的整数值。注意，在程序运行结束后，寄存器 0 中其余的比特（下标不小于 $n \cdot k$），以及其他寄存器中的所有比特，可能是任意值。

- 第一个任务 $(s = 0)$ 是要找出输入整数 $a[0], a[1], \ldots, a[n-1]$ 中的最小值。具体来说，$c[0]$ 必须是 $a[0], a[1], \ldots, a[n-1]$ 中的最小值。$c[1], c[2], \ldots, c[n-1]$ 的值可以是任意的。

- 第二个任务 $(s = 1)$ 是要将输入整数 $a[0], a[1], \ldots, a[n-1]$ 进行非降序排序。具体来说，对于所有的 i $(0 \leq i \leq n-1)$，$c[i]$ 应当等于 $a[0], a[1], \ldots, a[n-1]$ 中第 $1+i$ 小的整数（也就是说，$c[0]$ 是输入整数中的最小整数）。

请帮 Christopher 写一下解决这些任务的程序。每个程序至多只能包含 q 条指令。

实现细节

你要实现如下函数:

```
void construct_instructions(int s, int n, int k, int q)
```

- s：任务类型。
- n：输入中的整数的数量。

- k：输入中的每个整数的比特数。
- q：允许的最大的指令数。
- 该函数将被恰好调用一次，并应当为所要解决的任务创建一个指令序列。

该函数应当调用以下函数中的一或多个，以创建指令序列：

```
void append_move(int t, int y)
void append_store(int t, bool[] v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
```

- 每个函数分别往程序追加一条 $move(t, y)$、 $store(t, v)$、 $and(t, x, y)$、 $or(t, x, y)$、 $xor(t, x, y)$、 $not(t, x)$、 $left(t, x, p)$、 $right(t, x, p)$ 或 $add(t, x, y)$ 指令。
- 对于所有相关的指令， t、 x、 y 必须至少为 0 且至多为 $m - 1$。
- 对于所有相关的指令， t、 x、 y 不必是两两之间不同的。
- 对于指令 $left$ 和 $right$， p 必须至少为 0 且至多为 b。
- 对于指令 $store$， v 的长度必须为 b。

你还可以调用以下函数，以帮助测试你的答案：

```
void append_print(int t)
```

- 在评测你的答案时，对该函数的所有调用都将被忽略。
- 在评测程序示例中，该函数将往程序追加一个 $print(t)$ 操作。
- 当评测程序示例在执行某个程序过程中遇到一个 $print(t)$ 操作时，它会打印出由寄存器 t 中前 $n \cdot k$ 比特构成的 n 个 k-比特整数（细节可参见“评测程序示例”部分）。
- t 必须满足 $0 \leq t \leq m - 1$。
- 对该函数的任何调用，都不会算到你所创建的指令的数量里面。

在追加最后一条指令之后，`construct_instructions` 应当返回。随后你创建的程序将在一定数量的测试用例上评测，其中每个测试用例给出的输入数据为 n 个 k-比特整数 $a[0], a[1], \ldots, a[n-1]$。如果程序对给定输入数据的输出结果 $c[0], c[1], \ldots, c[n-1]$ 满足如下条件，你的答案就将被视为通过了对应的样例：

- 如果 $s = 0$， $c[0]$ 应当为 $a[0], a[1], \ldots, a[n-1]$ 中的最小值。
- 如果 $s = 1$，对所有 i ($0 \leq i \leq n-1$) 来说， $c[i]$ 应当是 $a[0], a[1], \ldots, a[n-1]$ 中第 $1 + i$ 小的整数。

在评测你的答案时，可能会给出下面的错误信息之一：

- `Invalid index`：在调用某些函数时的参数 t, x 或 y 所给出的寄存器下标是不正确的（可能是负数）。
- `Value to store is not b bits long`：提供给 `append_store` 的 v 的长度不等于 b。
- `Invalid shift value`：提供给 `append_left` 或 `append_right` 的 p 的值不在 0 和 b 之间（包括 0 和 b）.
- `Too many instructions`：你的函数试图追加超过 q 条指令。

例子

例 1

设 $s = 0$，$n = 2$，$k = 1$，$q = 1000$。有两个输入的整数 $a[0]$ 和 $a[1]$，每个都有 $k = 1$ 个比特。在程序执行前，$r[0][0] = a[0]$ 且 $r[0][1] = a[1]$。处理器中的其他所有比特都被设置为 0。在程序中的指令全部执行完毕后，我们想要得到 $c[0] = r[0][0] = \min(a[0], a[1])$，其为 $a[0]$ 和 $a[1]$ 中的最小值。

提供给程序的输入，总共有四种可能情形：

- 情形 1：$a[0] = 0, a[1] = 0$
- 情形 2：$a[0] = 0, a[1] = 1$
- 情形 3：$a[0] = 1, a[1] = 0$
- 情形 4：$a[0] = 1, a[1] = 1$

我们可以注意到，对于所有 4 种情形，$\min(a[0], a[1])$ 等于 $a[0]$ 和 $a[1]$ 的按位与。因此，一种可能的答案是通过做如下调用而创建的程序：

1. `append_move(1, 0)`，追加一条指令，将 $r[0]$ 拷贝到 $r[1]$。
2. `append_right(1, 1, 1)`，追加一条指令，将 $r[1]$ 中的所有比特右移 1 位，接着将结果存回到 $r[1]$ 中。由于每个整数都是 1-比特长的，这将使得 $r[1][0]$ 等于 $a[1]$。
3. `append_and(0, 0, 1)`，追加一条指令，将 $r[0]$ 和 $r[1]$ 做按位与, 接着将结果存到 $r[0]$ 中。在本指令执行完毕后，$r[0][0]$ 被设置成 $r[0][0]$ 和 $r[1][0]$ 的按位与，其值等于 $a[0]$ 和 $a[1]$ 的按位与，也就是所要求的结果。

例 2

设 $s = 1$，$n = 2$，$k = 1$，$q = 1000$。与前面的例子一样，这里程序的输入数据也只有 4 种可能的情形。对于所有 4 种情形，$\min(a[0], a[1])$ 是 $a[0]$ 和 $a[1]$ 的按位与，而 $\max(a[0], a[1])$ 是 $a[0]$ 和 $a[1]$ 的按位或。一个可能的答案是做如下调用：

1. `append_move(1,0)`
2. `append_right(1,1,1)`
3. `append_and(2,0,1)`
4. `append_or(3,0,1)`
5. `append_left(3,3,1)`

6. `append_or(0,2,3)`

在执行完这些指令后，$c[0] = r[0][0]$ 存有 $\min(a[0], a[1])$，而 $c[1] = r[0][1]$ 则存有 $\max(a[0], a[1])$，也就对输入数据做好了排序。

约束条件

- $m = 100$
- $b = 2000$
- $0 \leq s \leq 1$
- $2 \leq n \leq 100$
- $1 \leq k \leq 10$
- $q \leq 4000$
- $0 \leq a[i] \leq 2^k - 1$（对于所有 $0 \leq i \leq n - 1$）

子任务

1. （10 分） $s = 0, n = 2, k \leq 2, q = 1000$
2. （11 分） $s = 0, n = 2, k \leq 2, q = 20$
3. （12 分） $s = 0, q = 4000$
4. （25 分） $s = 0, q = 150$
5. （13 分） $s = 1, n \leq 10, q = 4000$
6. （29 分） $s = 1, q = 4000$

评测程序示例

评测程序示例按以下格式读取输入：

- 第 1 行：s n k q

接下来还有若干行，每行描述一个单独的测试用例。每个测试用例将以如下格式给出：

- $a[0]$ $a[1]$ $\ldots$ $a[n-1]$

这样，就描述出了一个包含 n 个整数 $a[0], a[1], ..., a[n-1]$ 的测试用例。在所有测试用例的描述之后，将跟着仅包含 -1 的单独一行。

评测程序示例首先调用 `construct_instructions(s, n, k, q)`。如果该调用违反了在程序说明中的某些限制，评测程序示例将打印出在“实现细节”部分列出的错误信息之一，并且退出。否则，评测程序示例首先依次打印出 `construct_instructions(s, n, k, q)` 所追加的指令。对于 *store* 指令，将把 v 从下标 0 到 $b - 1$ 依次打印出来。

随后，评测程序示例将依次处理测试用例。对于每个测试用例，评测程序示例将在该测试用例中的输入数据上运行所创建的程序。

对于每个 $print(t)$ 操作，设 $d[0], d[1], \ldots, d[n-1]$ 为一个整数序列，使得对于所有 i ($0 \leq i \leq n-1$)，$d[i]$ 为（在执行该操作时）寄存器 t 中比特序列 $i \cdot k$ 到 $(i+1) \cdot k - 1$ 的整数值。评测程序按照如下格式打印出该序列：`register` t: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$。

如果所有指令都被执行完毕，评测程序示例将打印出程序的输出结果。

如果 $s = 0$，评测程序示例在每个测试用例上的输出结果为如下格式：

- $c[0]$。

如果 $s = 1$，评测程序示例在每个测试用例上的输出结果为如下格式：

- $c[0]$ $c[1]$ $\ldots$ $c[n-1]$。

在处理完所有测试用例后，评测程序打印出 `number of instructions:` X，这里 X 是你的程序的指令数量。