Problem D: ARMPIT Computations

The popular ARMPIT processor has a simple architecture that is built around a single register that is 12 bits long. The most significant bit is viewed as being at the left end, with the least significant bit at the right end. The contents of the register are often represented as an unsigned integer written in base 10. The processor supports the following operations:

Instruction	Result
`ORWITH` *¡v¿*	modifies the register by performing a bitwise OR with *¡v¿*
`LSL` *¡amt¿*	bitwise left shifts the register by *¡amt¿* positions
`ROR` *¡amt¿*	bitwise right rotates the register by *¡amt¿* positions
`ADDONE`	increments the register by 1
`NOT`	inverts (complements) each bit in the register
`ADDSHIFT` *¡amt¿*	adds the register to a copy of itself that has been bitwise left shifted by *¡amt¿* positions

In the above table:

- *¡v¿* is an unsigned integer stored in 4 bits; bitwise OR combines this with the 4 least significant bits of the register
- *¡amt¿* is an unsigned integer stored in 3 bits
- a left shift by *¡amt¿* positions moves *¡amt¿* 0's into the right end of the register, and discards the *¡amt¿* bits that "fall off" the left end
- a bitwise rotation by 1 position puts the least significant bit into the most significant bit position, and all other bits shift right by 1 position; a bitwise rotation by *¡amt¿* positions is equivalent to *¡amt¿* successive rotate-by-1 operations
- for operations `ADDONE` and `ADDSHIFT`, addition is performed mod 2^{12}, i.e., with truncation on overflow

Note that with the exception of `ADDSHIFT`, similar operations are found in most CPU instruction sets. Here are examples of the supported operations, assuming that the register stores 6 (0000 0000 0110) before each operation, *¡v¿* is 12 (1100), and *¡amt¿* is 5 (101). For convenience, leading (high-end) zeros in the register are often omitted.

- `ORWITH` makes the register 14, since the OR of binary 0110 and 1100 is 1110.
- `LSL` makes the register 192, since left shifting 0110 by 5 positions gives binary 1100 0000. If, on the other hand, the register initially stored 1111 1111 1111, left shifting by 5 positions would yield 4064 (binary 1111 1110 0000), because the result is truncated to 12 bits.

- `ROR` makes the register 768, since the binary value 0110 becomes $0011\,0000\,0000$ when rotated rightward by 5 positions.
- `ADDONE` makes the register store 7. If the register had initially stored 4095, it would contain 0 after the `ADDONE` operation.
- `NOT` makes the register store 4089 (binary $1111\,1111\,1001$).
- `ADDSHIFT` makes the register store 198, because 6 is added to 192 (see the example for `LSL` to explain the 192).

For each of a given list of 12-bit target values, determine the length of the shortest sequence of ARMPIT instructions that puts that value into the register, under the assumption that the register is initially filled with zeros. For example, for a target value of 769, the shortest sequence contains 3 operations (one possiblity is: `ORWITH` 6, `ROR` 5, `ADDONE`).

Input

The first line of the input contains a positive integer T ($T \leq 200$), representing the number of test cases. Each of the next T lines contains a single integer between 0 and 4095, inclusive, the target value for that test case.

Output

For each of the T test cases, output a line containing a single integer, the minimum number of ARMPIT instructions that must be executed to obtain the target value.

Sample Input	*Sample Output*	*Sample Output, with visualized whitespace*
4 769 0 10 20	3 0 1 2	3\n 0\n 1\n 2\n

Note: ␣ is a space, and \n is a newline character.