JUDGING NOTES

1. In the past, the Judges have issued very few actual clarifications - and those were only for ambiguities. Please read the problem statement and examine the sample test cases carefully before submitting any request for clarifications.

2. The following guidelines apply to handling input/output in programs:
   - All input comes from standard input.
   - All output goes to standard output.
   - Unless the problem explicitly states otherwise, the input for a problem consists of a single test case.
   - Output formatting should follow the sample output in the problem statement, although extra whitespace within reason is acceptable. For example, if you print out a gigabyte of blanks, then the Judges will treat that as Wrong Answer; however, an extra blank at the end of a line or an extra blank line between test cases is acceptable.
   - For problems with floating point output, the Judges will accept a range of answers as correct provided they satisfy the constraints described in the problem statement. These constraints will be specified as an absolute and/or relative tolerance, which will be given.
   - There is no such thing as "Presentation Error" or "Format Error." If you misspell the word "impossible," for example, and the problem requires that word as output, then your submission will be judged as "Wrong Answer."
   - Unless a problem specifically indicates that uppercase or lowercase letters are important, then either will be accepted. For example, "Yes" or "yes" would be treated the same, but "yse" would be judged as "Wrong Answer."

3. Your program may be run on multiple input files. Note that this means that if your program has more than one error (say, Time Limit Exceeded and Wrong Answer), then you can get either error as the judgment. See the *DOMjudge* documentation for details on how the judging works.

4. The time limits for each problem will be specified in the problem statement. For information regarding memory limits, see the separate *Technical Notes*.

5. Input size constraints on test cases will be given as part of the problem statements. If the input contains multiple test cases, then the problem will also state an upper bound on the number of test cases.

6. If you submit a solution that has a Compile Error, then you will be notified of it (just as with any other error). However, Compile Errors do not count toward penalty time.

7. The problem set may include one or more *interactive problems*. In many respects, interactive problems are like any other problem – your program will read from standard input and print results to standard output. However, with an interactive problem, standard input and output of your program are connected to another (judge) program, with which your program must communicate back and forth.

   See the back page for additional notes on interactive problems.

Additional Judging Notes on Interactive Problems

- In most programming environments, program output is buffered to speed up I/O operations. With interactive problems, it is crucial to make sure the output is actually sent from your program and not simply stored in internal buffers. This typically means flushing the output buffers after each line of output (that is, after each *newline* character) as follows:
  - In C (or C++ using `cstdio`), you can use `fflush(stdout)`.
  - A C++ output stream is flushed automatically each time you write the `endl` manipulator. When using other means or if you want to be sure, call `cout.flush()`.
  - In Java and Kotlin, the `System.out` stream has so-called "auto-flush" functionality and its buffer is therefore flushed automatically with each newline character. When using other streams or if you want to be sure, invoke the `flush()` method of the stream.
  - In Python, you can use `sys.stdout.flush()`.

- Interactive problems are judged in a way similar to other problems, but there are some differences:
  - When your program attempts to read data, it will wait until more data are available or until the judge program terminates the input (unless you read in a non-blocking way, which is beyond the scope of these notes). Thus, if your program attempts to read more input than can currently be provided (e.g., because you forgot to flush your previous output, or because of some other reason), then the program will stall indefinitely and your submission will get Time Limit Exceeded.
  - As usual, the judgement given to an incorrect submission is the first error discovered, but this does not always mean exactly the same thing as for traditional problems. For instance, if your submission to a traditional problem is too slow on a particular test case, you would get Time Limit Exceeded on that test case. With an interactive problem, the judgement may be Wrong Answer, if the solution exhibits an incorrect answer before it runs out of time.
  - The time limit for an interactive problem is how much time your submission may spend; the time spent by the judge program is not counted towards this.
  - The judge program may behave in an adversarial way and adapt the input provided to your program based on your previous output, with the intent to discover errors in your algorithm.
  - Because of timing between the interactive judge program and your submission, judgements for incorrect submissions to interactive problems are not necessarily deterministic. We can guarantee the following: a judgement of "Wrong Answer" means that your submission produced incorrect output, and a judgement of "Run Time Error" means that your submission returned a non-zero exit code. If your submission does both, you may receive either judgement.

- For some problems, the judges may provide a simple testing tool simulating the judging process. If this is the case, you will find such a tool where you normally see the sample data. Executing the tool with a "-h" option should describe how to use the tool. Of course, the testing tool will only implement some test scenarios and only some functionality of the real judge program.