Maratona de Programação da SBC 2025

Sub-Regional Brasil do ICPC
Competencia Boliviana de Programación
The 2025 ICPC Gran Premio de Centroamerica
The 2025 ICPC Gran Premio de Mexico

September 13th 2025

Time limits

Prior to the contest, the judges will have resolved all problems using languages from at least two of the three specified language groups: C/C++, Java/Kotlin, and Python 3. The time limit for each problem is determined based on the runtime of these solutions
The times are given in seconds:

Problem	Time limit per test case*
A	2.0
B	1.0
C	0.5
D	1.5
E	1.5
F	0.5
G	1.5
H	1.0
I	0.5
J	0.5
K	0.5
L	0.5
M	0.5

`* Note that the time limits do not vary according to the programming language`

Memory limits

`C, C++20, Python: 1GB`
`Java, Kotlin: 1GB + 100MB stack`

Other limits

`Source file size: 100KB`

Building commands

```
C: gcc -x c -g -O2 -std=gnu11 -static -lm
C++20: g++ -x c++ -g -O2 -std=gnu++20 -static
Java: javac
Python: pypy3 -m py_compile
Kotlin: kotlinc -J-Xms1024m -J-Xmx1024m -J-Xss100m -include-runtime
```

C/C++

- Your program should return zero, executing, as last command `return 0` or `exit(0)`.
- It's been noticed that in some problems where the inputs are too big, the objects `cin` could be slow due to the buffer syncronization from the library `stdio`. If you still want to use `cin` and `cout`, a work around this problem is to use `ios_base::sync_with_stdio(0)`, and the beginning of your `main` function. Note that, in this case, using `scanf` and `printf` in the same program is not recommended, because having separate buffer could cause unexpected behaviors.

Java

- Do not declare '`package`' in your Java program.
- Note that you should obey convention while naming the source file. This means that name of your public class should be a capital letter (A, B, C, etc).
- The command to execute a Java solution is the following:
  `java -Xms1024m -Xmx1024m -Xss100m {problem_code}`

Kotlin

- Do not declare '`package`' in your Kotlin program.
- Note that you should obey convention while naming the solution file. This means that name of your source file should be (A.kt, B.kt, C.kt, etc).
- The command to execute a Kotlin solution is the following:
  `java -Xms1024m -Xmx1024m -Xss100m {problem_code}Kt`

Python

- Note that only Python 3 is supported.
- Python submissions will have their syntax checked; programs that do not pass this check will receive a "Compilation Error" verdict.
- Note that, this year, Python submissions will be executed with the **PyPy** interpreter, instead of the traditional CPython. This generally results in better performance, but be aware of any subtle differences in behavior.
- Command to execute a Python solution: `pypy3 {source}.py`

Instructions to submit a solution in Boca

Submitting solutions

To submit a solution to a problem you should use Boca's web portal:

- Open your browser
- Login with your team credentials (use the provided user and password).
- Go to the `Runs` tab. Select the target problem, the language used and the source file.

Verdicts

To see the verdict of the judge to your submission you should use Boca's web portal:

- Open your browser
- Login with your team credentials (use the provided user and password).
- Go to the `Runs` tab.

The verdicts you can get from the judge to your submission are the following:

- 1 - YES
- 2 - NO - Compilation error
- 3 - NO - Runtime error
- 4 - NO - Time limit exceeded
- 5 - NO - Wrong answer
- 6 - NO - Contact staff

The meaning of 1, 2, 3, 4 and 5 are self-explanatory.

- Regarding 1 and 5:
  - if the output of the solution submitted by the team is exactly the same as the judge's output or, in problems where it is specified that multiple solutions can be accepted and your solution is consistent with problem specifications, the verdict is "YES'
  - otherwise is "Wrong Answer'.
- Regarding 6: this verdict is used in unexpected circumstances. In this case, to get more information use the menu"Clarifications" and provide the "run" number.

Your program may be executed on multiple input files. Note that this means that if your program has more than one error (for example, "Time Limit Exceeded" and "Wrong Answer"), you may receive any of these errors as the verdict.

The output formatting must follow the sample output provided in the problem statement, although extra whitespace, within reason, is acceptable. For example, if you print thousands of spaces within the problem's time and output limits, this will be judged as "Wrong Answer"; however, an extra space at the end of a line or an additional blank line are acceptable.

If your program generates excessive output, your submission will receive a "Runtime Error" verdict. Note that any content written to the standard error output (*stderr*) also counts towards this limit. Therefore, an excess of debugging messages can cause runtime errors. If you encounter an unexpected runtime error, consider reducing the logs in *stderr*.
This also applies to interactive problems, covering both the output that your solution sends to the interactive judge and the logs in *stderr*.

Note that, for submissions in Java or Kotlin, it is necessary to follow the convention for the source file name (and class name). Additionally, there should be no `package` declaration. If these rules are not followed, the submission may receive a "Compilation Error" or even a "Runtime Error" verdict.

Penalties

Each submission for a problem that receives a "`NO`" verdict before the first "`YES`" verdict for the same problem will incur a time penalty of 20 minutes, added to the team's total time. There is no time penalty for unsolved problems.

The exceptions to this rule are "`NO - Compilation error`" and "`NO - Contact staff`", which have no associated time penalty.

Interactive Problems

It is possible that the problem set contains zero or more interactive problems.

In many respects, interactive problems are like any other: your program must read from standard input and write to standard output. The difference is that, in this case, your program's input and output are connected to another program (the judge), with which it must communicate continuously, sending data and receiving responses.

For this, it is crucial that your program "flushes" the output immediately after each write (use operations like `fflush(stdout)` in C, `cout.flush()` in C++, `System.out.flush()` in Java/Kotlin or `sys.stdout.flush()` in Python). If you fail to do this, the communication may not work correctly, resulting in verdicts like "Wrong Answer" or "Time Limit Exceeded".

Additionally, pay attention to the following points:

- The problem statement defines the **interaction protocol**: who starts, the format of each message, and how the dialogue should proceed. Your program must **follow this protocol exactly**, respecting spaces, line breaks, and formats.
- If your program fails to read the judge's response, or tries to read when there is nothing available, a deadlock may occur and the verdict will be incorrect.
- Malformed outputs or outputs in the incorrect order can lead the judge to terminate the execution with a "Wrong Answer" verdict.
- Interactions must be performed efficiently. If the program spends too much time without responding, it may receive a "Time Limit Exceeded" verdict, even if the algorithm is correct.
- The judge may behave in an adversarial manner, adapting the inputs to your responses to try to expose errors in the algorithm.

Clarifications

To request clarifications regarding a problem statement you should use Boca's web portal:

- Open your browser
- Login with your team credentials (use the provided user and password).
- Pick the problem from where you want clarifications and write your question.

Score

To visualize the scoreboard, you should use Boca's web portal:

- Open your browser
- Login with your team credentials (use the provided user and password).
- To go the `Score` tab to visualize the local score board.