Note that these optimisations will not allow a slow algorithm to pass. If you implement these changes, but are still experiencing timeouts, you will likely need to improve the time complexity of your solution. Submissions in JavaScript and C# should not require any particular I/O optimisations.

C++

If using only C-style I/O (`scanf` and `printf`), you may ignore this document.

If you are using `cin` to read input, add these two lines to the start of your main function:

```
int main() {
    cin.sync_with_stdio(0);
    cin.tie(0);
    ...
}
```

This will speed up the `cin` stream, by unsyncing `cin` from `scanf` and `cout`. If you do this, note that you should *not* use `scanf`.

Additionally, note that using `endl` to print newlines can be slow, as it always triggers a flush of the output stream. In most cases, where flushing is not required, you should instead use `'\n'` to print newlines.

Python

If you are timing out, consider submitting using the PyPy interpreter instead, as it is often several times faster than CPython. Note however that there are a few minor differences between PyPy and CPython, which are listed here:

https://doc.pypy.org/en/latest/cpython_differences.html

Java

Java's `Scanner` class is extremely slow – if you are timing out, consider using `BufferedReader` instead, in conjunction with `StringTokenizer` to split lines into tokens efficiently.

For example, to read a single line containing two integers:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(br.readLine());
int a = Integer.parseInt(st.nextToken());
int b = Integer.parseInt(st.nextToken());
```

Note that `br.readLine()` can throw an `IOException`, so you will need to either catch it, or declare the calling method(s) as throwing `IOException`:

```
public static void main(String[] args) throws IOException {
    BufferedReader br = ...
```