14th Marathon of Parallel Programming

SBAC-PAD & WSCAD – 2019

October 17th, 2019.

Warmup Rules for Local Contest

For all problems, read carefully the input and output session. For all problems, a sequential implementation is given, and it is against the output of those implementations that the output of your programs will be compared to decide if your implementation is correct. You can modify the program in any way you see fit, except when the problem description states otherwise. You must upload a compressed file (zip) with your source code, the *Makefile* and an execution script. The script must have the name of the problem. You can submit as many solutions to a problem as you want. Only the last submission will be considered. The *Makefile* must have the rule *all*, which will be used to compile your source code. The execution script runs your solution the way you design it – it will be inspected not to corrupt the target machine.

All *Local Teams* must use the computers that the organization provides. Only the judges have access to the judge machine.

The execution time of your program will be measured running it with *time* program and taking the real CPU time given. Each program will be executed at least three times with the same input and the mean time will be taken into account. The sequential program given will be measured the same way. You will earn points in each problem, corresponding to the division of the sequential time by the time of your program (*speedup*). The team with the most points at the end of the marathon will be declared the winner.

This problem set contains 1 problem; pages are numbered from 1 to 1.

General information

Compilation

You must use CC or CXX inside your *Makefile*. Be careful when redefining them! There is a simple *Makefile* inside you problem package that should be modified. Example:

```
FLAGS=-O3
EXEC=sum
CXX=icpc

all: $(EXEC)

$(EXEC):
      $(CXX) $(FLAGS) $(EXEC).cpp -c -o $(EXEC).o
      $(CXX) $(FLAGS) $(EXEC).o -o $(EXEC)
```

Running

You must have an execution script that has the same name of the problem. This script runs your solution the way you design it. There is a simple script inside you problem package that should be modified. Example:

```
$ cat A
#!/bin/bash
# This script runs Problem A
# export OMP_NUM_THREADS=32
# mpiexec –n 32 ./sum
./sum
```

Measure the execution time of your solution using *time* program. Add input/output redirection when collecting time. Use *diff* program to compare the original and your solution results. Example:

```
$ time –p ./A < original_input.txt > my_output.txt
real 4.94
user 0.08
sys 1.56

$ diff my_output.txt original_output.txt

$
```

Problem A

Harmonic progression sum

The simplest harmonic progression is

$$1/_1, 1/_2, 1/_3, 1/_4, 1/_5, \ldots$$

Let $S_n = \sum_{i=1}^{n}(1/_i)$, compute this sum to arbitrary precision after the decimal point.

Input

The input contains only one test case. The first line contains two values: the first is the number of digits D and the second is the value of N. Consider ($1 \leq D \leq 10^5$) and ($1 \leq N \leq 10^8$).

The input must be read from the standard input.

Output

The output contains only one line printing the value of the sum with exact D precision.

The output must be written to the standard output.

Example

Input example 1	Output example 1
`12 7`	`2.592857142857`