Distributed Round 1 2017

Submissions

Testrun
0pt | Not attempted **0/327 users** correct (0%)

pancakes
2pt | Not attempted **984/406 users** correct (242%)
11pt | Not attempted **920/975 users** correct (94%)

weird_editor
3pt | Not attempted **859/434 users** correct (198%)
20pt | Not attempted **505/807 users** correct (63%)

todd_and_steven
1pt | Not attempted **718/365 users** correct (197%)
30pt | Not attempted **230/437 users** correct (53%)

query_of_death
4pt | Not attempted **483/262 users** correct (184%)
29pt | Not attempted **230/377 users** correct (61%)

Top Scores

mk.al13n	100
semiexp.	100
qwerty787788	100
EgorKulikov	100
ikatanic	100
ecnerwala	100
Golovanov399	100
fagu	100
eatmore	100
Errichto.rekt	100

Problem A. Testrun

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small
0 points
2 minute timeout

The contest is finished.

Problem

This is a way to test your solutions, not a real problem!

When you submit a solution to this problem, it will run one testcase on a 100 nodes. This will allow you to estimate how fast your solution will run on our system.

Remember to change your solution appropriately before submitting it for real, so you don't fail because of a compilation error! The best way to check is to run your solution on the small input before submitting to the large input.

Input

There is no input for this problem. This means you should not include / import an input library.

Output

Doesn't really matter what you output. If your solution runs successfully to completion, it will be judged as "Wrong Answer".

Limits

Each node will have access to 1 GB of RAM, and a time limit of 26 seconds. The maximum number of messages a single node can send is 5000, and the maximum sum of the sizes of those messages is 8MB.
This problem only has one small test case. It will run on 100 nodes.

Problem B. pancakes

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 2 points *2 minute timeout*	The contest is finished.
large 11 points *10 minute timeout*	The contest is finished.

Problem

Pancakes

At the Infinite House of Pancakes, **D** hungry diners are seated around a circular table. They are numbered from 0 at the top of the table and clockwise around to **D**-1. Each diner likes a particular type of pancake: the type with the same number as the diner.

You are a pancake server with a stack of pancakes. You are currently at the top of the table (just about to serve diner 0), and you will walk around the table clockwise, perhaps multiple times. Every time you pass by a diner, if the pancake at the top of the stack is the diner's preferred kind, you will serve that pancake. You will keep serving that diner until the top of the stack no longer matches the diner's preference; then you will move on to the next diner, and so on.

Each time you complete a revolution, after passing by diner **D**-1 and before reaching diner 0 again, you check to see if the stack is empty. If so, you are done. Otherwise, you continue clockwise around the table for another revolution, serving the diners.

For example, suppose there are 4 diners and 4 pancakes in the stack 3, 1, 2, 0 in order from top to bottom. You start your first revolution by skipping diners 0, 1 and 2 to serve diner 3 the top pancake, getting the stack down to 1, 2, 0. After that, you go back to diner 0 (starting a second revolution), skipping it to get to diner 1 and giving her pancake 1, getting the stack down to 2, 0. Since she doesn't like pancake 2, you go to diner 2, which gets it and you have only a single pancake 0 left. To serve it, you go through diner 3 and back to 0 (starting a third revolution), who gets the final pancake, and your job is done. You required 3 revolutions in total for this particular arrangement, even though you didn't need to finish the last one completely.

Note that there may be some diners who do not receive any pancakes.

Given the initial stack of pancakes, how many revolutions will you make around the table?

Input

The input library is called "pancakes"; see the sample inputs below for examples in your language. It defines three methods:

- **GetStackSize()**:
  - Takes no argument.
  - Returns a 64-bit integer: the initial number of pancakes in the stack.
  - Expect each call to take 0.8 microseconds.
- **GetNumDiners()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number **D** of diners seated around the table.
  - Expect each call to take 0.8 microseconds.
- **GetStackItem(i)**:
  - Takes a 64-bit number in the range $0 \le i <$ GetStackSize().
  - Returns a 64-bit integer: the i-th pancake type in the stack, where i = 0 corresponds to the first one we plan to serve.
  - Expect each call to take 0.8 microseconds.

Output

Output a single integer: the number of revolutions we make around the table.

Limits

Time limit: 3 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$0 \le$ GetStackItem(i) $<$ GetNumDiners().

Small dataset

Number of nodes: 10.
$1 \le$ GetStackSize() $\le 10^5$.
$3 \le$ GetNumDiners() $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ GetStackSize() $\le 10^8$.
$3 \le$ GetNumDiners() $\le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 3 For sample input 2: 1 For sample input 3: 4

The first sample case is the one explained in the statement.
Sample input libraries:
Sample input for test 1: pancakes.h [CPP] pancakes.java [Java]
Sample input for test 2: pancakes.h [CPP] pancakes.java [Java]
Sample input for test 3: pancakes.h [CPP] pancakes.java [Java]

Problem C. weird_editor

small 3 points *2 minute timeout*	The contest is finished.
large 20 points *10 minute timeout*	The contest is finished.

Problem

Weird Editor

You just installed a text editor in your computer to edit a text file containing only a positive integer (in base 10). Unfortunately, the editor you installed was not as versatile as you would have hoped.

The editor supports only one operation: choosing and removing any digit, and concatenating one `0` (zero) at the right end of the sequence. In this way, the length of the digit sequence is always preserved.

For instance, suppose your initial digit sequence is `3001`. If you applied the operation to the third digit from the left, you would obtain `3010`. If you then applied the operation on `3010` to the second digit form the left, you would obtain `3100`. In this case, `3100` is the largest result that can be obtained using the allowed operation zero or more times.

What is the maximum number that can be the result of applying the allowed operation to the given input number zero or more times? Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (1000000007).

Input

The input library is called "weird_editor"; see the sample inputs below for examples in your language. It defines two methods:

- **GetNumberLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of digits in the number you're given.
  - Expect each call to take 0.11 microseconds.
- **GetDigit(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetNumberLength().
  - Returns a 64-bit integer: The i-th digit in the given number. Digits are numbered from left (most significant) to right (least significant). That is, GetDigit(0) is the most significant digit and GetDigit(GetNumberLength() - 1) is the least significant digit.
  - Expect each call to take 0.11 microseconds.

Output

Output a single integer: the maximum number that can be obtained by applying the allowed operation to the input number zero or more times. Output that number modulo the prime 10^9+7 (1000000007).

Limits

Time limit: 3 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$0 \le$ GetDigit(i) ≤ 9, for all i.
GetDigit(0) $\ne 0$.

Small dataset

Number of nodes: 10.
$2 \le$ GetNumberLength() $\le 10^6$.

Large dataset

Number of nodes: 100.
$2 \le$ GetNumberLength() $\le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 3100 For sample input 2: 33000000 For sample input 3: 999999944

Sample input libraries:
Sample input for test 1: weird_editor.h [CPP] weird_editor.java [Java]
Sample input for test 2: weird_editor.h [CPP] weird_editor.java [Java]
Sample input for test 3: weird_editor.h [CPP] weird_editor.java [Java]

Problem D. todd_and_steven

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 1 points *2 minute timeout*	The contest is finished.
large 30 points *10 minute timeout*	The contest is finished.

Problem

Todd and Steven

By now, it is a programming interview cliché: How do you sort a very large sequence of unique integers in increasing order? Finally, we are ready to reveal the correct answer:

1. Give all of the odd integers from the sequence to one assistant named Todd, and ask Todd to sort those integers in increasing order.
2. Give all of the even integers from the sequence to another assistant named Steven, and ask Steven to sort those integers in increasing order.
3. Merge Todd's sequence with Steven's sequence to produce the final sorted sequence.

Todd and Steven have already performed steps 1 and 2 on a certain sequence, and now we would like you to perform step 3. Since the resulting sorted sequence can be very long, we only ask you to output a hash of it as proof that you found it. Let X_j denote the j-th element (counting starting from 0) of the merged sequence. Please find the sum, over all j, of (X_j) XOR j. (Here XOR refers to the bitwise operation between the two integers, represented in both C++ and Java by the operator ^.) Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (1000000007).

Input

The input library is called "todd_and_steven"; see the sample inputs below for examples in your language. It defines two methods:

- **GetToddLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in Todd's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetToddValue(i)**:
  - Takes a 64-bit integer argument in the range 0 ≤ i < `GetToddLength()`.
  - Returns a 64-bit integer: the i-th value of Todd's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetStevenLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in Steven's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetStevenValue(i)**:
  - Takes a 64-bit integer argument in the range 0 ≤ i < `GetStevenLength()`.
  - Returns a 64-bit integer: the i-th value of Steven's sorted sequence.
  - Expect each call to take 0.05 microseconds.

Output

Output one line with a single 64-bit integer: the sum described in the problem statement, modulo the prime 10^9+7 (1000000007).

Limits

Time limit: 4 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le$ `GetToddValue(i)` $\le 5 \times 10^9$, for all i.
`GetToddValue(i)` < `GetToddValue(i + 1)`, for all i. (Todd's sequence is sorted in increasing order.)
`GetToddValue(i) % 2 = 1`, for all i. (All elements in Todd's sequence are odd.)

$1 \le$ `GetStevenValue(i)` $\le 5 \times 10^9$, for all i.
`GetStevenValue(i)` < `GetStevenValue(i + 1)` for all i. (Steven's sequence is sorted in increasing order.)
`GetStevenValue(i) % 2 = 0`, for all i. (All elements in Steven's sequence are even.)

Small dataset

Number of nodes: 10.
$1 \le$ `GetToddLength()` $\le 10^6$.
$1 \le$ `GetStevenLength()` $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ `GetToddLength()` $\le 10^9$.
$1 \le$ `GetStevenLength()` $\le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 8 For sample input 2: 200 For sample input 3: 111

Sample input libraries:
Sample input for test 1: todd_and_steven.h [CPP] todd_and_steven.java [Java]
Sample input for test 2: todd_and_steven.h [CPP] todd_and_steven.java [Java]
Sample input for test 3: todd_and_steven.h [CPP] todd_and_steven.java [Java]

Problem E. query_of_death

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 4 points *2 minute timeout*	The contest is finished.
large 29 points *10 minute timeout*	The contest is finished.

Problem

Query of Death

We planned a nice simple warm-up DCJ problem for you this year: find the sum of many values. You can call a `GetLength()` function to get the number of values and a `GetValue(i)` function to get the i-th of those values; to make it even easier, each of those values is either 0 or 1. Simple, right? Unfortunately, we have been having a technical difficulty, and now the contest is starting and it is too late to fix it.

The issue is that there is exactly one value of i — we are not sure what that value is, but we will call it i_{qod} — that is a "query of death" (a term occasionally used at Google for a query with severe adverse effects) that causes the following malfunction. The first time that GetValue(i_{qod}) is called on a node, the function will return the correct i_{qod}-th value. However, this will cause the GetValue function to "break" on that node. After that, *every* future call to GetValue(i) on that node will return 0 or 1 purely at (pseudo)random, independently of the value of i or of any previous calls. Other nodes are not affected when a node breaks in this way, but the malfunction can still happen in the future: any other node on which you call GetValue(i_{qod}) will also break.

The i_{qod} value that causes the breakage is the same for every node within a test case; it may vary across test cases, though. Nodes do not remain broken across different test cases.

As an example, suppose that we have two unbroken nodes A and B, and two values i_{ok} and i_{qod}. Then the following sequence of calls would produce the following results:

1. GetValue(i_{ok}) on node A: the correct value is returned.
2. GetValue(i_{qod}) on node A: the correct value is returned, but node A breaks.
3. GetValue(i_{ok}) on node B: the correct value is returned.
4. GetValue(i_{ok}) on node A: a random value is returned.
5. GetValue(i_{qod}) on node A: a random value is returned.
6. GetValue(i_{qod}) on node B: the correct value is returned, but node B breaks.
7. GetValue(i_{qod}) on node B: a random value is returned.
8. GetValue(i_{ok}) on node B: a random value is returned.
9. GetValue(i_{qod}) on node A: a random value is returned.
10. GetValue(i_{ok}) on node A: a random value is returned.

We apologize for the inconvenience, but can you find the sum anyway?

Input

The input library is called "query_of_death"; see the sample inputs below for examples in your language. It defines two methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the total number of values to be summed up. (This function still works correctly even on a broken node.)
  - Expect each call to take 0.2 microseconds.
- **GetValue(i)**:
  - Takes a 64-bit number in the range $0 \le i <$ `GetLength()`.
  - Returns a 32-bit number (which is always either 0 or 1): the `i`-th value if the node is not broken, or 0 or 1 at (pseudo)random if the node is broken.
  - Expect each call to take 0.2 microseconds.

Output

Output a single line with one integer: the sum of all of the values.

Limits

Time limit: 2 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
There is exactly one i_{qod} value, which is the same for each node, and it is within the allowed range for GetLength().
$0 \le$ GetValue(i) ≤ 1, for all i.

Small dataset

Number of nodes: 10.
$1 \le$ GetLength() $\le 10^4$.

Large dataset

Number of nodes: 100.
$1 \le$ GetLength() $\le 10^8$.

Sample

Input	Output
See input files below.	For sample input 1: 2 For sample input 2: 3 For sample input 3: 3

The code for the samples simulates the node-breaking behavior described in the statement; the actual test cases have the specified behavior, but the implementation (e.g., of randomness on a broken node) is not necessarily the same.

Sample input libraries:
Sample input for test 1: query_of_death.h [CPP] query_of_death.java [Java]
Sample input for test 2: query_of_death.h [CPP] query_of_death.java [Java]
Sample input for test 3: query_of_death.h [CPP] query_of_death.java [Java]

Distributed Round 2 2017

Submissions

Testrun

0pt	Not attempted **0/58 users** correct (0%)

flagpoles

1pt	Not attempted **335/181 users** correct (185%)
11pt	Not attempted **277/320 users** correct (87%)

number_bases

5pt	Not attempted **241/186 users** correct (130%)
17pt	Not attempted **188/226 users** correct (83%)

broken_memory

3pt	Not attempted **196/88 users** correct (223%)
25pt	Not attempted **77/142 users** correct (54%)

nanobots

8pt	Not attempted **104/69 users** correct (151%)
30pt	Not attempted **31/68 users** correct (46%)

Top Scores

fagu	100
bmerry	100
krijgertje	100
ecnerwala	100
pashka	100
Swistakk	100
KalininN	100
adsz	100
Gennady.Korotkevich	100
eatmore	100

Problem A. Testrun

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small
0 points
2 minute timeout

The contest is finished.

Problem

This is a way to test your solutions, not a real problem!

When you submit a solution to this problem, it will run one testcase on a 100 nodes. This will allow you to estimate how fast your solution will run on our system.

Remember to change your solution appropriately before submitting it for real, so you don't fail because of a compilation error! The best way to check is to run your solution on the small input before submitting to the large input.

Input

There is no input for this problem. This means you should not include / import an input library.

Output

Doesn't really matter what you output. If your solution runs successfully to completion, it will be judged as "Wrong Answer".

Limits

Each node will have access to 1 GB of RAM, and a time limit of 26 seconds. The maximum number of messages a single node can send is 5000, and the maximum sum of the sizes of those messages is 8MB.
This problem only has one small test case. It will run on 100 nodes.

Problem B. flagpoles

small 1 points *2 minute timeout*	The contest is finished.
large 11 points *10 minute timeout*	The contest is finished.

Problem

Flagpoles

Cody-Jamal, the famous conceptual artist, was called to design the new United Nations headquarters. The entrance displays a single row of flagpoles with the flags of different countries. Each flagpole is exactly 1 meter away from its neighbor(s). Since different nations have different rules about how high their flags must be flown, the tips of the flagpoles may have different heights.

The scientists from the famous Detecting Collinearity Journal have become interested in the flagpoles. In particular, they want to know the maximum number of consecutive flagpoles with tips that are collinear. A set of contiguous flagpoles has collinear tips if there is a constant d such that, for every pair of adjacent flagpoles in the set, the height of the right flagpole's tip minus the height of the left flagpole's tip is equal to d. Notice that the condition is always true for a set of up to 2 flagpoles.

For example, if the flagpoles' heights are 5, 7, 5, 3, 1, 2, 3, in left-to-right order, the leftmost 2 flagpoles and the rightmost 3 flagpoles are examples of consecutive sets of flagpoles with collinear tips. The flagpoles with heights 7 and 1, together with those in between them, are another example. The leftmost 3 flagpoles, however, do not have collinear tips, so they do not form such a set.

Given the height in meters of each flagpole tip, in the left-to-right order in which they appear, can you help the DCJ calculate the maximum size of a set of consecutive flagpoles with collinear tips?

Input

The input library is called "flagpoles"; see the sample inputs below for examples in your language. It defines two methods:

- **GetNumFlagpoles()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of flagpoles in the row.
  - Expect each call to take 0.17 microseconds.
- **GetHeight(i)**:
  - Takes exactly one 64-bit integer argument: a position i, 0 ≤ i < `GetNumFlagpoles()`.
  - Returns a 64-bit integer: the height, in meters, of the flagpole at the ith position from left to right. The ith flagpole is always i meters to the right of the 0th flagpole.
  - Expect each call to take 0.17 microseconds.

Output

Output one line with a single integer: the maximum number of consecutive flagpoles with collinear top ends.

Limits

Time limit: 3 seconds.
Memory limit per node: 512 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le \text{GetHeight(i)} \le 10^{18}$.

Small dataset

Number of nodes: 10.
$1 \le \text{GetNumFlagpoles()} \le 10^{6}$.

Large dataset

Number of nodes: 100.
$1 \le \text{GetNumFlagpoles()} \le 10^{9}$.

Sample

Input	Output
See input files below.	For sample input 1: 4 For sample input 2: 4 For sample input 3: 2

Sample input 1 is the example given in the problem statement.

Sample input libraries:
Sample input for test 1: flagpoles.h [CPP] flagpoles.java [Java]
Sample input for test 2: flagpoles.h [CPP] flagpoles.java [Java]
Sample input for test 3: flagpoles.h [CPP] flagpoles.java [Java]

Problem C. number_bases

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 5 points *2 minute timeout*	The contest is finished.
large 17 points *10 minute timeout*	The contest is finished.

Problem

Number Bases

You are given three sequences X, Y, and Z of equal length. The sequences consist of digits. In this problem we deal with non-decimal bases, so digits are arbitrary non-negative integers, not necessarily restricted to the usual 0 through 9 range.

Your task is to investigate the possible bases for which the equation **X + Y = Z** is both valid and holds true. You must determine whether there is no such base, more than one such base, or only one such base. If there is only one such base, you must find it.

A base is an integer greater than or equal to 2. For X + Y = Z to be valid in base B, all digits in all three sequences have to be strictly less than B. For X + Y = Z to be true in base B, the sum of the integer denoted by X in base B and the integer denoted by Y in base B has to be equal to the integer denoted by Z in base B.

More formally: let S[i] be the i-th digit from the right of a sequence of digits S, with i counted starting from 0. Then, for a given S and an integer base B, we will define the integer denoted by S in base B as f(S, B) = the sum of all S[i] × B^i. Then the equation X + Y = Z is true for a base B if and only if f(X, B) + f(Y, B) = f(Z, B).

For example, consider the sequences X = {1, 2, 3}, Y = {4, 5, 6} and Z = {5, 8, 0}, written with the most significant digits on the left, as usual. That is, X[0] = 3, X[1] = 2 and X[2] = 1. B = 8 is an invalid base, because it is not strictly greater than the middle digit of Z. B = 10 is a valid base, but the expression is not true in that case because 123 + 456 ≠ 580. B = 9 is a valid base that also makes the expression true, because {1, 2, 3} in base 9 is 102, {4, 5, 6} in base 9 is 375 and {5, 8, 0} in base 9 is 477, and 102 + 375 = 477. For this case, B = 9 is the only possible choice of a valid base B that makes the expression true.

On the other hand, the one-digit sequences X = {10}, Y = {20} and Z = {30} have multiple bases for which the equation X + Y = Z is both valid and true. Any value of B greater than 30 would suffice.

Input

The input library is called "number_bases"; see the sample inputs below for examples in your language. It defines four methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of digits in each digit sequence X, Y, and Z.
  - Expect each call to take 0.34 microseconds.
- **GetDigitX(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence X, numbered from right (least significant) to left (most significant). That is, GetDigitX(0) is the least significant digit of X and GetDigitX(GetLength() - 1) is the most significant digit of X.
  - Expect each call to take 0.34 microseconds.
- **GetDigitY(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence Y, numbered from right (least significant) to left (most significant).
  - Expect each call to take 0.34 microseconds.
- **GetDigitZ(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence Z, numbered from right (least significant) to left (most significant).
  - Expect each call to take 0.34 microseconds.

Output

Output a single line with a single token x. If there is a single base B that makes the expression valid and true, x must be the base 10 representation of B. If there are multiple values of B that make the expression valid and true, x must be `NON-UNIQUE`. If there is no such value of B, x must be `IMPOSSIBLE`.

Limits

Time limit: 3 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$0 \le$ GetDigitX(i) $\le 10^6$, for all i.
$0 \le$ GetDigitY(i) $\le 10^6$, for all i.
$0 \le$ GetDigitZ(i) $\le 10^6$, for all i.
(Contrary to typical notation, it is possible for any of the digit sequences in the input to have a zero at its most significant position. Valid bases are not restricted to be less than 10^6 or any other upper bound.)

Small dataset

Number of nodes: 10.
$1 \le$ GetLength() $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ GetLength() $\le 10^8$.

Sample

Input	Output
See input files below.	For sample input 1: 9 For sample input 2: NON-UNIQUE For sample input 3: IMPOSSIBLE

Sample input libraries:
Sample input for test 1: number_bases.h [CPP] number_bases.java [Java]
Sample input for test 2: number_bases.h [CPP] number_bases.java [Java]
Sample input for test 3: number_bases.h [CPP] number_bases.java [Java]

Problem D. broken_memory

small 3 points *2 minute timeout*	The contest is finished.
large 25 points *10 minute timeout*	The contest is finished.

Problem

Broken Memory

As you may remember from last year, we have a tendency to screw things up at the worst possible moment. For 2017, we promised ourselves that we wouldn't misplace any test cases, though, and we delivered. Unfortunately, last night we spilled a peach smoothie over some of our Google Cloud servers. We immediately called the Dedicated Cloud Janitor (DCJ), but he was fed up with our continuous messes and refused to help. So, we are turning to our most powerful allies, our contestants, to once again rescue us from ourselves.

Fortunately, the servers were already loaded with the data for the problems, so before the damage, the memory was exactly the same in all of them. We conducted a preliminary investigation that revealed that every node's memory was damaged in a *different* place.

We have narrowed the search to a relatively small part of the memory, and we have encoded that as a list of integers for your convenience. Each node has the same list of integers, except at exactly one damaged position; the value there will be *different* from the corresponding value on all other nodes.

For example, suppose the original data is represented by the following list of integers: 1 5 9 3 1. Suppose the broken index for node 0 is 2. That means that when requesting the values for indices 0, 1, 3 and 4, the returned value will be correct (1, 5, 3 and 1, respectively). However, when requesting the value for index 2 on node 0, the returned value could be 1 or 5 or 352462352, for example, but definitely not the correct one (9). If, on the other hand, node 1's broken index is 0, then, when requesting the value of indices 1, 2, 3 and 4, the correct values (5, 9, 3 and 1, respectively) will be returned, but when requesting index 0, the returned value could be 9 or 5 or 379009, but definitely not the correct one (1). Notice that for node 3, neither index 2 nor index 0 can be the broken index, because each node is broken at a different index from all other nodes.

Can you find the broken index on each node for us?

Input

The input library is called "broken_memory"; see the sample inputs below for examples in your language. It defines two methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in the part of the memory where all the damage happened.
  - Expect each call to take 0.02 microseconds.
- **GetValue(i)**:
  - Takes a 64-bit number in the range 0 ≤ i < GetLength()
  - Returns a 64-bit integer: the i-th value in the memory.
  - Expect each call to take 0.02 microseconds.

Output

Output a single line with `NumberOfNodes()` integers: the broken index for each node, in ascending order of node ID.

Limits

Time limit: 2 seconds.
Memory limit per node: 256 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 128 KB. **(Notice that this is less than usual.)**
$1 \le \text{GetValue(i)} \le 10^{18}$, for all i.

Small dataset

Number of nodes: 10.

10 ≤ GetLength() ≤ 1000.

Large dataset

Number of nodes: 100.
100 ≤ GetLength() ≤ 10^7.

In this problem, the sample inputs are valid for running on 10 nodes, and the sample outputs are computed using 10 nodes.

Sample

Input	Output
See input files below.	For sample input 1: 0 1 2 3 4 5 6 7 8 9 For sample input 2: 29 28 27 26 25 24 23 22 21 20 For sample input 3: 12 13 14 15 8 9 10 11 4 5

Sample input libraries:
Sample input for test 1: broken_memory.h [CPP] broken_memory.java [Java]
Sample input for test 2: broken_memory.h [CPP] broken_memory.java [Java]
Sample input for test 3: broken_memory.h [CPP] broken_memory.java [Java]

Problem E. nanobots

small 8 points *2 minute timeout*	The contest is finished.
large 30 points *10 minute timeout*	The contest is finished.

Problem

Nanobots

A group of medical researchers is working on a new treatment to fight bacteria. In the treatment, special nanobots are transferred into the patient's body, where they locate and trap any harmful bacteria. This allows the patient to get better, and the researchers can later retrieve the trapped bacteria for further study.

Not any nanobot can trap any bacterium, though. A nanobot can be characterized by two traits: size and speed. A nanobot can only trap bacteria that are smaller than it (otherwise, the bacteria will not fit in the nanobot's cage) and also slower than it (otherwise, the bacteria can escape the nanobot). Formally, a nanobot with speed A and size B can trap a bacterium with speed C and size D if and only if A > C and B > D. The speeds and sizes of both nanobots and bacteria are in the inclusive range [1, `GetRange()`].

You have a group of nanobots and you want to know how effective they are at trapping bacteria. Your goal is to find how many of the `GetRange()`^{2} possible bacteria get trapped by the team of nanobots. Unfortunately, you cannot directly examine the speed and size of your nanobots. You can only experiment by introducing a bacterium with a specific size and speed, and watching whether this bacterium gets trapped by the team of nanobots or not. A team of nanobots will trap a bacterium with speed C and size D if and only if there is at least one nanobot in the team that has both speed strictly greater than C and size strictly greater than D.

You may carry out as many experiments of this sort as you want... within the allowed running time for the problem, of course! You can choose the speed and size of the bacteria in each experiment, and for each one, you receive one piece of data: whether or not the bacteria was trapped. Each experiment uses the full team of nanobots, and the same nanobot can catch bacteria in different experiments. Based on that information, you need to determine how many of the `GetRange()`^{2} possible bacteria would be trapped by the team of nanobots. (Because the speed can take any integer value in [1, `GetRange()`], and the same is true for the size, there are `GetRange()`^{2} possible bacteria.) Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (1000000007).

Distributed Code Jam is not a licensed physician. Nothing in this problem statement should be construed as an attempt to offer medical advice. Distributed Code Jam is also not a licensed scientist. Nothing in this problem statement should be construed as an attempt to offer scientific advice. Using nanobots to fight harmful bacteria definitely sounds cool, though.

Input

The input library is called "nanobots"; see the sample inputs below for examples in your language. It defines three methods:

- **GetNumNanobots()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of nanobots on the team.
  - Expect each call to take 0.2 microseconds.
- **GetRange()**:
  - Takes no argument.
  - Returns a 64-bit integer: the maximum valid value for speeds and sizes of both bacteria and nanobots.
  - Expect each call to take 0.2 microseconds.
- **Experiment(c, d)**:
  - Takes exactly two 64-bit integer arguments: a size c and a speed d, 1 ≤ c,d ≤ `GetRange()`.
  - Returns a char: T if a bacteria with size c and speed d is trapped by the nanobots, or E if it escapes.
  - Expect each call to take 0.2 microseconds.

Output

Output one line with a single integer: how many of the different bacteria with both size and speed in the inclusive range [1, `GetRange()`] would be trapped by the nanobots, modulo the prime 10^9+7 (1000000007). Two bacteria are considered different if and only if they have different speed and/or different size.

Limits

Time limit: 18 seconds.
Memory limit per node: 256 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$2 \le$ GetRange() $\le 10^{12}$.
`Experiment(c, d)` is either uppercase T or uppercase E, for all c, d.
The results of `Experiment` are consistent with the same team of `GetNumNanobots()` nanobots across all nodes.
It is possible that multiple nanobots might have the same size and speed.

Small dataset

Number of nodes: 10.
$1 \le$ `GetNumNanobots()` $\le 10^5$.

Large dataset

Number of nodes: 100.
$1 \le$ `GetNumNanobots()` $\le 10^7$.

Sample

Input	Output
See input files below.	For sample input 1: 26 For sample input 2: 998496508 For sample input 3: 0

Sample input libraries:
Sample input for test 1: nanobots.h [CPP] nanobots.java [Java]
Sample input for test 2: nanobots.h [CPP] nanobots.java [Java]
Sample input for test 3: nanobots.h [CPP] nanobots.java [Java]

Distributed World Finals 2017

Submissions

Testrun

0pt	Not attempted **0/9 users** correct (0%)

baby_blocks

2pt	Not attempted **21/21 users** correct (100%)
17pt	Not attempted **11/19 users** correct (58%)

lemming

5pt	Not attempted **21/21 users** correct (100%)
14pt	Not attempted **17/19 users** correct (89%)

median

10pt	Not attempted **11/18 users** correct (61%)
19pt	Not attempted **0/3 users** correct (0%)

lispp3

11pt	Not attempted **3/9 users** correct (33%)
22pt	Not attempted

Top Scores

ecnerwala	59
eatmore	49
krijgertje	48
pashka	48
Swistakk	48
W4yneb0t	48
Merkurev	48
Gennady.Korotkevich	42
tomconerly	38
adsz	38

Problem A. Testrun

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small
0 points
2 minute timeout

The contest is finished.

Problem

This is a way to test your solutions, not a real problem!

When you submit a solution to this problem, it will run one testcase on a 100 nodes. This will allow you to estimate how fast your solution will run on our system.

Remember to change your solution appropriately before submitting it for real, so you don't fail because of a compilation error! The best way to check is to run your solution on the small input before submitting to the large input.

Input

There is no input for this problem. This means you should not include / import an input library.

Output

Doesn't really matter what you output. If your solution runs successfully to completion, it will be judged as "Wrong Answer".

Limits

Each node will have access to 1 GB of RAM, and a time limit of 26 seconds. The maximum number of messages a single node can send is 5000, and the maximum sum of the sizes of those messages is 8MB.
This problem only has one small test case. It will run on 100 nodes.

Problem B. baby_blocks

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 2 points *2 minute timeout*	The contest is finished.
large 17 points *10 minute timeout*	The contest is finished.

Problem

Baby Blocks

Your two babies Alicia and Bobby love to play with toy blocks. Their blocks come lined up in a single line in a box; each block has a certain weight. When it is time to play, Alicia takes the leftmost i blocks for some number i ≥ 1, and Bobby takes the j rightmost blocks for some number j ≥ 1. i and j are not necessarily the same, and the babies always choose values such that i + j does not exceed the total number of blocks. There may be some blocks left over in the box after the babies have finished taking blocks.

Like many babies, Alicia and Bobby are very concerned about unfairness. After the babies have taken their blocks, but before the babies have started to play with them, Alicia will put all of her blocks on one side of a scale, and Bobby will put all of his blocks on the other side of the scale. If the two total weights are equal, the babies will play happily. Otherwise, they will start to cry and throw the blocks around. You would prefer to avoid this.

How many possible ways are there for the babies to take blocks so that they will be happy? Two ways are different if and only if their (i, j) pairs are different.

Input

The input library is called "baby_blocks"; see the sample inputs below for examples in your language. It defines two methods:

- **GetNumberOfBlocks()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of blocks in the box.
  - Expect each call to take 0.1 microseconds.
- **GetBlockWeight(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetNumberOfBlocks().
  - Returns a 64-bit integer: the weight of the i-th block in the box, where i = 0 corresponds to the leftmost block.
  - Expect each call to take 0.1 microseconds.

Output

Output a single integer: the number of different ways for the babies to take blocks that will make them happy.

Limits

Time limit: 3 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le \text{GetBlockWeight(i)} \le 10^9$.

Small dataset

Number of nodes: 10.
$2 \le \text{GetNumberOfBlocks()} \le 10^6$.

Large dataset

Number of nodes: 100.
$2 \le \text{GetNumberOfBlocks()} \le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 1 For sample input 2:

```
                        2
                        For sample input 3:
                        0
```

Sample input libraries:
Sample input for test 1: baby_blocks.h [CPP] baby_blocks.java [Java]
Sample input for test 2: baby_blocks.h [CPP] baby_blocks.java [Java]
Sample input for test 3: baby_blocks.h [CPP] baby_blocks.java [Java]

Problem C. lemming

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 5 points *2 minute timeout*	The contest is finished.
large 14 points *10 minute timeout*	The contest is finished.

Problem

Lemming

Our Distributed Code Jam team has a pet lemming, Larry, who likes to follow instructions. To give Larry some exercise, we place him on a table with grid of cells, each of which contains an arrow pointing either up, down, left, or right. Larry will always move one unit in the direction indicated by his current cell. If this takes him off the edge of the table, he falls harmlessly onto some padding. Otherwise, he follows the direction indicated by his new cell, and so on, possibly continuing in an infinite loop.

We don't want Larry to fall off the table or exercise forever, so we want to turn one or more of the existing cells on the table into blank *pickup points*. If Larry starts on or reaches a pickup point, we pick him up and the exercise period is over.

What is the minimum number of cells that we need to change into pickup points to ensure that no matter where Larry is initially placed on the grid, he will eventually be picked up, instead of falling off the table or exercising forever?

Input

The input library is called "lemming"; see the sample inputs below for examples in your language. It defines three methods:

- **GetRows()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of rows in the input grid.
  - Expect each call to take 0.06 microseconds.
- **GetColumns()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of columns in the input grid.
  - Expect each call to take 0.06 microseconds.
- **GetDirection(r, c)**:
  - Takes two 64-bit integers in the ranges $0 \le r <$ GetRows(), $0 \le c <$ GetColumns().
  - Returns a character: the contents of the cell at row r and column c of the input grid. The character is one of ^ (ASCII code 94), lowercase v, <, > which represents up, down, left and right respectively. Rows are numbered from top to bottom. Columns are numbered from left to right, as explained above.
  - Expect each call to take 0.06 microseconds.

Output

Output one line with a single integer: the minimum number of pickup points that you need to create.

Limits

Number of nodes: 100 **(for both the Small and Large datasets)**.
Time limit: 15 seconds.
Memory limit per node: 1 GB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le$ GetRows() $\le 30{,}000$.
$1 \le$ GetColumns() $\le 30{,}000$.

Small dataset

GetDirection(r, c) is one of the characters v, <, >, for all r and c.

Large dataset

GetDirection(r, c) is one of the characters ^, v, <, >, for all r and c.

Sample

Input	Output
See input files below.	For sample input 1: 6 For sample input 2: 8 For sample input 3: 4

For ease of reading, these are the input matrices in the samples:

```
<v><
<<v>
>><>

><><><><><><><>

<v<
>>^
v>>
^^^
```

Note that the last sample case would not appear in the Small dataset.

Sample input libraries:
Sample input for test 1: lemming.h [CPP] lemming.java [Java]
Sample input for test 2: lemming.h [CPP] lemming.java [Java]
Sample input for test 3: lemming.h [CPP] lemming.java [Java]

Problem D. median

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 10 points *2 minute timeout*	The contest is finished.
large 19 points *10 minute timeout*	The contest is finished.

Problem

Median

In this year's online rounds, we ran into trouble with queries of death and broken memory, so to make it up to you, we decided to put a very easy median-finding problem in the Finals. We were determined to avoid any more issues with peach smoothie spills, so we asked our DCJ (Data Curating Janitor) to set up two special arrays of 100 nodes for this problem, one for each of the Small and Large datasets.

Our DCJ prepared each array according to our instruction:

1. Choose N pieces of data to store. Since the problem is about finding a median, each of the N pieces of data is a non-negative integer. (The same number might appear more than once in the data.) N is also guaranteed to be odd, because who likes finding the median of an even number of integers?
2. A median-finding problem would be far too easy if the data were given in order, so, choose one of the N! permutations of the data uniformly at (pseudo-)random.
3. Write these N pieces of data, in the order of the chosen permutation, clockwise around the edge of a circular disk, starting from a *fixed point* on the very bottom.
4. Make one exact copy of that disk for each node to use.
5. A contestant can call GetData(i) on a node. Each disk has a data-reading needle that starts at the same fixed point. That node will move its data-reading needle to read and return the data i places away (clockwise) from the fixed point, and then the data-reading needle will go back to the fixed point. Since the disk is a circle, it is allowed for i to be greater than or equal to N; the node will return the (i % N)th piece of data. For example, for N = 5, calls to GetData(0) and GetData(5) would return the same piece of data.
6. A contestant can also call GetN() to learn the value of N.

Our DCJ (Data Curating Janitor) completed these tasks the night before the Finals, and, to celebrate, we threw a party and brought in a DCJ (Disc Controlling Jockey) to play some music. Unfortunately, this morning, we learned that the Disc Controlling Jockey mistook our array of nodes for the Large dataset for a very elaborate turntable, and might have spun any or all of the disks! This means that, for the Large dataset only, even though all the nodes' disks have the same data in the same clockwise order, *their fixed points might no longer be the same*. For example, a piece of data that one node thinks is the 0th might be the 3rd on another node!

Moreover, we forgot to ban flavors of smoothie other than peach, and our Disc Controlling Jockey also spilled a strawberry smoothie all over the array of nodes for the Large dataset. Because of this, for the Large dataset only, the GetN() function no longer returns any useful data.

The Finals have already started, and we do not have time to fix the system. Can you find the median of the data anyway?

Input

The input library is called "median"; see the sample inputs below for examples in your language. It defines two methods:

- **GetN()**:
  - Takes no argument.
  - Returns a 32-bit integer.
  - Expect each call to take 0.1 microseconds.
- **GetData(i)**:
  - Takes exactly one 64-bit integer argument: an index i, $0 \le i \le 10^{18}$.
  - Returns a 32-bit integer: the number that is i places away clockwise from the node's fixed point, as described in the problem statement.
  - Expect each call to take 0.1 microseconds.

The data is generated as follows: values for N, a data set of N integers X_0, X_1, ..., X_{N-1}, and 100 "delta" integers d_0, d_1, ..., d_{99} are chosen by the test setter. Then, a permutation P of the integers 0 through N-1 is chosen uniformly at (pseudo)-random*; all nodes use the same values of N, Xs, ds and P.
GetN() returns N in the Small dataset and -1 in the Large dataset, on all nodes.
GetValue(i) on node j returns $X_{P[(i + d_j) \% N]}$.

*: for technical reasons, the randomness used is weaker than something like `std::random_shuffle` in C++ or `java.util.Collections.shuffle` in Java. For transparency, this is the exact procedure used to retrieve the i-th (0-based) element out of N of a random permutation (you do not necesarilly need to fully understand this to solve the problem):

```
int index = i;
do {
   int MASK = (power == 32) ? ~0 : ((1 << power) - 1);
   index += add;
   index *= m1;
   index &= MASK;
   index ^= index >> ((power / 2) + 1);
   index += add2;
   index *= m2;
   index &= MASK;
   index ^= index >> (2 * power / 3);
} while (index >= N);
return index;
```

where `power` is ceil($\log_2$(N)), and `add`, `add2`, `m1` and `m2` are distinct constants between 0 and N - 1, inclusive, such that `m1` and `m2` are odd.

Output

Output one line with a single integer: the median of the data.

Limits

Number of nodes: 100 **(for both the Small and Large datasets)**.
Time limit: 7 seconds.
Memory limit per node: 32 MB. **(Notice this is less than usual.)**
Maximum number of messages a single node can send: 5000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le X_i \le 10^9$, for all i.
N % 2 = 1. (N is odd.)
$1 \le N < 10^9$

Small dataset

$d_i = 0$, for all i. (The nodes all indexed relative to the same original fixed point.)

Large dataset

$0 \le d_i < N$, for all i.

Sample

Input	Output
See input files below.	For sample input 1: 6 For sample input 2: 1000000000 For sample input 3: 1

The sample input has access to GetN() for all cases, but contains values of d_i other than 0. To test the Small dataset, you can edit the files to have all d_i equal to 0 (the answer is of course the same). To test the Large, you can just not use the GetN() function.
Sample input libraries:
Sample input for test 1: median.h [CPP] median.java [Java]
Sample input for test 2: median.h [CPP] median.java [Java]
Sample input for test 3: median.h [CPP] median.java [Java]


Powered by
Google Cloud Platform

Problem E. lispp3

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 11 points *2 minute timeout*	The contest is finished.
large 22 points *10 minute timeout*	The contest is finished.

Problem

Lisp+++

After a year of parenthetical bliss using Lisp++, Alyssa thought it was finally time to go beyond the parentheses. She decided to create an extension of the language that adds the + operator to simplify the semantics. She fittingly named the new language Lisp+++.

A valid Lisp+++ program consists of one of the following. (In this specification, *P* stands for some valid program -- not necessarily the same program each time.)

- `()` Literally, just an opening parenthesis and a closing parenthesis.
- *PP* Two programs (not necessarily the same), back to back.
- (*P*+*P*) A pair of parentheses enclosing two programs (not necessarily the same), separated by a + character.

Alyssa's new Lisp+++ requires a compiler that must be able to evaluate a string consisting of `(`, `)`, and/or + characters and determine whether it is a valid Lisp+++ program, and provide the user with some helpful information if it is not. If the program is valid, the compiler should print -1. Otherwise, it should print the length of the longest prefix that could be extended into a valid program by adding one or more additional characters to the end. If that prefix is the empty prefix, the compiler should print 0. In particular, if the input string is not a valid program, but can be extended to a valid program, the compiler should print the length of the input string.

For example:

- `(()+())()` is a valid program, so the compiler should print -1.
- `(()()+)` is not a valid program. The prefix `(()()+` is the longest prefix that is or could be extended into a valid program; in this case, one possible valid extension is `(()()+())`. So, the compiler should print 6. The only longer prefix is `(()()+)` (i.e., the entire string), but there is no way to add (any number of) characters to the end of that string to make it into a valid program.
- `)` is not a valid program. The prefix `)` cannot be extended into a valid program. The empty prefix is not a valid program, but it can easily be extended into one (by adding `()`, for example). So the compiler should print 0.

Given a string, what should Alyssa's Lisp+++ compiler print?

Input

The input library is called "lispp3"; see the sample inputs below for examples in your language. It defines two methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 32-bit integer: the length of the string.
  - Expect each call to take 0.08 microseconds.
- **GetCharacter(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetLength().
  - Returns a character `(`, `)`, or +: the character at position i, counting from left to right.
  - Expect each call to take 0.08 microseconds.

Output

Output a single line with a single integer: what the compiler should print, as described above.

Limits

Number of nodes: 100 **(for both the Small and Large datasets)**.
Time limit: 5 seconds.
Memory limit per node: 128 MB.

Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 128 KB. **(Notice this is less than usual.)**
$1 \le$ GetLength() $\le 10^9$. GetCharacter(i) is always (,), or +.

Small input

number of i such that GetCharacter(i) = + ≤ 100. (There are at most 100 +s in the input.)

Large input

No additional limits. (There may be more than 100 +s in the input.)

Sample

Input	Output
See input files below.	For sample input 1: -1 For sample input 2: 6 For sample input 3: 0

The sample input files are the same examples from the problem statement, in the same order.

Sample input libraries:
Sample input for test 1: lispp3.h [CPP] lispp3.java [Java]
Sample input for test 2: lispp3.h [CPP] lispp3.java [Java]
Sample input for test 3: lispp3.h [CPP] lispp3.java [Java]