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.

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 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]

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 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 ≤ r < GetRows(), 0 ≤ 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 ≤ GetRows() ≤ 30,000.
1 ≤ GetColumns() ≤ 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]

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 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.
$1 \leq X_i \leq 10^9$, for all i.
N % 2 = 1. (N is odd.)
$1 \leq N < 10^9$

Small dataset

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

Large dataset

$0 \leq d_i < N$, 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]

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 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 ≤ 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.)**
$1 \leq$ GetLength() $\leq 10^9$. 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]