# A recent visit to the Exploratorium at Pier 15 displayed the following exhibit:

# When you use a simple rule to move through a group of (positive integer) number blocks, you’re likely to end up on the same block every time.

Consider the following group of number blocks represented by an array of ints.

int[] numbers = {2, 1, 3, 5, 2, 7, 12, 3, 2,

5, 5, 3, 6, 4, 2, 1, 7, 3, 5, 4, 6};

Pick a starting index k, 0 <= k < numbers.length. If k = 2, is chosen you start with the value at index 2 (numbers[2] = 3) and move that many indices (new index = 2 + 3 = 5). Repeat this process until the next move will move you to an index greater than or equal to the size of the array.

For the array numbers and starting index k = 2, the above process will visit the following indices: 2, 2+3=5, 5+7=12, 12+6=18, 18+5 = 23 which is too big, therefore the last index is 18.

In this program you are to complete two methods in the SoonerOrLater class. The two methods are the visit and the itBeginsHere method. The SoonerOrLater class has a single constructor with a single int[] parameter representing the group of number blocks.

public class SoonerOrLater

{

private int[] nums;

/\*

\* precondition: 0 <= start < nums.length

\* nums.length > 0

\* nums[k] > 0 for all k, 0 <= k < nums.length

\*/

public SoonerOrLater(int[] n) { nums = n; }

/\*

\* post condition: returns the largest int n

\* such that n = start,

\* while n < num.length, n = n + nums[n]

\*/

public int visit(int start) { /\* Add code here \*/ return -1; }

/\*

\* post condition: returns the largest int n, n >= 0

\* such that visit(k) returns the same value for k <= n

\*/

public int itBeginsHere() { /\* Add code here \*/ return -1; }

}

The visit() method returns the largest int n such that,

starting with n = start, then while n < num.length, set n = n + nums[n].

The following code shows the results of the visit method.

|  |  |
| --- | --- |
| The following code | Returns |
| int[] n = {2, 1, 3, 5, 2, 7, 12, 3, 2, 5, 5, 3, 6,  4, 2, 1, 7, 3, 5, 4, 6};  SoonerOrLater sl = new SoonerOrLater(n); |  |
| sl.visit(0) | 18 |
| sl.visit(1) | 18 |
| sl.visit(2) | 18 |
| sl.visit(3) | 16 |
| sl.visit(4) | 18 |
| sl.visit(5) | 18 |
| sl.visit(6) | 18 |
| sl.visit(7) | 16 |
| sl.visit(8) | 16 |
| sl.visit(9) | 16 |
| sl.visit(10) | 16 |
| sl.visit(11) | 16 |
| sl.visit(12) | 18 |
| sl.visit(13) | 20 |
| sl.visit(14) | 16 |
| sl.visit(15) | 16 |
| sl.visit(16) | 16 |
| sl.visit(17) | 20 |
| sl.visit(18) | 18 |
| sl.visit(19) | 19 |
| sl.visit(20) | 20 |

The theme of this Sooner or Later exhibit is, with enough numbers, when you use this simple rule to move through a numbers, you will end on the same number. The itBeginsHere() method searches for the last location (index) where you end on the same number.

Stated in technical notation, the itBeginsHere() method returns the largest int n, n >= 0 such that visit(k) returns the same value for k <= n.

That is, the itBeginsHere()method searches for the largest index, n, in nums with visit(n) == visit[n-1]. This implies, let x = itBeginsHere(),

* visit(x) != visit(x+1)
* and if x > 0, then visit(x) == visit(x-1) for all x > 0.

The following code shows the results of the itBeginsHere method.

|  |  |
| --- | --- |
| The following code | Returns |
| int[] n = {2, 1, 3, 5, 2, 7, 12, 3, 2, 5, 5, 3, 6,  4, 2, 1, 7, 3, 5, 4, 6};  SoonerOrLater sl = new SoonerOrLater(n); |  |
| sl.itBeginsHere()   * Since visit(n) == same value for all value for n <= 2 | 2 |