# Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
  - Name the Clones
  - The Race of the Clones
  - Partial Palindrome
  - Growth Curve
  - The Family Tree of Mr C
  - Timely Tasks
  - Plenty of Palindromes
  - Count and Say Sequence
  - Orbiting Indices
  - Zig-zag Numbers
  - Parent Palindrome
  - Leaderboard
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

# Orbiting Indices

## LAB-PRAC-11_FUN-PTR

**Orbiting Indices [20 marks]**

---

**Problem Statement**
The first line of the input will give you a strictly positive number n. The next line of the input will give you a list of n non-negative integers, separated by a space. Store these in an array, say arr. We will play a game with this array. Say our starting index is j. Then in the next step, we will walk arr[j] + 1 steps to the right of the array. If we reach the end of the array while walking to the right, we simply wrap around and start walking from index 0. This is the first round of the game.

Say the index we land after following the above procedure is k. Then we repeat the above process again for another round, this time moving arr[k] + 1 steps to the right and wrapping around if we ever reach the end of the array. If following this process again and again, if we ever reach the original, starting index j, then index j is called an *orbiting index* since we keep coming back to that index again and again when playing this game.

In your output, print, for each of the n indices of the array, if that index is an orbiting index or not. If the index is not an orbiting index, print -1 else print the number of minimum number of rounds required to reach that index from the index itself.

**Caution**

1. This problem will not necessarily benefit from recursive formulations but you should write your program using functions to make it neat and easy-to-debug.
2. The integers in the list will be non-negative but can be zero.
3. Be careful about spelling and capitalization errors.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

---

**EXAMPLE 1**:
INPUT
4
1 1 1 1

OUTPUT:
2
2
2
2

**Explanation**: All the indices are orbiting indices. The orbits for the different indices are given below. All orbits are of length 2.

1. 0 => 2 => 0
2. 1 => 3 => 1
3. 2 => 0 => 2
4. 3 => 1 => 3

**EXAMPLE 2**:
INPUT
4
0 0 0 2

OUTPUT:
-1
-1
2
2

**Explanation**: The first two indices are not orbiting indices - starting from them causes us to get stuck inside a (3 => 2 => 3) loop and we are never able to reach the original indices themselves. The orbits for last two indices are given below. Both orbits are of length 2.

1. 2 => 3 => 2
2. 3 => 2 => 3

---

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## Start Solving! (/editor/practice/6223)