Write C program for the below scenarios:

1. An n×n two-dimensional array with elements already arranged in a non-decreasing serpentile order. The arrangement for a 4×4 two-dimensional array is shown in Fig. 1. The element present at index [0][0] is the minimum and that at index [n-1][n-1] is the maximum. If one traverses this array starting from [0][0] and follows the marked arrow directions then the result will be a non-decreasing sorted sequence of elements. In this sequence, the pth largest element gets positioned at p. Taking this p as an input from the user, write an efficient algorithm to print the row and column indices along with the value of the pth largest element in the sorted sequence.

   Click Fig. 1 to view the image. In case link does not work one can refer the figure shown below.

1 ↗	2 ↙	6 ↗	7 ↙
3 ↙	5 ↗	8 ↙	13 ↗
4 ↗	9 ↙	12 ↗	14 ↙
10 ↙	11 ↗	15 ↙	16 ↗

Input Format

- Line 1 contains an integer N, representing the dimension of an N × N array.
- Line 2 contains N×N space separated array elements in row major order.
- Line 3 contains an integer P, position of the pth largest element in non-decreasing sorted sequence.

Constraints

- All inputs range in between 1 and 1000.

Output Format

- Line 1 displays three space separated integers, row index of the pth element, column index of the pth element, and the value of the pth element.

Sample Input 0

```
4
124 169 505 563 224 278 585 803 251 638 690 934 664 684 944 981
9
```

Sample Output 0

```
2 1 638
```

Sample Input 1

```
5
1 2 6 7 15 3 5 8 14 16 4 9 13 17 22 10 12 18 21 23 11 19 20 24 25
23
```

Sample Output 1

```
3 4 23
```

---

2. **GCD** of two numbers is the largest positive integer that divides both numbers without leaving a remainder.
   Example: **GCD** (36,48) = 12; **GCD** (56,98) = 14; **GCD** (72,120) = 24

   The **Fibonacci series** is a sequence of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1. So, the Fibonacci series begins like this **0, 1, 1, 2, 3, 5, 8, 13, 21, 34**, and so on.

Problem Statement

Write a C program for the following operations:

- **G m n**: Computes **GCD** of numbers **m** and **n** using a recursive function. *Note*: here **G**, **m** and **n** are space separated.
- **F n**: Prints first **n** terms of the **Fibonacci series** using a recursive function. Note: here **F** and **n** are space separated.

Input Format

- Line 1 contains an integer **N**, the number of queries.
- The following **N** lines contain queries in the format as mentioned above.

Constraints

- All inputs range in between 1 and 100.

Output Format

- The **N** lines in the output display the result after the execution of each query.

Sample Input 0:	**Sample Output 0:**	**Explanation:**
2 F 5 G 36 48	0 1 1 2 3 12	Here, the first line take 2 as # of input queries. Next line is F 5, i.e. print Fibonacci Series up to 5th term, so output is 0 1 1 2 3 The third line G 36 48 says calculate GCD (36,48)., so the output is 12

Sample Input 1:	**Sample Output 1:**	**Explanation:**
5 F 10 F 2 G 12 18 G 72 120 F 8	0 1 1 2 3 5 8 13 21 34 0 1 6 24 0 1 1 2 3 5 8 13	Here there are 5 queries. 1. F 10: Print Fibonacci series up to 10th term, 2. F 2: Print Fibonacci series up to 2nd term, 3. G 12 18: Calculate GCD (12,18), 4. G 72 120: Calculate GCD (72,120) 5. F 8: Print Fibonacci series up to 8th term In the output each line prints the answer to an individual query.

3. There is an integer array ***nums*** **already** sorted in ascending order with **n** values. One has to rotate ***nums*** to the left by **k** steps, where **k (1 <= k < n)** is a **non-negative number**, such that the resulting array is {***nums*[n-k]**, ***nums*[n-k+1]**, ..., ***nums*[n-1]**, ***nums*[0]**, ***nums*[1]**, ..., ***nums*[n-k-1]**}. Update the contents of an array using the following algorithm.

   Step 1: Reverse the given array ***nums***.
   Step 2: Reverse the first **k** elements of the array ***nums*** obtained in **Step 1**. **Step 3:** Reverse the last **n – k** elements of the array ***nums*** obtained in **Step 2**.

   For example, if ***nums*[0..9] = {1,2,3,4,5,6,7,8,9,10}** and **k = 2**, then the above algorithm updates ***nums***
   as {**9,10,1,2,3,4,5,6,7,8**}.

   Given the array ***nums*** after the possible rotation and an integer target, say **t**, return the index of the target (, i.e. **t**) if it is in ***nums***, otherwise **-1**.

Input Format

- Line 1 contains an integer **n**, the number of elements in an array.
- Line 2 contains **n** space separated array elements.
- Line 3 contains an integer **k**, the pivot index to rotate the array.
- Line 4 contains an integer **t**, the target element to be searched.

Constraints

- ***nums*** is an ascending array that is rotated by **k** steps.
- 1 <= **n** <= 105
- -231 <= ***nums*[i]** <= 230
- All values of array ***nums*** are unique
- 0 <= **k** <= **n**, else print "-1"
- -231 <= target <= 230, else print "-1"

If input values are not satisfying any of the constraints, print -1.

Output Format

- Line 1 displays the space separated updated contents of ***nums*** after rotation.
- Line 2 displays the index of the target element that is searched.

Sample Input 0:
7
1 2 3 3 5 6 7
3
3
Sample Output 0:
-1

Sample Input 1
8
1 2 3 4 5 6 7 8
3
5
Sample Output 1:
6 7 8 1 2 3 4 5
7

---

4. A self-dividing number is a number that is divisible by every digit it contains.
   For example, 128 is a self-dividing number because 128 % 1 == 0, 128 % 2 == 0, and 128 % 8 == 0.
   A self-dividing number is not allowed to contain the digit zero.
   Given two integers left and right, return a list of all the self-dividing numbers in the range [left, right]

   EXAMPLE:

Example 1:
Input: left = 1, right = 22
Output: [1,2,3,4,5,6,7,8,9,11,12,15,22]
Example 2:
Input: left = 47, right = 85
Output: [48,55,66,77]
Constraints:
1 <= left <= right <= 104Write a program to extract all digits from a given alphanumeric string and store them in a separate string.

Example: Input: "abc123xyz" Output: "123".

---

5. You are given an integer *N*. Can you find the least positive integer *X* made up of only 9's and 0's, such that X is a multiple of *N*?

Update

- *X* is made up of one or more occurrences of 9 and zero or more occurrences of 0.

Input Format:

- The first line contains an integer T which denotes the number of test cases. T lines follow.
  Each line contains the integer *N* for which the solution has to be found.

Output Format:

- Print the answer *X* to STDOUT corresponding to each test case. The output should not contain any leading zeros.

Constraints:

- 1 <= T <= 10^4
  1 <= N <= 500
- 0 should not be in the lead

Input Sample:

- 3
- 5
- 7
- 1

Output Sample:

- 90
- 9009
- 9

Explanation

- 90 is the smallest number made up of 9's and 0's divisible by 5. Similarly, you can derive for other cases.