## Question - 1
**Kinder Square**

Given a list of integers, return a list of numbers that after being squared and added 10, the last digit is not 5 or 6.
Complete the *kinderSquare* method. It receives an array of integers, **arr**. It returns a list of integers consisting of those elements of the original list that after being squared and added 10, their last digit is not 5 or 6.
Please note that the relative order of the elements in the output array must be the same order that the one in the input array.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***
The first line contains an integer, **n**, denoting the number of elements in the list.
Each of the *n* subsequent lines contains an integer, denoting the elements of **arr**.

**Constraints**
$1 \le \mathbf{n} < 2000$
$0 \le \mathbf{arr[i]} < 100$ for $0 \le i < n$

**Output Format**
The output will consist of the elements of the solution array, one per line.

**Sample Input 0**

```
3
3
1
4
```

**Sample Output 0**

```
19
11
```

**Explanation 0**
There are 3 elements in the list: 3, 1 and 4. Their squares are 9, 1 and 16, respectively. When they are added ten we get 19, 11 and 26 respectively. Given that 26 has 6 as its last digit, it won't be in the solution array. 19 and 11 do not end with either 5 or 6, so they will be in the solution array.
Note that the solution array is [19, 11]. The array [11, 19] is not a solution because it does not respect the relative order of the original elements.

## Question - 2
**Minefield**

You have to walk through a minefield and you know the steps you can take in order to not come across a mine. The field is mapped to a cartesian coordinate system and each step you could take is in a straight line, adding to one of the coordinates (**x** or **y**) of the standing point, the value of the other coordinate. This means that from point *(x, y)* you can go either to *(x + y, y)* or *(x, x + y)*. The idea is to see if it is possible to go from a starting point *($x_1$, $y_1$)* to an ending point *($x_2$, $y_2$)*.

For example, given the entry point *(2, 3)*, the next step could be *(5, 3)* or *(2, 5)*.
Determine if there is a mine free path from a given entry point to a given exit point of the field. Complete the *isPossible* method. It receives 4 integers, **x_start**, **y_start**, **x_end**, **y_end**. Those represent the starting and ending coordinates. The method must return the string **YES** if there is a path between the two points and **NO** otherwise.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***
The first line of the input will be an integer **numTestCases**. This number represents the number of instances the algorithm will run (how many pairs of coordinates will be tested).
Then, the next 4 line pattern is repeated *numTestCases* times.
- *x_start* coordinate for current test.
- y_*start* coordinate for current test.
- *x_end* coordinate for current test.
- *y_end* coordinate for current test.

**Constraints**
$2 \le numTestCases \le 1000$
$1 \le x\_start_i \le 5$
$1 \le y\_start_i \le 5$
$x\_start_i \le x\_end_i \le 500$
$y\_start_i \le y\_end_i \le 500$

**Output Format**
The output consists of *numTestCases* lines, the *ith* line being the string **YES** if there is a path between *($x\_start_i$, $y\_start_i$)* and *($x\_end_i$, $y\_end_i$)* and **NO** otherwise.

**Sample Input 0**

```
2
1
1
2
5
1
1
6
3
```

**Sample Output 0**

```
YES
NO
```

**Explanation 0**
There are 2 test cases. The first one starts at (1, 1) and needs to finish at (2, 5). The second one starts at (1, 1) and needs to finish at (6, 3).
For the first case, a valid path is moving to (2, 1), then to (2, 3) and finally to (2, 5). For the second case, there is no valid path.

## Question - 3
**Multiple Choice Panic**

Today is the history exam, and Bob didn't study for it! Alice, the professor, is renowned for her very hard and tricky multiple-choice type of exams. Bob will lose the whole semester if he fails this exam, so he is quite concerned about it. Being backed into a corner, he is tempted to answer the questions randomly, but only if the odds are in his favor.

The exam has **N** questions, and Bob needs **P** points to pass it. For each question, there is a tuple (**q**, **pc**, **pw**), in which:
- q: number of options of the multiple choice question
- pc: points gained if you answer this question correctly
- pw: points lost if you answer the question wrongly

Only one option can be selected per question (as there is only one right answer for each). Bob knows nothing about any of them so the best he can do is choose randomly.
Bob is not obliged to answer every question in the exam. If Bob does not answer a question, it yields 0 points.

Complete the *isExpectedToPassExam* method. It receives 5 arguments:
- **N***:* Number of questions
- **P***:* Score Bob needs to pass
- **q**: An array of *N* integers, $q_i$ being the number of options for question *i*
- **pc**: An array of *N* numbers, $pc_i$ being the points gained for answering question *i* correctly
- **pw**: An array of *N* numbers, $pw_i$ being the points lost for answering question *i* wrongly
It returns the string **YES** if the expected score of answering all questions randomly is *P* or higher, otherwise it returns **NO**.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***
The first line will contain an integer, **t**, denoting the number of test cases.
Then, the next 5 line pattern will be repeated *t* times:
- *N* for current test
- *P* for current test
- *q*, represented as numbers separated by whitespace
- *pc*, represented as numbers separated by whitespace
- *pw*, represented as numbers separated by whitespace

**Constraints**
$0 < t \leq 100$

*$1 < N \leq 100$*
*$1 \leq P$*
*$1 < q_i$*
*$1 \leq pc_i$*
*$0 \leq pw_i$*

**Output Format**
Output will consist of *t* lines. Line *i* will consist of the string **YES** if the solution of test *i* is yes, and **NO** otherwise.

**Sample Input 0**

```
2
2
1
2 2
2 2
1 1
3
2
2 4 2
1 4 1
0.5 1 1
```

**Sample Output 0**

```
YES
NO
```

**Explanation 0**
The first exam has two multiple choice questions. At least 1 point is needed to pass the exam. The 2 questions have the same composition. Each one has 2 choices and if answered correctly they both yield 2 points. If answered wrongly, they substract 1 point. The expected score when answering randomly is 1, which is equal to the minimum score needed to pass the exam, so answer is **YES**.
In the second exam, the expected score when answering randomly is 0.5, and the minimum score needed to pass is 2, so answer is **NO**.

## Question - 4
**OpenMateLab**

As part of an initiative to develop an open source alternative to "Matelab" (a numerical methods environment), we must implement the sequences module. Although most common sequences (such as Fibonacci) have been implemented, the team also included the Jaime Roos sequence, which is defined the following way:

$JR_0 = \mathbf{a}$
$JR_1 = \mathbf{b}$
$JR_2 = \mathbf{c}$
$JR_n = JR_{n-1} + JR_{n-2} + JR_{n-3}$
With *a, b* and *c* being variable and *$2 < n$.*

The team has been practicing TDD so they already wrote all the tests needed. We now need your help to get them to pass.

**Input Format**

***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***

Input will consist of 7 lines.

Line 1 will be the first term of the Fibonacci sequence, $Fib_0$

Line 2 will be the second term of the Fibonacci sequence, $Fib_1$

Line 3 will be the term of the Fibonacci sequence that needs to be calculated. (**i** such that we want $Fib_i$)

Line 4 will be the first term of the Jaime Roos sequence, $JR_0$

Line 5 will be the second term of the Jaime Roos sequence, $JR_1$

Line 6 will be the third term of the Jaime Roos sequence, $JR_2$

Line 7 will be the term of the Jaime Roos sequence that needs to be calculated (**j** such that we want $JR_j$)

**Constraints**

$0 \leq Fib_0 \leq 1000$

$1 \leq Fib1 \leq 1000$

$0 \leq i \leq 20$

$0 \leq JR_0 \leq 1000$

$1 \leq JR1 \leq 1000$

$1 \leq JR2 \leq 1000$

$0 \leq j \leq 20$

**Output Format**

Output will consist of two lines.

The first line will contain an integer denoting $Fib_i$

The second line will contain an integer denoting $JR_j$

**Sample Input 0**

```
0
1
5
1
1
1
6
```

**Sample Output 0**

```
5
17
```

**Explanation 0**

In this case, the first two terms of the Fibonacci sequence are 0 and 1.

So, $Fib_2$ will be 1, $Fib_3$ will be 2, $Fib_4$ will be 3 and $Fib_5$ will be 5. So, the 5th term of this Fibonacci sequence is 5. Then the first line of the output will be 5.

In this case, the first three terms of the Jaime Roos sequence are 1, 1 and 1.

So, $JR_3$ will be 3, $JR_4$ will be 5, $JR_5$ will be 9 and $JR_6$ will be 17. So, the 6th term of this Jaime Roos sequence is 17. Then the second line of the output will be 17.

## Question - 5

**Dishwashers Paradise**

At a local restaurant, Mark is a kitchen porter (the person that washes dishes) and he only works if the kitchen has any dirty dishes. The restaurant assigned him an assistant as he's really tired of washing in an inefficient way. Mark thought of a system that would help them finish faster:

- Organize dishes in stacks. Each dish is unique and has an identifier
- Each stack has the same maximum size
- When a dirty dish arrives, it goes at the top of the first stack that is not full
- Mark or the assistant always have to grab a dish from the top of the first stack

After a dish is washed, the assistant needs to organize the dirty dishes by having the maximum quantity of stacks full. To do this, starting from the first stack, the assistant needs to move a dish from the top of the next stack to the actual one until all fillable stacks are full. Note that the stacks closest to the sink must be full.
Besides, Mark may want to know the number of dishes on a specific stack.
The new assistant doesn't want to work, so you, the assistant's friend, are called to build a system that covers the functionality explained above. Complete the following methods:

- *add*. It receives an integer **element**, representing the id of the dish.
- *remove*. It does not receive any parameters and represents washing a dish. It must return an integer denoting the id of the removed dish. **Note**: There will only be calls to remove for dishes that have already been added.
- *count*. It receives an integer **stackIndex**. It must return an integer denoting the number of dishes in stack **stackIndex**, or -1 if such stack does not exist.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***
The first line contains an integer, **n**, denoting the maximum size of the stacks.
The second line contains an integer **q**, denoting the number of queries.
The next **q** lines contain queries. Queries can be:
- ADD i, where **i** is the id of a dish.
- REMOVE.
- COUNT j, where **j** is the id of a stack.

**Constraints**
*1 ≤ **n** ≤ 10*
*3 ≤ **q** ≤ 25000*
*0 ≤ **i** ≤ 100000*

**Output Format**
For each query of type COUNT or REMOVE, there must be a line with its output.

**Sample Input 0**

```
2
12
ADD 13
ADD 7
ADD 17
COUNT 0
COUNT 1
ADD 2
COUNT 1
```

```
ADD 47
REMOVE
COUNT 0
COUNT 1
COUNT 2
```

**Sample Output 0**

```
2
1
2
7
2
2
-1
```

**Explanation 0**

On the example, the first line tells you that the maximum size of each stack is 2.
Then you have 12 queries.
After the first 3 ADD queries, you will have:

```
STACK 0    STACK 1

| 7  |       |    |
| 13 |       | 17 |
```

So, COUNT 0 will output the size of stack 0, which is 2.
And, COUNT 1 will output the size of stack 1, which is 1.

Then ADD 2, results in:

```
STACK 0    STACK 1

| 7  |     | 2  |
| 13 |     | 17 |
```

And, COUNT 1 will output the size of stack 1, which is 2.

Then, as 2 is the maximum stack size, the next ADD results in:

```
STACK 0  STACK 1  STACK 2

| 7  |   | 2  |   |    |
| 13 |   | 17 |   | 47 |
```

REMOVE will remove dish number 7 (it outputs 7) from stack 0, and after organizing  the dishes, the stacks look like this:

```
STACK 0  STACK 1

| 2  |     | 47 |
| 13 |     | 17 |
```

Finally:
COUNT 0 will output 2, since there are 2 dishes in stack 0.
COUNT 1 will output 2, since there are 2 dishes in stack 1.
COUNT 2 will output -1, since there is no stack 2.

## Question - 6
**Freeing the Zoo**

**Problem Statement**

After 140 years as a popular tourist destination, the city zoo decided to free all the animals in captivity, moving them to a natural reserve.

To do this, the zoo hired a moving company that will send one truck to move two animals simultaneously. The truck has a fixed moving capacity that **must** equal the weight of two animals. The zoo wants the heaviest animal to be moved first.

Decide which two animals to move on the truck. Complete the *getAnimals* method. It receives an array of animal weights and the truck capacity and should return an array with two elements, containing the two weights of the animals that are going to be moved. In case no animals can be moved with the given capacity, return an empty array. We want to do this as quickly as possible, because the zoo doesn't want to waste the truck driver time.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***
The first line contains an integer **n**, denoting the quantity of animals in the array.
The second line contains *n* integers, separated by whitespace, denoting the animal weights array.
The third line contains an integer denoting the truck capacity.

**Constraints**
$0 < \mathbf{n} \leq 10^6$

**Output Format**
The first line of the output contains the length of the solution array, **s**.
The second line contains *s* integers, separated by whitespace, denoting the solution array.

**Sample Input 0**

```
6
1 5 2 7 9 3
10
```

**Possible Sample Output 0**

```
2
1 9
```

**Explanation 0**
There are two pairs of weights that sum 10. Those would be [7, 3] and [1, 9]. Since we need to move the heaviest animal first, we set the answer to [1, 9] because 9 > 7 > 3 > 1

**Sample Input 1**

```
6
1 5 2 7 9 3
2
```

**Possible Sample Output 1**

```
0
```

**Explanation 1**
We need to get two animals whose weights sum 2. We can see there is no pair that fullfills that. The answer is [].

## Question - 7
**Hermione**

Hermione is studying in Hogwarts to become a magician. She has an exam next week so she goes to the library to find a specific book to study.
Each book has a *unique* identification number, called **ID**. In the library the books are ordered by ID, but today a naughty student, Draco Malfoy, mixed the books a little.
The books are now swapped in such a way that each book is at most 1 position away from the original (the books are *nearly* sorted).
This means each book can end being in the original position, one to the left, or one to the right. position(ID) = oldPosition(ID) - 1 | oldPosition(ID) | oldPosition(ID) + 1

You have to help Hermione find the new position of the book ID that she is looking for. But she doesn't have much time to study, so she has to find the book **as quickly as possible**.
Have in mind that the book Hermione is looking for is always present. And the first and the last books of the collection could only be swapped in one direction (the first book **can't** be swapped with the last one)

Complete the function *findBookPosition*. It receives an array of book IDs, **bookIDList** (array of IDs with the order they have in the library now) and an integer **k**, which is the book Hermione is looking for. It must return an integer representing the index of the book with ID *k* in the unsorted library. The book will always be present.

**Input Format**
***Input is already parsed by us. The format is only important if you want to use the 'Custom input' option.***

The first line contains an integer **t**, the number of test cases.
For the next **t** cases:
  The first line contains an integer ***n***, the size of ***bookIDList.***
  The second line contains **n** separated integers, denoting the elements of the array ***bookIDList***.
  The third line contains an integer ***k***, the book ID Hermione is looking for.

**Constraints**
$6 \leq \boldsymbol{t} \leq 30$
$50 \leq \boldsymbol{n} \leq 10^4$

**Output Format**
The output will contain **t** lines. The *ith* line will contain the position of book **k** in test *i* and another integer separated with a whitespace, which is for internal use. **Don't** worry about it.

**Sample Input 0**

```
1
8
2 1 4 3 15 20 27 31
4
```

**Sample Output 0**

```
2 4
```

**Explanation 0**
Given the array of IDs, *bookIDList* = [2, 1, 4, 3, 15, 20, 27, 31] and that Hermione is looking for the book with ID, *k* = 4.
The original order in the library of the IDs was [1, 2, 3, 4, 15, 20, 27, 31], because they were sorted in increasing order.
The book ID 4 was at original index 3, but now is at index 2. So the first integer is 2.
The second integer, 4, is for internal checks. **Don't** worry about it.