Instructions:

- Arrays are indexed from 1 to *n* unless otherwise specified, to follow the convention of the textbook. This means when coding, arrays must be of actual size $n + 1$. The 0^{th} cell is ignored. This is why in the test cases there is a "dummy" row 0, and "dummy" values in cell 0 in each row (listed as "NA"). Do not use the 0^{th} row or 0^{th} column, unless the algorithm or problem specifically asks it.
- **YOU MAY NOT IMPORT ANY LIBRARIES!**
- **YOU MAY NOT USE ANY ARRAY METHODS!**

Problem 1 (8 pts) Write the method below which takes an *n* by *n*, two-dimensional array *A* (rows and columns indexed from 1 to *n*) and determines if some value appears more than once in *A*. If so, the method returns an array with the coordinates of both locations.

```
int [] dupeLocations(int n, int [][] A)
```

For example, If *A* is the array below, the method would return the array `{2, 5, 4, 3}` since $A[2][5] = A[4][3]$. However, if $A[2][5]$ were 1000 instead of 999, then *A* would have no duplicate elements, and the array returned would be `{-1}`.

A	1	2	3	4	5
1	68	150	77	2	83
2	41	69	31	80	**999**
3	29	57	24	11	432
4	46	-32	**999**	112	499
5	119	100	298	34	7

Problem 2 (8 pts) Write the method which determines if *n* by *n*, two-dimensional array *A* has two identical rows.

```
int [] sameRows(int n, int [][] A)
```

For example, the call `sameRows(4, A1)` would return the array `{-1}` since none of the rows are identical in every column. The call `sameRows(4, A2)` would return the `{2, 4}`, since rows 2 and 4 are identical in every column:

A1	1	2	3	4
1	7	4	9	6
2	7	4	7	6
3	4	4	9	6
4	7	4	9	8

A2	1	2	3	4
1	7	4	9	6
2	7	4	7	6
3	4	4	9	6
4	7	4	7	6

Problem 3 (8 pts) For this problem, you ***may NOT declare arrays***.

Write the method below which takes an integer *n* and an array *A* (indexed from 1 to *n*) and determines if it is a permutation of the set $\{1, 2, \ldots, n\}$. In other words, the method returns true if each value in the set $\{1, 2, \ldots, n\}$ is located somewhere in the array:

```
boolean isPermutation(int n, int [] A)
```

For example, the array below is not a permutation with $n = 4$ because the value 2 is missing and because the value 3 appears twice:

1	2	3	4
3	1	4	3

The following array is not a permutation with $n = 7$ because the value -90 is out of range, 16 is out of range, and the numbers 3 and 7 are missing:

1	2	3	4	5	6	7
2	1	-90	6	4	16	5

But the following array is a permutation with $n = 7$:

1	2	3	4	5	6	7
7	3	2	6	1	5	4

Problem 4 (8 pts) Write the method below which takes an array *A* (indexed from 1 to *n*) and an integer *g*. This method determines if there are three distinct indices *i*, *j*, and *k* such that $A[i] + A[j] + A[k] = g$. If there are, it returns the array `{i, j, k}`. If not, it returns the array `{-1}`.

For example, if passed array *A* below with *g* = 96, it would return the array `{2, 5, 7}`, since $A[2] + A[5] + A[7] = 8 + 37 + 51 = 96$. However, for the same array with *g* = 101, it would return the array `{-1}` since no three distinct elements in *A* sum to 101.

	1	2	3	4	5	6	7	8
A	5	8	16	22	37	44	51	82

```
int [] sumOfThree(int n, int [] A, int g)
```

Problem 5 (8 pts) You may use Strings and any String methods you want for this problem. Write the method `validBoard` which takes positive integer *n* and two-dimensional array *grid* with *n* rows and *n* colums, containing 0's and 1's:

```
String validBoard(int n, int [][] A)   {. . .}
```

This method does the following:

1. If a row contains no 1's or two or more 1's, return a String that says which row contains no 1's or which row contains two or more 1's, such as:

   `"Row 5 has no 1's."` or
   `"Row 5 has two or more 1's."`

2. Otherwise, if the same is true for a column, return a String that says which column contains no 1's or which column contains two or more 1's, such as:

   `"Column 7 has no 1's."` or
   `"Column 7 has two or more 1's."`

3. Otherwise, if two 1's lie in the same diagonal (which can be in either direction), return a String that give the two rows of the 1's that lie in the same diagonal, such as: rows, as follows:

   `"The 1's in rows 5 and 8 are in the same diagonal."`

4. Otherwise, simply return the string: `"VALID BOARD."`

Finding the DIAGONAL conflicts is worth half of the points for this problem!

The output for the first six tests should look like this:

```
Test 1:  VALID BOARD.

Test 2:  Row 6 has two or more 1's.

Test 3:  Row 5 has no 1's.

Test 4:  Column 2 has two or more 1's

Test 5:  The 1's in rows 4 and 8 are in the same diagonal.

Test 6:  The 1's in rows 3 and 7 are in the same diagonal.
```

Problem 6 (8 pts) Write the method below which takes an integer $n \geq 2$ and prints its prime factorization:

```
void printPrimeFactorization(int n)
```

Reminder: Every integer $n \geq 2$ can be written as the product of prime numbers (numbers divisible by 1 and itself only). For example, $720 = 2 \times 2 \times 2 \times 2 \times 3 \times 3 \times 5 = 2^4 \times 3^2 \times 5^1$.

The call `printPrimeFactorization(720)` would result in the output: `2^4 * 3^2 * 5^1`.

For this problem, you ***may not call other methods.***