1. *Cutting Board*. Given a rectangular board of size ***m*** by ***n***, you are to cut it into as many squares (of side length ***a***) as possible. You may assume that ***m***, ***n***, and ***a*** are positive integers.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of three integers for a test case, ***m***, ***n***, and ***a*** as defined above (separated with a whitespace).

- Example:

```
4
3 5 1
3 5 3
3 5 2
3 8 2
```

Output: ***k*** lines of output, each of which is a single integer (for each test case) representing the maximum number of cut squares (i.e., those having the side of ***a***) for the input in the test case defined above.

- Example (the output for the above inputs):

```
15
1
2
4
```

2. *Cutting Board II*. Given a rectangular board of size ***m*** by ***n***, you are to cut it into as many rectangles (having the size ***a*** by ***b***) as possible. You may assume that ***m***, ***n***, ***a***, and ***b*** are positive integers.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of four integers for a test case, ***m***, ***n***, ***a***, and ***b*** (separated with a whitespace).

- Example:

```
6
2 4 1 1
2 4 1 2
2 4 2 1
3 5 2 1
3 5 1 2
3 5 5 4
```

Output: ***k*** lines of output, each of which is a single integer (for each test case) representing the maximum number of cut rectangles (i.e., those having the dimension of ***a*** by ***b***) for the input in the test case defined above.

- Example (the output for the above inputs):

```
8
4
4
7
7
0
```

3. *Switch On/Off*. In the Arts Building, there is a long exhibition corridor and on the wall there are ***m*** colored light bulbs from the entrance to exit, each of which can be turned on and off by a switch. These switches are initially turned off. There are ***n*** visitors walking through the corridor. The first visitor toggles (turns on) each switch; the second one toggles (turns off) every second switch (i.e., the 2nd, 4th,…will be turned off). Continuing this way, the ***i***-th visitor toggles every ***i***-th switch (i.e., if it is on, then turns it off; and vice versa) from the entrance. Your job is to calculate how many switches are still on after all the visitors passing through the corridor.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of two integers for a test case, ***m***, ***n*** (separated with a whitespace). You may assume that ***m*** and ***n*** are positive numbers and ***m*** >= ***n***.

- Example:

```
8
4 1
4 2
4 3
4 4
25 24
25 25
500 500
500 250
```

Output: ***k*** lines of output, each of which is a single integer (for each test case) representing the number of switches which are still on for the inputs in the test cases.

- Example (the output for the above inputs):

```
4
2
1
2
4
5
22
258
```

4. *Anagrams*. An anagram is a type of word play, the result of rearranging the letters of a word to produce a new word, using all the original letters exactly once. The following list some anagrams, “dormitory” and “dirtyroom”, “debitcard” and “badcredit”, “mummy” and “mymum”, and “orchestra” and “carthorse”. You are to write a Java program to detect whether a pair of two words is an anagram.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of two strings for a test case, ***s***, ***t*** (separated with a whitespace). You may assume that ***s*** and ***t*** consist of only English lowercase letters.

- Example:

```
8
aaaab   baaaa
aaaab   baaa
agentleman  elegantman
hotwater  worthtea
theeyes   theysee
astronomers  nomorestars
astronomers  nomorestar
dirtyroom   dormitory
```

Output: ***k*** lines of output, each of which is a single word “Yes” (for an anagram) or “No” (if not an anagram) for each test case as defined above.

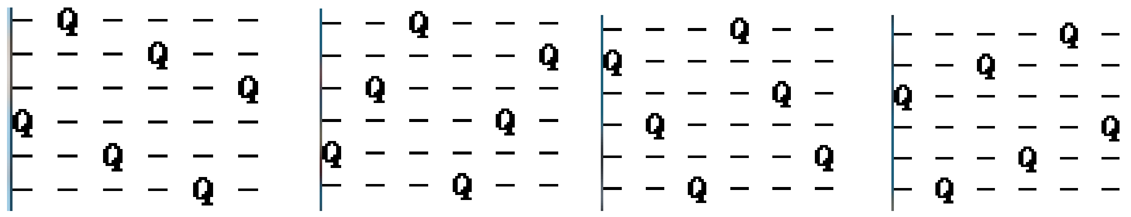
- Example (the output for the above inputs):

```
Yes
No
Yes
Yes
Yes
Yes
No
Yes
```

Note: You may use class ***Scanner***’s method ***next***() to read a string/text/word.

5. *N_Queens Plus Puzzle*. The N-Queens Plus puzzle is like the N-Queens puzzle: it poses the problem of placing one chess queen on an ***N***-by-***N*** chessboard and placing the remaining ***N -1*** queens on the chessboard so that no two queens (of these ***N*** queens) attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The following gives all possible different placements for a 6-Queens puzzle (a total of 4). You job is to calculate the number of solutions for *N-Queens Plus puzzle* given ***N*** and the location of the first queen, ***qRow*** and ***qCol***.

- We use row and column numbers (from ***0*** to ***N-1***) to describe the locations of the queens. The bottom-left location is starting grid, row 0 and column 0. The top-right is the row ***N-1*** and column ***N-1***.
- Thus, the leftmost of the following 6-Queens Puzzle solutions serves as the solution for 6-Queens Plus when (***qRow***, ***qCol***) is (0, 4), (1, 2), (2, 0), (3, 5), (4, 3), and (5, 1).
- There are no solutions for 6-Queens Plus puzzle for (***qRow***, ***qCol***) taking (3, 3). It is possible that there are many solutions for an N-Queens Plus puzzle (for a given (***qRow***, ***qCol***)).



Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of three integer ***N***, ***qRow***, ***qCol*** defined above. You may assume that ***N*** is a positive number and that ***N*** < 15, 0 <= ***qRow*** < ***N***, and 0 <= ***qCol*** < ***N***.

- Example:

```
6
6 0 4
6 4 0
6 3 3
7 0 0
7 5 6
4 0 0
```

Output: ***k*** lines of output, each of which is a single integer (for each test case) representing the number of solutions for the input in the test case defined above.

- Example (the output for the above inputs):

```
1
1
0
4
7
0
```

6. *Gold Coins*. There are ***n*** gold coins worth *different* values (dollars), $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, …, $\boldsymbol{v_n}$. We want to select ***m*** coins that will have a total value of ***s*** dollars. For example, given ***n*** = 9 coins of values 1, 2, 3, 5, 7, 8, 10, 11, and 13 (dollars), if we are to take ***m*** = 3 coins with a total value of ***s*** = 18 dollars, we will have five different possible ways to do this:

(1 + 7 + 10)   (2 + 3 + 13)   (2 + 5 +11)   (3 + 5 + 10)   (3 + 7 + 8)

If we take ***m*** = 4 coins, there would be three solutions:

(1 + 2 + 5 + 10)   (1 + 2 + 7 + 8)   (2 + 3 + 5 + 8)

If we take ***m*** = 5 coins, there would be only one solution: (1 + 2 + 3 + 5 + 7).

Your job is to write a Java program to find the number of solutions.

Input: the first line gives an integer ***k***, which is the number of test cases and is followed by ***k*** test-case-inputs. Each test-case-input has same format of inputs given in 3 lines:

- A line of two integers, ***s*** and ***m***, as defined above (***s*** <= 500; ***m*** <= 50).
- A line of one integer, ***n***, (the number of gold coins, ***n*** <= 100).
- A line of ***n*** integers (the values of the ***n*** coins, $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, …, $\boldsymbol{v_n}$, which are positive integers, and which are distinct and given in *increasing* order).

Example:

```
5
10 3
9
1 2 3 5 7 8 10 11 13
10 4
9
1 2 3 5 7 8 10 11 13
18 3
9
1 2 3 5 7 8 10 11 13
18 4
9
1 2 3 5 7 8 10 11 13
20 3
10
1 2 4 5 10 11 13 15 17 19
```

Output: ***k*** lines of output, each of which is a single *long* integer (for each test case) representing the number of solutions for the input in the test case defined above (*long* is one of Java's primitive data types. The number of solutions for a certain test case may be too large for a regular *int* number to store).

- Example (the output for the above inputs):

```
2
0
5
3
4
```