Question 1: PrimeTriplets

Problem Statement

Given a list of integers, determine ***how many unique combinations*** of three elements from the list that produce a prime product.

Using external `is_Prime()` method is **NOT allowed**. If you decide to have any prime check mechanism in your code, it has to be manually implemented.

Method Signature

Java

```
public  class  Question1 {
        public  static  int     countPrimeTriplet(List<Long> n) {
                // your code here
        }
}
```

Cpp

```
class Question1 {
public:
    static int countPrimeTriplet(vector<long long>& n) {
        // your code here
    }
};
```

Python

```
from typing import List

class Question1:
    @staticmethod
    def countPrimeTriplet(n: List[int]) -> int:
        # your code here
```

Input

- `n` is a list of long integers where `1 ≤ len(n) ≤ 25`.

Output

- Return an integer - the count of unique combinations of three distinct elements in the list whose product is a prime number.

Constraints

- `1 ≤ len(n) ≤ 25`
- `1 ≤ n[i] ≤ 10^19`

Test Cases

Test Case #	Input	Expected Output
0	[1, 3, 5, 7, 11, 13, 17, 1, 19, 23, 29, 31]	10
1	[2, 3, 4289, 5, 7, 1021, 11, 3019, 967, 17, 923, 19, 29, 31]	0
2	[1, 3, 5, 4673, 7, 1217, 11, 13, 1, 2543, 17, 19, 23, 29, 821]	
3	[2, 3, 4643, 5, 7, 9311, 1171, 2027, 11, 13, 17, 19, 2473, 2129]	
4	[1, 2347, 1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 7481, 5647, 31]	
5	[2, 3, 4129, 5, 7, 1399, 4327, 8713, 11, 13, 17, 19, 23, 9983, 8971]	
6	[1, 3, 723226807416245459, 2, 739972444344274363, 7, 5, 11, 13, 169917336923781151, 17, 19, 987223004210710331, 376834422262171399, 23, 29, 1, 31, 37]	

Test Case #	Input	Expected Output
7	[2, 294589784991461689, 3, 359087329106418229, 5, 7, 565571075242943507, 11, 13, 920970999126726979, 17, 19, 710194101483294751, 970093981381389251, 556348586753427287, 23, 29, 31, 37, 41]	
8	[561325539269875847, 467553833883105737, 174545849082960241, 818310483624095489, 2, 3, 5, 7, 11, 13, 17, 19, 23, 1, 29, 428193534162129107, 954243130743585217, 31, 37, 1, 41]	
9	[3, 380597290666960121, 5, 7, 876025983958399961, 11, 13, 254379103870810549, 17, 19, 924332569386582269, 491830154963583209, 23, 29, 1, 31, 37, 198714392410133029, 1, 41, 43]	
10	[542199767432699813, 777802938716202353, 212811997458829949, 633783071857165753, 533313557535144863, 2, 3, 5, 7, 890156159572285607, 592943765603823229, 11, 13, 17, 19, 23, 29, 506549493724652201, 31, 37, 467789808681792097]	
11	[2, 3, 5, 673964350278421231, 7, 570852327064636369, 11, 13, 17, 19, 100529311121033219, 782905267445065939, 220494211474505791, 23, 29, 31, 37, 41, 453200337385175791, 1, 1]	
12	[897755467895795269, 712518787676852867, 2, 3, 5, 7, 11, 13, 167457552962765929, 785318264549791391, 519695709762146947, 235664032771761491, 17, 19, 23, 686301435895832203, 29, 31, 135494377082075023, 1, 37, 1, 41]	
13	[2, 276928791662738263, 3, 154706813960393137, 5, 677522714231077789, 7, 847393503652079827, 429007505304621623, 580613740587896963, 830333069581984819, 11, 13, 17, 338200654917516311,	

Test Case #	Input	Expected Output
	19, 23, 791488592359165213, 699476185102200133, 29, 31, 214902385940531323, 1, 37, 1, 41, 43]	
14	[547324030460904911, 2, 242396078936426879, 3, 174160645112100077, 5, 756260982850495393, 7, 11, 13, 285032243509355101, 17, 231769208562727523, 19, 244224781588206871, 23, 29, 340255250467333219, 31, 37, 665453238385590779, 41, 43, 1, 47, 1, 53, 59]	
15	[2, 180754170614523281, 994088034025083043, 3, 5, 500316715813742939, 7, 11, 13, 17, 19, 804886532361322069, 240035844062540969, 23, 29, 163131102228287173, 31, 37, 225746466150511693, 41, 43, 231899900411438251, 47, 748082880400673899, 1, 1]	
16	[310785949449602293, 237921582705770603, 2, 3, 5, 582063826368886079, 7, 275087005057190617, 11, 861900284925848383, 13, 616598029020261121, 17, 145645825520243639, 19, 717107908895557243, 23, 293679483397604107, 250548046658158313, 29, 31, 37, 612502205445612343, 41, 1, 43, 1, 47, 53, 59]	
17	[2, 3, 879272409632603453, 5, 725211450611275153, 7, 135730094775525343, 11, 13, 563794758677534911, 17, 19, 23, 29, 406058158390879763, 686768933549124061, 31, 37, 751333681358862757, 41, 465185147290317199, 43, 47, 282224429542246151, 53, 59, 1, 63, 1, 67, 71, 73]	

Problem 2: SuperAttackers

You have a 50x50 chessboard. Both rows and columns of the chessboard are numbered from 0 to 49, inclusive.

A super attacker is a chess piece that attacks all cells that are in the same row, in the same column, or on the same diagonal. In this problem, we will be using at most sixteen super attackers.

Some super attackers are already placed on the board. You are given their coordinates in the `row` and `col` arrays. More precisely, for each valid `i`, there is a super attacker on the cell `(row[i], col[i])`. These super attackers are placed in such a way that no two of them attack each other.

You want to add additional super attackers onto the chessboard in such a way that in the final configuration no two super attackers will attack each other. Find any one valid solution. Return an array with the coordinates of the added super attackers. More precisely, if you want to place super attackers onto the cells `(r0, c0)`, `(r1, c1)`, and so on, return the array `[r0, c0, r1, c1, ...]`.

Method Signature

Java

```
public int[] addAttackers(int[] row, int[] col, int add) {
    // Your implementation here
}
```

C++

```
vector<int> addAttackers(vector<int> & row, vector<int> & col, int add) {
    // Your implementation here
}
```

Python

```
def addAttackers(self, row: List[int], col: List[int], add: int) -> List[int]
```

Constraints

- `row` will have between 0 and 16 elements, inclusive.
- Each element of `row` will be between 0 and 49, inclusive.
- `col` will have the same number of elements as `row`.
- Each element of `col` will be between 0 and 49, inclusive.

- The super attackers described by `row` and `col` stand on distinct cells and they do not attack each other.
- `add` will be between 0 and 16, inclusive.
- The number of elements in `row` plus the value of `add` will be at most 16.

Examples

Example 1

```
row = [3]
col = [5]
add = 1
```

Example Output: [0, 0]

There is a super attacker at (3, 5). We are asked to add one more super attacker. In the example output shown above, we place it at (0, 0).

Example 2

```
row = [0]
col = [1]
add = 1
```

Example Output: [4, 7]

There is a super attacker at (0, 1). This time we cannot place the second super attacker at (0, 0) because the two super attackers would attack each other.

Test Cases

Your outputs don't have to exactly match expected outputs given here. To quote the problem description, "**Find any one valid solution**".

#	row	col	add	expected output
0	[3]	[5]	1	[0, 0]

#	row	col	add	expected output
1	[0]	[1]	1	[4, 7]
2	[0]	[1]	3	[4, 7, 15, 0, 49, 49]
3	[14, 19]	[3, 47]	0	[]
4	[]	[]	2	[0, 0, 1, 2]
5	[1, 2, 3]	[7, 2, 19]	1	[0, 1]
6	[]	[]	16	
7	[]	[]	0	
8	[4, 35, 36, 16, 24, 39, 1, 34, 32, 28, 2, 23]	[27, 29, 21, 32, 25, 2, 6, 0, 22, 15, 35, 45]	4	
9	[38, 35, 48, 15, 2, 21, 24, 34, 6, 14, 30, 26]	[21, 36, 12, 46, 8, 6, 32, 16, 41, 44, 47, 38]	4	
10	[43, 49, 35, 22, 30, 36, 46, 14, 7, 13, 38, 0]	[18, 15, 16, 34, 19, 49, 33, 38, 40, 24, 35, 10]	4	
11	[4, 32, 34]	[17, 39, 30]	13	
12	[23, 2, 43, 45, 20, 36, 13, 11, 30, 8, 33, 6, 28, 41]	[48, 38, 30, 40, 1, 15, 22, 11, 12, 6, 25, 17, 0, 7]	2	
13	[4, 1, 2, 11, 32, 12, 31, 29, 36, 44, 34, 40, 39, 24]	[23, 16, 8, 47, 31, 49, 33, 36, 0, 4, 21, 2, 15, 7]	2	
14	[5]	[19]	15	