Hello Student Recruitment Team,

Please find below information as requested.

GCSE equivalent

1. Official Name: West African Senior School Certificate Examination
2. School: Federal Government Girls College Sagamu Nigeria
3. Individual Subjects:
4. English Language- C4
5. Mathematics – B3
6. Economics- A1
7. Government- A1
8. Islamic Studies- C4
9. Literature-In-English- A1
10. Yoruba Language- A1
11. Biology- A1
12. Visual Arts- B2
13. Year Obtained: June 2010

A-Level Equivalent

1. Official Name: Diploma II in Law
2. School: University of Lagos
3. Individual Subjects:
4. Introduction to Law of Contract I – 5
5. Use of English I- 4
6. Introduction to Law- 3
7. Introduction to Civil Wrongs- 4
8. Introduction to Constitution- 4
9. Elements of Commercial Law- 5
10. Introduction to Nigerian Courts- 4
11. Introduction to Real Property Rights- 3
12. Introduction to Crime- 5

Total No of Units Taken: 34

CGPA: 4.12

1. Year Obtained: October 2011

Bachelor’s Degree

1. Official Name: Bachelor of Laws
2. School: University of Lagos Nigeria
3. Course: Law
4. Date Attended: 01 November 2011 - 28October 2015
5. Grade: 2:1 (Achieved)

Master’s Degree

1. Official Name: Master of Laws
2. School: University of East Anglia UK
3. Course: International Commercial and Business Law
4. Date Attended: 23 September 2019 – 23 September 2020
5. Grade: Distinction (Achieved)

Master’s Degree

1. Official Name: Master of Science
2. School: Robert Gordon University UK
3. Course: Information Technology
4. Course Duration: 24 January 2022 – 30th June 2023 (current)
5. Grade: Distinction (Expected)

Best regards,

Oluwaseyi

Please send all available details regarding the below qualification(s), or equivalent:

GCSE (Mid High School Exams)  
A level (Senior High School Exams)  
Degree information, including Masters if applicable

If you did not study a particular qualification referenced above, please provide details of the equivalent level qualification studied.

Please respond to this email with the following information:

* The official name of the qualification type studied
* All individual subjects / modules studied and their grades, scores or percentages. Please include the overall grade achieved for the qualification out of a total possible grade if applicable
* Dates of examinations taken, or due to be taken, including the official end date for your course
* Any other relevant information concerning the qualification we should be aware of

You are given a string of digits **panel** and an array of strings **codes**. Each string in the **codes** array consists of digits only and represents a code in the following format: **"<index><pattern>"**, where both **index** and **pattern** should consist of at least one digit. Since there are several ways to split the code, let’s consider them all in ascending order of **index** length and call them split-cases. For instance, for the **code = "1324"**, the split-cases are:

* split-case 1: **index = "1"** and **pattern = "324"**;
* split-case 2: **index = "13"** and **pattern = "24"**;
* split-case 3: **index = "132"** and **pattern = "4"**.

For each code in **codes** and for every split-case of this code, check whether a string **pattern** is present at the **index** position in the **panel** string. Return a string array consisting of results of these checks, where each element is either **pattern**, if this **pattern** is present in **panel**, or otherwise **"not found"**.

Example

* For **panel = "2311453915"** and **codes = ["0211", "639"]**, the output should be **solution(panel, codes) = ["not found", "11", "not found", "39", "not found"]**.

**Explanation:**  
Let's consider all possible codes:

* + For **codes[0] = "0211"** there are three split-cases:
    - **index** is **0** and **pattern** is **"211"**. Since **panel[0..2] = "231" ≠ pattern**, this is not a valid code. The answer for this split-case is **"not found"**.
    - **index** is **02** and **pattern** is **"11"**. **panel[2..3] = "11" = pattern**, so this is a valid code. The answer for this split-case is **"11"**.
    - **index** is **021** and **pattern** is **"1"**. As **panel[21]** doesn't exist, this is not a valid code. The answer for this split-case is **"not found"**.
  + For **codes[1] = "639"** there are two split-cases:
    - **index** is **6** and **pattern** is **"39"**. **panel[6..7] = "39" = pattern**, so this is a valid code. The answer for this split-case is **"39"**.
    - **index** is **63** and **pattern** is **"9"**. As **panel[63]** doesn't exist, this is not a valid code. The answer for this split-case is **"not found"**.  
      The resulting array is **["not found", "11", "not found", "39", "not found"]**.

Input/Output

* **[execution time limit] 3 seconds (java)**
* **[input] string panel**

A string of digits.

*Guaranteed constraints:*  
**1 ≤ panel.length ≤ 103**.

* **[input] array.string codes**

An array of strings of digits.

*Guaranteed constraints:*  
**1 ≤ codes.length ≤ 100**,  
**2 ≤ codes[i].length ≤ 10**.

* **[output] array.string**

An array of strings, where each string is either the existing **pattern** or **"not found"**.

**[Java] Syntax Tips**

**// Prints help message to the console**

**// Returns a string**

**//**

**// Globals declared here will cause a compilation error,**

**// declare variables inside the function instead!**

**String helloWorld(String name) {**

**System.out.println("This prints to the console when you Run Tests");**

**return "Hello, " + name;**

**}**

You need to get to the mall to buy some new shoes, but you're not sure how to get there. Your city is a **n × m** rectangular grid of blocks, where your home is located at the coordinates **(x1, y1)** and the mall's location is **(x2, y2)**. It is guaranteed that **(x1, y1)** and **(x2, y2)** are not the same.

Since you're not sure exactly how to get to the mall, you follow a movement strategy based on these rules:

* Move diagonally on each move, starting in the direction **(+1, +1)**. It means that standing at a cell with coordinates **(x, y)**, you'll move to the cell with coordinates **(x + 1, y + 1)** unless the new cell is outside the city grid,
* If the current move would take you outside the city grid, come back and reverse the direction that was leading outside the grid (eg: if the **x** coordinate is outside the grid, reverse the **x** movement direction),
* If the current move would escape the city grid outside of a corner, reverse both directions.

Your task is to determine how many steps it will take to reach the mall at **(x2, y2)**. Return **-1** if it's not possible to reach the mall using this strategy.

For better understanding how the movement rules work, take a look at the animations in the examples section.

**Notes:**

* A diagonal movement or a bounce (direction change) each count as a single step.
* Coordinates of a cell are defined differently in comparison to mathematical coordinates, take a closer look at the animations in the examples section below.

Example

* For **n = 5**, **m = 5**, **x1 = 2**, **y1 = 1**, **x2 = 1**, and **y2 = 2**, the output should be **solution(n, m, x1, y1, x2, y2) = 7**.

You are given two arrays of integers **a** and **b**, and an array **queries**, the elements of which are queries you are required to process. Every **queries[i]** can have one of the following two forms:

* **[0, i, x]**. In this case, you need to assign **a[i]** the value of **x** (**a[i] = x**).
* **[1, x]**. In this case, you need to find the total number of pairs of indices **i** and **j** such that **a[i] + b[j] = x**.

Perform the given queries in order and return an array containing the results of the queries of the type **[1, x]**.

Example

* For **a = [3, 4]**, **b = [1, 2, 3]**, and **queries = [[1, 5], [0, 0, 1], [1, 5]]**, the output should be **solution(a, b, queries) = [2, 1]**.

The arrays look like this initially:  
**a = [3, 4]** and **b = [1, 2, 3]**

For the query **[1, 5]**, there are two ways to form a sum of **5** using an element from each array: **5 = 3 + 2 = a[0] + b[1]** and **5 = 4 + 1 = a[1] + b[0]**. So the result is **2**.

The query **[0, 0, 1]** re-assigns the value of **a[0]** to **1**, so the arrays now look like this:  
**a = [1, 4]** and **b = [1, 2, 3]**

For the final **[1, 5]** query, there's now only one way to form a sum of **5** using an element from each array: **5 = 4 + 1 = a[1] + b[0]**. So the result is **1**.

Since the two queries of type **[1, x]** gave results of **2** and **1** respectively, the answer is **[2, 1]**.

* For **a = [2, 3]**, **b = [1, 2, 2]**, and **queries = [[1, 4], [0, 0, 3], [1, 5]]**, the output should be **solution(a, b, queries) = [3, 4]**.

The arrays look like this initially:  
**a = [2, 3]** and **b = [1, 2, 2]**

For the query **[1, 4]**, there are three ways to form a sum of **4** using an element from each array: **4 = 2 + 2 = a[0] + b[1]**, **4 = 2 + 2 = a[0] + b[2]**, and **4 = 3 + 1 = a[1] + b[0]**. So the result is **3**.

The query **[0, 0, 3]** re-assigns the value of **a[0]** to **3**, so the arrays now look like this:  
**a = [3, 3]** and **b = [1, 2, 2]**

For the query **[1, 5]**, there are now **4** ways to form a sum of **5** using an element from each array: **5 = 3 + 2 = a[0] + b[1]**, **5 = 3 + 2 = a[0] + b[2]**, **5 = 3 + 2 = a[1] + b[1]**, and **5 = 3 + 2 = a[1] + b[2]**. So the result is **4**.

Since the two queries of type **[1, x]** gave results of **3** and **4** respectively, the answer is **[3, 4]**.

Input/Output

* **[execution time limit] 1 seconds (java)**
* **[input] array.integer a**

An array of integers.

*Guaranteed constraints:*  
**1 ≤ a.length ≤ 5 · 104**,  
**0 ≤ a[i] ≤ 109**.

* **[input] array.integer b**

An array of integers.

*Guaranteed constraints:*  
**1 ≤ b.length ≤ 103**,  
**0 ≤ b[i] ≤ 109**.

* **[input] array.array.integer queries**

An array of queries, where **queries[i][0]** represents the type of query, and the other elements represent the parameters of the query (**i** and **x** for type **0**, and just **x** for type **1**).

For queries of the type **[0, i, x]**, it is guaranteed that **0 ≤ i < a.length** and **0 ≤ x ≤ 109**.  
For queries of the type **[1, x]**, it is guaranteed that **0 ≤ x ≤ 2 · 109**

*Guaranteed constraints:*  
**1 ≤ queries.length ≤ 103**.

* **[output] array.integer**

The output of the queries of the type **[1, x]**, in the order that they are given in the input.