Programming Exercise 4: Greedy Algorithms Instructor: Neil Patrick Del Gallego
Individual

[45 pts] General Instructions: This is a programming exercise for you to test your algorithmic design skills, specifically for implementing **greedy** solutions. You are to devise a **greedy solution** for all problems specified. The most efficient solution for these problems is clearly **a greedy algorithm! Not providing a greedy solution will have no merit.** For convention and consistency, the mini-program are to be done in JAVA. Please use only 1 java class for this exercise.

1. **[15 pts] The Grand Dinner**
Each team participating in this year's ACM World Finals contest is expected to join the grand dinner to be arranged after the prize giving ceremony ends. In order to maximize the interaction among the members of different teams, it is expected that no two members of the same team sit at the same table. Now, given the number of members in each team (including contestants, coaches, reserves, guests etc.) and the seating capacity of each available table, you are to determine whether it is possible for the teams to sit as described in the previous paragraph. If such an arrangement is possible you must also output one possible seating arrangement. If there are multiple possible arrangements, any one is acceptable.

Input: The first line contains two integers **M** ($1 \le M \le 70$) and **N** ($1 \le N \le 50$) denoting the number of teams and the number of tables respectively. The second line contains **M** integers where the **i-th** ($1 \le i \le M$) integer $\mathbf{m_i}$ ($1 \le m_i \le 100$) indicates the number of members of team **i**. The third line contains **N** integers where the **j-th** ($1 \le j \le N$) integer n_j ($2 \le n_j \le 100$) indicates the seating capacity of table **j**. The input is simply represented as an array of strings, using digits as characters. It's up to you how to parse and represent it in your code.

Output: Print a line containing either **1 or 0** depending on whether or not there exists a valid seating arrangement of the team members. In case of a successful arrangement, print **M** additional lines where the **i-th (1 ≤ i ≤ M)** of these lines contains a table number (an integer from 1 to N) for each of the members of team **i**. Table numbers need not be ordered.

Sample Input:	**Sample Output:**
4 5 4 5 3 5 3 5 2 6 4	1 1 2 4 5 1 2 3 4 5 2 4 5 1 2 3 4 5

Coding Instructions
For #1, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void assignTable(String[] input) {
  //properly read the input, parse and represent in your own preference.
  //your print statements here
}
```

2. **[15 pts] The Shoemaker's Problem**
Shoemaker has **N** jobs (orders from customers) which he must make. Shoemaker can work on only one job in each day. For each **i-th** job, it is known the integer $\mathbf{T_i}$ **($1 \le T_i \le 1000$)**, the time in days it takes the shoemaker to finish the job. For each day of delay before starting to work for the **i-th** job, shoemaker must pay a fine of $\mathbf{S_i}$ **($1 \le S_i \le 10000$)** cents. Your task is to help the shoemaker, writing a program to find the sequence of jobs with minimal total fine.

Input: First line of input contains an integer **N** ($1 \le N \le 1000$). The next **N** lines each contain two numbers: the **time** and **fine** of each task in order.

Output: You program should print the sequence of jobs with minimal fine. Each job should be represented by its **index (starts at 1)** in input. All integers should be placed on only one output line and separated by one space.

Sample Input:	**Sample Output:**
4 3 4 1 1000 2 2 5 5 3 1 1 2 2 3 3	2 1 3 4 1 2 3

Coding Instructions

For #2, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void shoemaker(int[] jobTime, int[] fine) {
  //your print statements here
}
```

3. [15 pts] Building Designing

An architect wants to design a very high building. The building will consist of some floors, and each floor has a certain size. The size of a floor must be greater than the size of the floor immediately above it. In addition, the designer (who is a fan of a famous Spanish football team) wants to paint the building in blue and red, each floor a color, and in such a way that the colors of two consecutive floors are different. To design the building the architect has **n** available floors, with their associated sizes and colors. All the available floors are of different sizes. The architect wants to design the highest possible building with these restrictions, using the available floors.

Input: The first line contains the number of available floors. Then, the size and color of each floor appear in each line. Each floor is represented with an integer between -**999999** and **999999**. There is no floor with size 0. Negative numbers represent **red floors** and positive numbers **blue floors**. The size of the floor is the **absolute value of the number**. There are no two floors with the same size. The maximum number of floors for a problem is 500000.

Output: The output will consist of a line with the number of floors of the highest building with the mentioned conditions.

Sample Input:	**Sample Output:**
5 7 -2 6 9 -3	2
8 11 -9 2 5 18 17 -15 4	5

Coding Instructions

For #3, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void design(int[] floors) {
  //your print statements here
}
```

Submission Instructions

- Use only 1 Java class for this exercise. You are free to use as many functions, inner classes or structs to organize your code.
- Follow the coding instructions! Not following the coding instructions will merit deductions.
- Test your code thoroughly and try out different test cases, especially the edge cases (0 or N values). Your instructor will run with at least 3 test cases to check your code.
- Optimize your performance. If your code executes for more than 5 seconds, it will automatically fail the test case! Your code will be tested on an i7 processor with 8GB of RAM.
- Submit this exercise on or before the indicated deadline in Canvas.