Problem Set

ICPC Online Preliminary Programming Competition 2020-2021 Asia – Topi (Region)

Instructions

- Do not open the booklet unless you are explicitly told to do so. You can only read these instructions below.
- Do not create disturbance or move around unnecessarily in the arena.
- If you have any question regarding the problems, send a clarification from the judges using PC^2.
- There would be no internet access and mobile phones are also not allowed.
- Before submitting a run, make sure that it is executable via command line. For Java, it must be executable via "javac" and for GNU C++ via "g++". In particular, Java programmers need to remove any "package" statements and source code's file name must be the same as of main class. C++ programmers need to remove any getch() / system("pause") like statements.
- Do not attach input files while submitting a run, only submit/attach source code files, i.e., *.java or *.cpp or *.py.
- Language supported: C/C++, Java, and Python 2 & 3
- Source code file name should not contain white space or special characters.
- You will have to make an Input file for every problem based on the format provided in the Sample Input. For example, the input file for Problem 1 shall be “P1_input.txt”; for problem 2, it shall be “P2_input.txt”; and so on. The outputs of all the problems would have to be on the console according to the sample output given for each problem.
- While programming, do not open input files by providing absolute/complete paths (absolute paths of your machine and judge's machine would be different). Just open them from the current directory from where your code is running.
- Please, do not create/open any file for output. All outputs must be on console.
- Please strictly meet the output format requirements as described in problem statements, because may be some auto judge will be evaluating your program which will compare your output with judge's output byte-by-byte and not tolerate even a difference of single byte. So, be aware! **Pay special attention to spaces, commas, dots, newlines, decimal places, case sensitivity, etc.**
- Unless mentioned in some problem, all your programs must meet the time constraint of 5 seconds.
- The decision of judges will be final.

PROBLEM 1: Strange Password

Time limit: 10 seconds

One day Ali Baba was wandering around in a forest when he heard some horses. He followed the sounds and from behind a tree, he saw 40 horsemen standing in front of a cave. He heard one of them repeating some numbers and the door of the cave opened, and all the horsemen went into the cave. Fearing for his life, he came near the door and found out that it was no ordinary door. It was a 21st century door equipped with digital display and voice recognition system. The display showed the number 15, but he could not understand what it meant. He returned home hoping to come again. The next day when the horsemen were entering the cave, he saw the number 10 on the display. He went near and the number changed to 7. He went a few steps back and came near the door again and saw that the number had changed again. So, he figured out that it was a random number. Now he was curious what the leader of the horsemen would say to open the door. In the next few days, he observed the scenario and figured out that the password consisted of a sequence of numbers that change with the number shown on the display.

One day Ali Baba felt himself lucky as when he approached the door, it displayed 1. He spoke 1 and the door opened. In the next few days, he figured out that when the display was 2, the door could be opened by saying 11. He tried for 3 and found out that saying 21 was the password to open the door.

Ali Baba realized that the password consisted of visually grouping the consecutive runs of the same digit. Thus, starting with the number 1, the second number is generated by saying one 1 which produces 11. The third number is coming from 11 which becomes two 1s, i.e., 21. The fourth number is coming from 21 and it becomes one 2 and then one 1, resulting in 1211. The fifth number is generated from 1211 and comes as one 1, one 2, and two 1s, i.e., 111221. He also noticed that the displayed number was never less than 1 or more than 50.

Your job is to write a program for Ali Baba to generate the desired numbers in the sequence.

Input

The first line of the input consists of *n*, $(0 < n <= 5)$, representing number of cases. Each of the following lines consists of *x*, meaning that the password is the x^{th} number in the sequence $(1 <= x <= 50)$.

Output

Each line consists of the test case given by "Case #i: ", where *i* is the test case number. This is followed by the password corresponding to the x^{th} number in the sequence.

Sample input	Sample Output
5 1 4 5 7 9	Case #1: 1 Case #2: 1211 Case #3: 111221 Case #4: 13112221 Case #5: 31131211131221

PROBLEM 2: Fair Game

Time limit: <5 seconds

Selection of players for the two competing sides before playing a game of cricket in our neighborhood is a clever task for captains. Usually, two bullies from the street are self-appointed captains of the opposing teams, and they would take turns to pick their players. On each turn, they would choose the most capable player. Or, towards the end they would choose the least inept player from the pool of remaining candidates until all those wannabe cricketers are assigned to one side or the other. The aim of this process was to come up with two evenly matched teams, and to remind each of us of our precise ranking in the neighborhood pecking order.

We all assumed this was a fair process. But did it produce two evenly matched teams? We believed so, but then there were indications, while play was in progress, that made us think that the other team was stronger than ours and may be an exchange of a couple of players between the teams would make them balanced. We want to eliminate that unfair balancing between the teams, so there is a scope of improvement in the team selection process.

We want to find a process that will result in the creation of two evenly balanced teams considering the strength of each player. Of course, the two teams may not be perfectly evenly balanced but the difference between their strength should be minimal. So, a pool of players is available for selection by the two teams. Each player must belong to one of the two teams. The number of players on the two teams must not differ by more than 1. Each player has a skill-point associated with him indicating their strength. The total skill-points of the players on both teams should be as nearly equal as possible. (The absolute difference of the sum of skill-points of players in each team should be the least).

Input

The first line of input will contain the number of test cases 'T'(1<=T<=100). This is followed by 'T' test cases. Each subsequent test case starts with N, the total number of players. The next N numbers on the same line show the skill-point of each person. Each skill-point shall be an integer between 1 and 450. There shall be at most 100 players in all (1<=N<=100).

Output

Your output should be exactly T lines, each corresponding to the output of a test case. Each line consists of the test case given by "Case #i: ", where *i* is the test case number, followed by the sum of the skill-points of the players on one team, and the sum of skill-points of the players on the other team. Print the smaller sum first.

Sample input	**Sample Output**

4
3 90 200 100
10 2 3 10 5 8 9 7 3 5 2
10 1 1 1 1 1 1 1 1 1 9
8 87 100 28 67 68 41 67 1
	Case #1: 190 200
	Case #2: 27 27
	Case #3: 5 13
	Case #4: 229 230

PROBLEM 3: Prime Factors

Time limit: 10 seconds

There is a method in arithmetic that, for every positive non-prime integer *n*, there is a unique way to represent *n* as a product of two or more primes. Also, several arrangements for prime factors may exist. For Example,

$$10 = 2 \times 5 \qquad 20 = 2 \times 2 \times 5$$
$$= 5 \times 2 \qquad = 2 \times 5 \times 2$$
$$= 5 \times 2 \times 2$$

Let there be a function $F(n)$ that returns the number of all possible sequences of the prime factors of *n*. So, $F(10) = 2$ and $F(20) = 3$.

For a given positive number *j*, there always exists at least one number *n* such that $F(n) = j$. We want to know the smallest such *n*.

Input

The first line of the input contains the number of test cases (at most 1 000 test cases). Each subsequent line contains the number *j* (with $j < 2^{63}$), for which the smallest *n* is desired, such that $F(n)=j$. Each test case is a positive integer.

Output

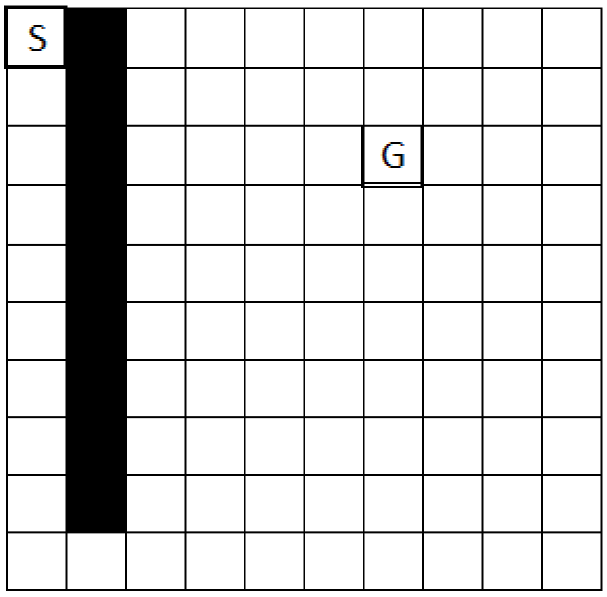
Each line consists of the test case given by “Case #i: ”, where *i* is the test case number, followed by the numbers *j* and *n*.

Sample Input 1	**Sample Output 2**
4 1 2 3 105	Case #1: 1 2 Case #2: 2 6 Case #3: 3 12 Case #4: 105 720

PROBLEM 4: Pawn's Path

Time limit: 20 seconds

You need to move a pawn placed on a 10x10 cell board in minimum number of steps without hitting any blocked cell. The pawn can take any of the four steps, i.e., UP, DOWN, RIGHT and LEFT. Write a program to find the minimum number of steps required to reach the given goal cell position given the pawn's starting position.



Input

The first line is the number of test cases. Each test case is given as follows: the first line of the test case contains the starting cell position (x_s,y_s). The second line contains goal cell position (x_g,y_g). The following 10 lines contains cell board information ordered row wise. A cell value of 1 means it is a blocked cell, while 0 means a clear cell.

```
xs,ys
xg,yg
c00,c01,...,c09
c10,c11,...,c19
.
.
.
c90,c91,...,c99
```

Output

Each line consists of the test case given by "Case #i: ", where *i* is the test case number, followed by the number *m*, where *m* is the minimum number of steps required to reach from starting position to goal position, if there exists a path. If there is no path from starting position to the goal position, output is -1 as shown for the third sample input.

Sample input	Sample Output
3 0,0 6,2 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0 6,2 2,2 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,1,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0 6,2 2,2 0,1,0,0,1,1,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,1,0,0,1,0,0,0,0,0 0,0,0,0,1,0,0,0,0,0	Case #1: 22 Case #2: 4 Case #3: -1

PROBLEM 5: Denominations

Time limit: 5 seconds

Faisal works as a cashier/teller at a local bank located in a not-so-well-off neighborhood. Customers come along all day long, making a good turn over. Every once in a while, an old lady of the neighborhood would cash a cheque. She requests him to give her the minimum number of currency notes, down to 10-rupee bills. For the rest, he gives her toffees that the old lady would give to her children.

Input

The first line of the input contains n, which is the number of test cases ($1 \leq n \leq 1000$). Each subsequent line contains the amount to be given to the old lady (in the range $10 - 1{,}000{,}000$).

Output

Each line consists of the test case given by "Case #i: ", where *i* is the test case number. For each test case, output the minimum count of each denomination starting from the largest non-zero denomination. It should then list all the remaining denominations, even if zero, as shown in the sample output. The possible denominations are 5000, 1000, 500, 200, 100, 50, 20, and 10, respectively.

Sample input	Sample Output
2 4270 320	Case #1: 4x1000, 0x500, 1x200, 0x100, 1x50, 1x20, 0x10 Case #2: 1x200, 1x100, 0x50, 1x20, 0x10

PROBLEM 6: Travelling Salesman Bonus

Time limit: 10 seconds

The travelling salesman problem (TSP) is a well-known problem in which a salesperson must travel between all cities, but not repeating a city, in the least amount of time/distance. Your company sells computers to many cities and wishes to employ salesmen that will visit the cities to try and sell computers. Instead of trying to solve the TSP, however, they decided to hire multiple salesmen and let each salesman select a route of their choice which they will visit to sell the company's product. A salesman is paid a bonus by the number of cities they travel in a trip, not counting the home city. You may wish to start at any city which will be your home city. Therefore, it is advantageous to select a route covering the greatest number of cities.

As with the TSP, you are not allowed to repeat a city. The distance between any two cities is considered the same and can be travelled in either direction. The cities are connected in such a way that no two cities may be connected via multiple paths. The salesman is paid a constant amount per city visited. Therefore, the higher the number of cities visited, the higher would be the payment. As a senior salesman, you have been given the opportunity to select your preferred route. It is your job to select a route that maximizes your reward. In case there are multiple solutions with the same bonus, you may select any one of your choice.

Input:

The first line in the input file is the number of test cases. Each subsequent case is given in a new line and contains the following: The first number is the number of total cities, followed by links in the form of *x y* indicating that there is a link between city *x* and city *y*. The values are comma separated as shown.

Output:

Each line consists of the test case given by "Case #i: ", where *i* is the test case number. This is followed by the selected pair of starting and ending city along with the expected bonus from the visit as shown.

Sample input	**Sample Output**
2 5, 0 1, 1 2, 2 4, 4 3 6, 1 2, 2 4, 0 4, 3 4, 3 5	Case #1: Pair (0,3), Bonus = 4 Case #2: Pair (1,5), Bonus = 4

Note: City numbers start from 0.