**Sapientia**
**Hungarian University of Transylvania**

# Sapientia ECN International Programming Contest – 2014

# PROBLEMS

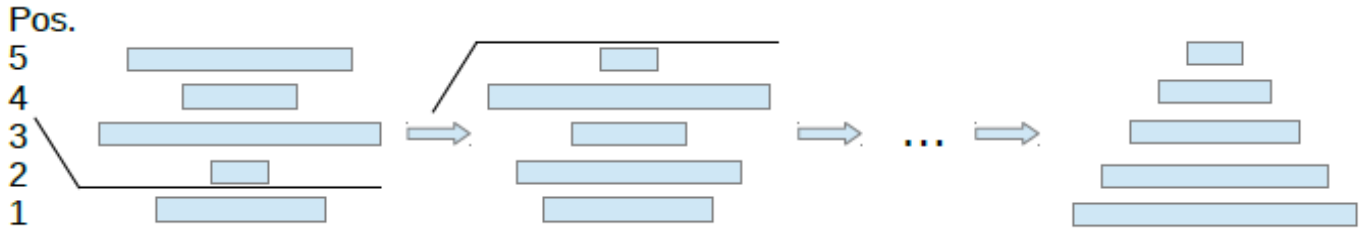

**Târgu Mureş / Marosvásárhely, 10th May 2014**

# PROBLEM A

## Pancake sorting

| Input File: | A.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

You are given a plate of *n* pancakes which have distinct diameters from 1 to *n* and are placed one on another in an arbitrary order. You are also given a spatula which can be inserted at any point and used to flip all pancakes above it. Sort the stack of pancakes using only this reversing action as many times as needed. The pancake of the maximum diameter must be on the plate and the one with the minimum diameter on the top of the stack (see picture below).




### Input

The input file stores several test cases, the first line contains the number of tests. For each test case the next lines contain the number of pancakes *n* followed by their initial order starting from the plate, all of these are separated by space characters.

### Output

For each test case you have to print the positions the spatula has to be inserted below. Positions are numbered starting with 1 from the plate (in the example from the picture, we would have to print 2). Mark that the sorting is done by adding a non-existing position 0 at the end. Separate positions by space characters and the test cases by new lines.

### Restrictions and refinements

- The input file contains at most 10 test cases, and $2 \leq n \leq 100$.
- There is more than one correct solution, but you shouldn't exceed $n^2$ flips for a single plate.

| Input: A.in |
|---|
| 2<br>5 3 1 5 2 4<br>3 2 1 3 |

| Output: Standard output |
|---|
| 2 3 4 1 2 3 2 3 0<br>1 2 0 |

# PROBLEM B

## Supermarket

| Input File: | B.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Specialists in supermarkets work hard to optimize their company's profit, in one hand maximizing their income, in other hand minimizing their costs. Profit maximization is achieved with different methods, including efficient marketing, proper price politics and optimal product arrangement. Specialists realized that with negligible extra cost, only by putting the proper products to the proper places the sales can be nicely increased. For example the candies and toys placed on the lower shelves are much easier observed by children, who in one way or another usually achieve them to be landing in their parents' basket. In other hand, there are products like salt and sugar, for example, which are bought basically in the same quantities independently by their position; strategically important places aren't worth "sacrificing" for them.

Given a number of $n$ products, $n$ positions and the expected daily profit resulting for each product in every position, calculate the optimal arrangement of the products.

### Input

The first line of the input file contains the number $t$ of test cases. The first line of each test case represents the number $n$ of products and positions. The next $n$ lines contains the $p[i][j]$ values, the expected daily profit for every product in every position, where the rows represent the products, the columns the positions.

### Output

For each test case your program has to print on a separate line the optimal order of the products. The numbering of products starts at 1.

### Restrictions and refinements

- $1 \leq t, n \leq 10$
- $0 \leq p[i][j] \leq 10\,000$
- every input and output value will be an integer

**Sample Input: B.IN**

```
2
4
210 220 210 180
325 280 190 175
160 410 360 280
270 340 390 245
3
95 60 55
70 80 85
55 50 50
```

**Sample Output: Standard output**

```
2 3 4 1
1 3 2
```

## Explanation

In the following tables the chosen products for every position are marked with grey:

| | Position 1 | Position 2 | Position 3 | Position 4 |
|---|---|---|---|---|
| Product 1 | 210 | 220 | 210 | 180 |
| Product 2 | 325 | 280 | 190 | 175 |
| Product 3 | 160 | 410 | 360 | 280 |
| Product 4 | 270 | 340 | 390 | 245 |

Total estimated profit: 325+410+390+180 = 1305

| | Position 1 | Position 2 | Position 3 |
|---|---|---|---|
| Product 1 | 95 | 60 | 55 |
| Product 2 | 70 | 80 | 85 |
| Product 3 | 55 | 50 | 50 |

Total estimated profit: 95+50+85 = 230

# PROBLEM C

## integer-to-english

| Input File: | C.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Write a program that takes an integer and displays the English name of that value.

You should support both positive and negative numbers, in the range supported by a 32-bit integer (approximately -2 billion to 2 billion).

### Remark

1 = one, 2 = two, 3 = three, 4 = four, 5 = five, 6 = six, 7 = seven, 8 = eight, 9 = nine, 10 = ten,
11 = eleven, 12 = twelve, 13 = thirteen, 14 = fourteen, 15 = fifteen, 16 = sixteen, 17 = seventeen,
18 = eighteen, 19 = nineteen, 20 = twenty, 30 = thirty, 40 = forty, 50 = fifty, 60 = sixty, 70 = seventy,
80 = eighty, 90 = ninety, 100 = hundred, 1000 = thousand, 1000000 = million, 1000000000= billion;

### Input

The first line of the input file contains the number *t* of test cases, and then the input text file contains lines, each line having a number.

### Output

For each number your program has to print on a separate line the English name of that value.

| Sample Input: C.IN |
|---|
| 5<br>110<br>121<br>1032<br>11043<br>1200000 |

| Sample Output: Standard output |
|---|
| one hundred ten<br>one hundred twenty one<br>one thousand thirty two<br>eleven thousand forty three<br>one million two hundred thousand |

# PROBLEM D

## N-Queens Puzzle

| | |
|---|---|
| **Input File:** | `D.IN` |
| **Output File:** | `STANDARD OUTPUT` |

Place $n$ queens on a $n \times n$ chessboard so that no two queens attack each other.

### Input

The input file consists of up to ten different values of $n$ separated by spaces.

### Output

For each $n$ write a solution to the standard output using the following encoding:

$c1\ c2\ c3\ ...\ cn$ <newline character>.

This means that the first queen is in column $c1$, the second queen is in column $c2$, etc. Columns are numbered from 0.

### Restrictions and refinements

- You can assume that $n \geq 5$ and $n \leq 10\ 000$.
- Time limit: 1 minute.

| Sample Input: D.IN |
|---|
| 5 8 |

| Sample Output: Standard output |
|---|
| 1 4 2 0 3<br>5 2 0 7 3 1 6 4 |

# PROBLEM E

## Primes: shoulder-to-shoulder

| Input File: | E.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Given a natural number ($0 < n < 10^{100}$) your task is to find its optimal partitioning in concatenated prim numbers ($n = p_1p_2 \ldots p_m$) such that:

- $1 < p_i < 2^{32}$, $1 \le i \le m$;
- $m$ is minimal.

(Each prime is obtained using consecutive digits of the given natural number; see the sample input/output)

### Input

The first line stores the number of the test-cases. Line $i$+1 stores two values: the natural number ($n$) and a binary digit (0/1) separated by a single space (0: only the minimal $m$ must be printed; 1: an optimal partitioning must be printed too).

### Output

For each test case print to a new line one of the following

- NO (there is no solution)
- The minimal $m$ (if the solution exists and the corresponding digit is 0)
- The minimal $m$ and the optimal partitioning (there is a solution and the corresponding digit is 1). The value of $m$ is followed by character ':' and one space. The $p_i$ values are separated by character ',' and one space.

**Sample Input: E.IN**

```
4
7919 1
5555555 1
444 1
1383 0
```

**Sample Output: Standard output**

```
1: 7919
7: 5, 5, 5, 5, 5, 5, 5
NO
2
```

# PROBLEM F

## Train timetable update

| Input File: | F.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Andrew developed his own website for train timetable and he needs to update it periodically. For this purpose, he must collect the whole data from the official website. The initial data set contains an old list of stations with its own coding. After collecting the new train timetable, he must review this data set, because some stations are no more used and other new stations must be inserted.

### Input

The initial data set is described as follows: the initial number of stations followed by the list of stations: the internal code (31 bits integer) and the station name, each station on a single line. The collected stations of the new train timetable follow after the initial data set, each name on a single line.

The name of each station is at most 35 characters long. Beware: empty lines may occur freely in the collected list.

### Output

Andrew wants to display the stations that are no more used (internal code and name) sorted by internal code, and the new stations in alphabetic order.

You have to display the results following this rule. Excepting the explanatory texts – *Stations not used* and *New stations*, the information are displayed after one space character on every line.

### Restrictions and refinements

- The initial data set contains at most 150 000 records, and you have to manage at most 1500 distinct new stations.
- Response time: two seconds.

**Sample Input: F.IN**

```
5
1 Paris Nord
2 Madrid Sur
3 Berlin Westen
4 Roma Est
5 Wien

Paris Nord
Roma Est
Budapest Keleti
Bucuresti Nord

Paris Nord
Bucuresti Nord
Madrid Sur
```

**Sample Output: Standard output**

```
Stations not used:
 3 Berlin Westen
 5 Wien
New stations:
 Bucuresti Nord
 Budapest Keleti
```

# PROBLEM G

## Silly Sort[1]

| Input File: | G.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Your younger brother has an assignment and needs some help. His teacher gave him a sequence of numbers to be sorted in ascending order. During the sorting process, the places of two numbers can be interchanged. Each interchange has a cost, which is the sum of the two numbers involved.

You must write a program that determines the minimal cost to sort the sequence of numbers.

### Input

The input file contains several test cases. Each test case consists of two lines. The first line contains a single integer $n$ ($n$ >1), representing the number of items to be sorted. The second line contains $n$ different integers (each positive and less than 1000), which are the numbers to be sorted.

The input is terminated by a zero on a line by itself.

### Output

For each test case, the output is a single line containing the test case number and the minimal cost of sorting the numbers in the test case.

Place a blank line after the output of each test case.

**Sample Input: G.IN**

```
3
3 2 1
4
8 1 2 4
5
1 8 9 7 6
6
8 4 5 3 2 7
0
```

**Sample Output: Standard output**

```
Case 1: 4

Case 2: 17

Case 3: 41

Case 4: 34
```

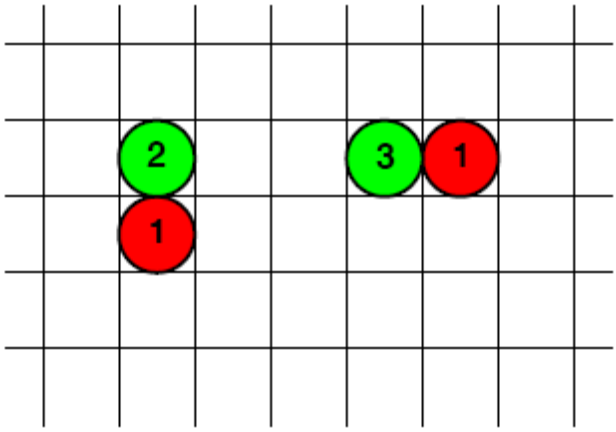
[1] This problem was one of the problems of the 2002 ACM International Collegiate Programming Contest World Finals in Honolulu, USA, http://icpc.baylor.edu/download/worldfinals/problems/2002WorldFinalProblemSet.pdf, http://uva.onlinejudge.org/external/10/1016.html, https://icpcarchive.ecs.baylor.edu/external/24/2481.html.

# PROBLEM H

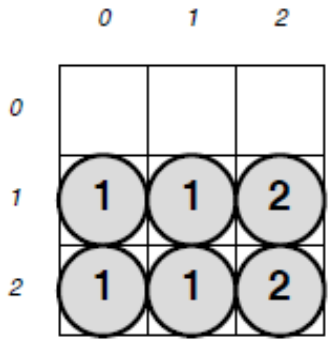
## Quantum Coins Game

| Input File: | H.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Let us suppose we have a board of square-shaped cells, similar to a chessboard (without coloring). In each cell we have a number of "quantum" coins (0, 1, 2, ...). The quantum property simply means that the coins have a fission and fusion property: each coin sitting on a cell can be transformed by its annihilation into two new coins, appearing on the cell just above it, or it can be transformed into three new coins appearing on the neighboring cell to its left (if such neighboring cells there exists on the board at all). This is what we call a fission.




The process is reversible: the reverse transformation is the fusion. On the figure above when 1 red coin disappears, then simultaneously 2 respectively 3 green coins appear, depending on the direction **U**p respectively **L**eft of the fission. Conversely, according to the fusion's direction **D**own or **R**ight, when 2 or 3 green coins disappear from a cell, then 1 red coin will born in their down or right near neighbor respectively.



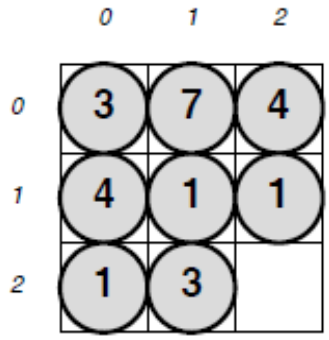

At the absolute 0 temperature everything is frozen, the quantum coins too, consequently they exhibit a regular "crystal" structure. This means that

*(a)* each cell contains either 0 (none), 1 or 2 coins, and

*(b)* the non empty rows are identical.

So, for example on a tiny square board of just 3×3 cells, a frozen state can looks like in the figure above.

Increasing the temperature, random fissions and fusions duration shown in the next figure. (The numbers denote as before the amount of coins sitting in that position.)




Notice the following interesting fact. Suppose we do know the frozen configuration, and suppose we miss the steps of the transformation. Looking at the final configuration into which this evolved, it is quite easy to reconstruct a possible scenario of the fission and fusion steps, which could have been taken place. Such a sequence for the configurations above could be

$$U^2_{2,2},\ L^1_{2,1},\ U^2_{1,2},\ U^3_{2,0},\ R^1_{1,0},\ L^3_{1,2},\ U^4_{1,1},\ L^1_{0,1},\ D^3_{1,1},$$

where the denotations are obvious: $U^2_{2,2}$ means fission upward twice at the cell (2, 2), ..., and $D^3_{1,1}$ means fusion downward three times at the cell (1, 1). It is clear however, that this scenario is not the only one possible. The question now is: what if we do NOT know the frozen configuration, and our quantum refrigerator is broken? That is, if we dispose of just the high temperature configuration, like in the figure, the question is, whether can or can not be *computed* a frozen configuration, from which this state possibly came out? Even more ambitiously, suppose we dispose of a hypothetical high temperature configuration. Can one compute all the frozen states, from which this can be attained if such a state there exists at all? In other words, the task is to compute all the crystal phases, into which a given configuration can be frozen within the margins of the given board, if they exist, or certify the non-existence of such a phase!

Help the physicists and show them how this task can be completed: discover the hidden laws of the frozen quantum coins world! Write a program which performs such a computation and reveal the crystal structures of quantum coins.

## Input

Contains several blocks, each one with the structure:

- the first line, a comma separated pair of positive integers, the size of the bord, say *n, m*
- *n* lines of a number of *m* comma separated non-negativ integers, representing the coin-configuration on the board

## Output

It should contain – for each input block – succesively:

- a single line containing the number 0, if solution doesn't exist for that input case, or
- a line containing a positive integer $k > 0$, representing the number of solutions found, followed by *k* blocks, each structured identically to the input block of which solutions they represent.

| Sample Input: H.IN |
|---|
| 3,3<br>3,7,4<br>4,1,1<br>1,3,0 |

| Sample Output: Standard output |
|---|
| 1<br>3,3<br>0,0,0<br>1,1,2<br>1,1,2 |

# PROBLEM I

## Minimal difference

| Input File: | I.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

Given 2*N* digits, your task is to create two *N*-digit numbers (leading zeros allowed) with minimal difference.

### Input

The first line of the input contains *n*, the number of test cases ($0 < n < 1000$).

This is followed by *n* lines, each containing ten integers $a_0, a_1, ..., a_9$ ($0 \le a_i < 1000$).

Each line describes one test case by listing the number of available digits: $a_i$ is the amount available from *i*. The total number of digits is always even.

### Output

Your solution should contain one line for each test case, with a single integer: the minimum difference of two *N*-digit numbers built from the given digits.

| Sample Input: I.IN |
|---|
| 5<br>0 0 2 2 0 0 4 0 0 0<br>1 1 1 1 1 1 1 1 1 1<br>3 0 0 1 0 0 1 0 0 1<br>3 3 3 3 3 3 0 0 0 2<br>2 1 1 0 1 1 1 1 1 1 |

| Sample Output: Standard output |
|---|
| 0<br>247<br>21<br>17<br>136 |

### Explanation

2366 – 2366 = 0

50123 – 49876 = 247

060 – 039 = 21

5432104012 – 5432103995 = 17

50012 – 49876 = 136

# PROBLEM J

## Sidewalk

| Input File: | J.IN |
|---|---|
| Output File: | STANDARD OUTPUT |

A sidewalk of length $N$ and width $L$ has to be paved with tiles. Tiles are of different types. From each type an infinite amount is available. The length of each tile is $L$. The width of tiles can take any of the following values: $a_1$, $a_2$, ..., $a_k$. The sidewalk has $M$ number of occupied zones which will not be paved. These zones are square-shaped of length 1 with known coordinates: $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_m, y_m)$. The following figures show three different ways of paving a sidewalk of size 6×3 having three occupied zones at positions (6, 2), (3, 1) and (6, 3), using two types of tiles: 1×3 and 2×3, respectively.

### The problem

Given the size of the sidewalk, the types of the available tiles and the coordinates of the occupied zones, calculate the number of different ways of pavings modulo 666 013.

### Input

The first line of the input file is the number of tests, and then the first line of each test contains four natural numbers, $N$, $L$, $K$ and $M$ separated by spaces. These represent the length and width of the sidewalk; the number of types of tiles, and the number of occupied zones, respectively. The next line contains the widths of tiles separated by spaces, i.e. $a_1$, $a_2$, ..., $a_k$. The next $M$ lines contains the coordinates of the occupied zones. Each line consists of two natural numbers separated by a space.

### Output

The standard output will contain the number of different ways of pavings modulo 666 013.

### Restrictions and refinements

- $0 < N \leq 100000$
- $0 < K \leq L \leq 255$
- $0 < a_1, a_2, ..., a_k \leq L$ are pairwise distinct
- $0 \leq M \leq 450$
- $M = 0$ for 20% of the tests
- At least one solution is guaranteed.
- The usage of 64 bit integers is recommended when doing multiplications.
- Maximum running time allowed per test is 2s.
- Total memory available is 32MB.

| Sample Input: J.IN |
|---|
| 1<br>6 3 2 3<br>2 1<br>6 2<br>3 1<br>6 3 |

| Sample Output: Standard output |
|---|
| 4 |

### Explanation

There are 4 distinct ways of paving the sidewalk. Three of them are shown in the figure above.