1 Welcome to Computer Programming

Before you start working on your assignment, here is a kind reminder. Please adhere to the following rules to avoid any potential penalties:

1. Minimize the Use of Generative AI (not mandatory, but its use may hinder your learning experience)
2. Avoid Plagiarism: While discussing ideas with classmates is encouraged, copying their code is not allowed.
3. Try Your Best but Not Too Much: You don't need to achieve a perfect score. Challenge yourself but understand that perfection isn't necessary.

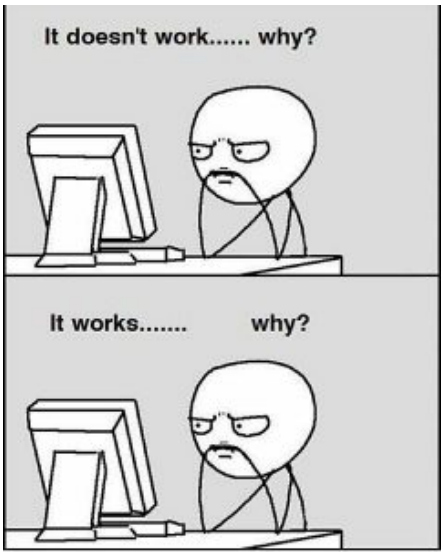
Please note that Problem 6 submitted will be checked for plagiarism. If your work is suspected of being copied, you may get zero point for this problem.

2 Grading

Each problem only if your submission achieves AC (Accepted), you can receive the full score, otherwise you will get 0 points. Your score will be compared against the following score chart:

Number of correct answers	0	1	2	3	4	5	6
Your assignment score	0	30	50	65	80	90	100

Table 1: Score Table



3 Problems

Problem 1. *Score*

There is an objective test result such as "OOXXOXXOOO". An 'O' means a correct answer of a problem and an 'X' means a wrong answer. The score of each problem of this test is calculated by itself and its just previous consecutive 'O's only when the answer is correct. For example, the score of the 10th problem is 3 that is obtained by itself and its two previous consecutive 'O's. Therefore, the score of "OOXXOXXOOO" is 10 which is calculated by "1+2+0+0+1+0+0+1+2+3". You are to write a program calculating the scores of test results.

Input Description

The input begins with an integer T ($1 \leq T \leq 1000$), representing the number of test cases. Each test case consists of:

- A single line containing a string of length L ($0 \leq L \leq 80$)
- The string contains only 'O' and 'X' characters
- No spaces between characters

Output Description

For each test case, output a single integer representing the total score according to the scoring rules.

Sample Input	Sample Output
5	10
OOXXOXXOOO	9
OOXXOOXXOO	7
OXOXOXOXOXOXOX	55
OOOOOOOOOO	30
OOOOXOOOOXOOOOX	

Table 2: Sample I/O

Problem 2. *Sort Array*

Given an array of integers nums, sort the array in ascending order and return it. I highly recommend implementing quick sort or merge sort, otherwise, you might get a TLE (Time Limit Exceeded).

Input Description

The input consists of an array of integers nums where:

- $1 \leq nums.length <= 5 * 10^6$
- $-5 \times 10^7 \leq nums[i] \leq 5 * 10^7$

Output Description

Print sorted array.

Sample Input	Sample Output
4 5 2 3 1	1 2 3 5
6 5 1 1 2 0 0	0 0 1 1 2 5

Table 3: Sample I/O

Example 1:

- Input: nums = [5,2,3,1]
- Output: [1,2,3,5]
- Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:

- Input: nums = [5,1,1,2,0,0]
- Output: [0,0,1,1,2,5]
- Explanation: Note that the values of nums are not necessarily unique.

Problem 3. *Merge Two Sorted Lists Using Pointers*

Given two sorted linked lists, write a function that merges them into a single sorted linked list using only pointer manipulation. The merged list should be created by manipulating the pointers of the original lists without creating new nodes. I highly recommend implementing it with pointers, otherwise, you might get an MLE (Memory Limit Exceeded).

Input Description

The input consists of two lines:

- First line contains n_1 ($1 \leq n_1 \leq 10^6$) followed by n_1 integers representing the first sorted linked list
- Second line contains n_2 ($1 \leq n_2 \leq 10^6$) followed by n_2 integers representing the second sorted linked list

All integers in the lists are in the range $[-10^5, 10^5]$ and are given in non-decreasing order.

Output Description

Output a single line containing $(n_1 + n_2)$ integers representing the merged sorted list.

Sample Input	Sample Output
4 1 3 5 7 4 2 4 6 8	1 2 3 4 5 6 7 8
3 1 2 3 3 1 2 3	1 1 2 2 3 3
2 -1 5 2 0 7	-1 0 5 7

Table 4: Sample I/O

Problem 4. *Robot Motion Planning*

The world of robotics intersects various Computer Science disciplines, including artificial intelligence, algorithms, and engineering. This problem focuses on robot navigation in a bounded rectangular grid world.

Your task is to track a robot's position as it explores a pre-Columbian flat world. The robot moves according to specific commands while avoiding locations where previous robots have been lost.

A robot position consists of a grid coordinate (a pair of integers: x-coordinate followed by y- coordinate) and an orientation (N,S,E,W for north, south, east, and west). A robot instruction is a string of the letters 'L', 'R', and 'F' which represent, respectively, the instructions:

- Left: the robot turns left 90 degrees and remains on the current grid point.
- Right: the robot turns right 90 degrees and remains on the current grid point.
- Forward: the robot moves forward one grid point in the direction of the current orientation and mantains the same orientation.

The direction North corresponds to the direction from grid point (x, y) to grid point (x, y + 1). Since the grid is rectangular and bounded, a robot that moves "off" an edge of the grid is lost forever. However, lost robots leave a robot "scent" that prohibits future robots from dropping off the world at the same grid point. The scent is left at the last grid position the robot occupied before disappearing over the edge. An instruction to move "off" the world from a grid point from which a robot has been previously lost is simply ignored by the current robot.

Figure 1: Cry! Stanley

Input Description

The first line of input is the upper-right coordinates of the rectangular world, the lower-left coordinates are assumed to be 0,0. The remaining input consists of a sequence of robot positions and instructions (two lines per robot). A position consists of two integers specifying the initial coordinates of the robot and an orientation (N,S,E,W), all separated by white space on one line. A robot instruction is a string of the letters 'L', 'R', and 'F' on one line. Each robot is processed sequentially, i.e., finishes executing the robot instructions before the next robot begins execution.

You may assume that all initial robot positions are within the bounds of the specified grid. The maximum value for any coordinate is 50. All instruction strings will be less than 100 characters in length.

Output Description

For each robot position/instruction in the input, the output should indicate the final grid position and orientation of the robot. If a robot falls off the edge of the grid the word 'LOST' should be printed after the position and orientation.

Sample Input	**Sample Output**
5 3 1 1 E RFRFRFRF 3 2 N FRRFLLFFRRFLL 0 3 W LLFFFLFLFL	1 1 E 3 3 N LOST 2 3 S

Table 5: Sample I/O

Problem 5. *Tower of Hanoi Recursive Solution*

Given three pegs (labeled as 1, 2, and 3) and n disks of different sizes initially stacked in ascending order on peg 1, write a program to print the sequence of moves to transfer all disks from peg 1 to peg 3.

Some Rules :

- Only one disk can be moved at a time
- A disk can only be placed on top of a larger disk or an empty peg
- Each move should be printed in the format: Move disk "disk number" from peg "source" to peg "destination"

Input Description

A single integer n ($1 \leq n \leq 10$) representing the number of disks.

Output Description

Print the sequence of moves required to solve the Tower of Hanoi puzzle. Each move should be printed on a new line.

Sample Input	Sample Output
3	Move disk 1 from peg 1 to peg 3 Move disk 2 from peg 1 to peg 2 Move disk 1 from peg 3 to peg 2 Move disk 3 from peg 1 to peg 3 Move disk 1 from peg 2 to peg 1 Move disk 2 from peg 2 to peg 3 Move disk 1 from peg 1 to peg 3

Table 6: Sample I/O

Note:

The solution should be implemented using recursion. For N disks, the program will make $2^N - 1$ moves.

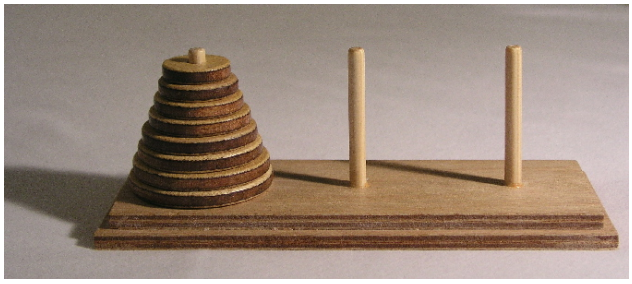


Figure 2: Tower of Hanoi

Problem 6. *Drop out periodic function(D.O.P.F)*

The Drop Out Periodic Function (D.O.P.F) is a renowned mathematical function that elegantly describes the relationship between a college student's changing mindset and the duration of their university studies. This function was invented by ***Li, Wei-Cheng****, a senior student in the Department of Learning Science at National Taiwan Normal University. Thanks to his contribution, we can now fully comprehend this function.*

However, what if we were unaware of the function's exact form and only knew its inputs and outputs? Would it be possible to reconstruct any arbitrary function under these circumstances? For instance, if you were given several pairs of (x_i, y_i) *where* $f(x_i) = y_i$*, could you recreate the D.O.P.F function?*

We'll assume the function is an nth-degree polynomial, and you'll be provided with n+1 valid points. Your program should implement the Lagrange Interpolation Method to solve this problem, reconstructing the original function based on the given data points.

Input Description

The input data consists of $N + 1$ lines. N represents the number of subsequent data lines (where $2 \leq N \leq 10$), and it also indicates that the polynomial being simulated is of degree $N-1$. The following N lines represent N data points, each containing two integers separated by a space. The first integer represents X_i (where $0 < X_i < 10^6$), and the second integer represents Y_i (where $0 < Y_i < 10^6$). The function that guarantees the answer will be a polynomial with all integer coefficients.

Sample Input	Sample Output
2 1 3 2 5 3 7	f(x) = 2x+1
2 10 -20 20 -40 30 -60	f(x) = -2x+0

Table 7: Sample I/O

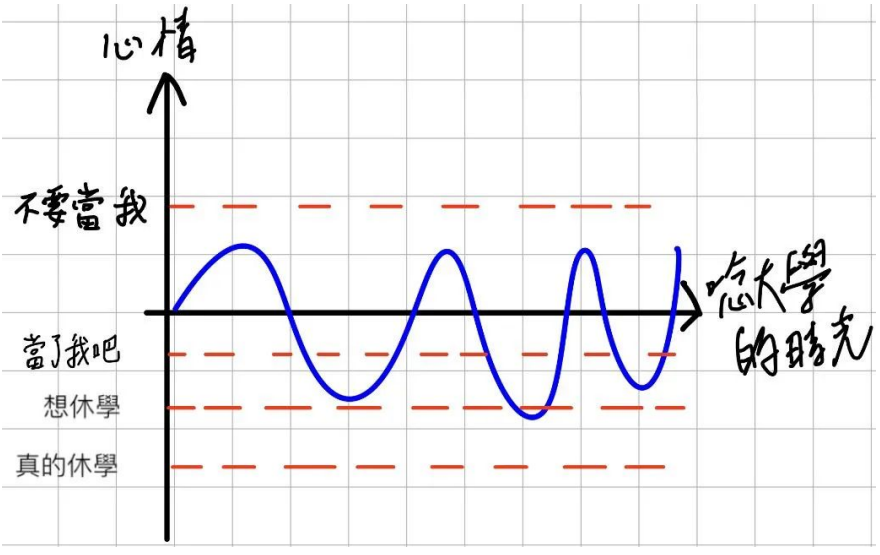


Figure 3: Drop out periodic function (This image is from Li,Wei-Cheng 's Thread)

Lagrange Interpolation Method Explained

The Lagrange Interpolation Method is a way to find a polynomial that passes through a given set of points. It's like connecting the dots, but with a smooth curve!

Basic Idea:

1. We have some points (x, y) that we know our function passes through.
2. We want to find a polynomial that goes through all these points.

How it works:

1. For each point, we create a special polynomial called a Lagrange basis polynomial.
2. We combine these polynomials to get our final function.

Let's break it down step-by-step:

Step 1: Lagrange Basis Polynomials

For each point (x_1, y_1), we create a polynomial $L_1(x)$ that: - Equals 1 when $x = x_1$ - Equals 0 when x is any other given x-value

The formula for this is:

$$L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Where:

- $\prod$means multiply everything together
- $\prod_{j \neq i}$means we do this for all j, except when j = i

Step 2: Combining the Polynomials

We multiply each Lagrange basis polynomial by its y-value and add them all together:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

This P(x) is our final interpolated polynomial!

Example:

Let's say we have three points: (1, 1), (2, 4), and (3, 9).

1. First, we create the Lagrange basis polynomials:

$$L_1(x) = \frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{x^2 - 5x + 6}{2}$$

$$L_2(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)} = -x^2 + 4x - 3$$

$$L_3(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)} = \frac{x^2 - 3x + 2}{2}$$

2. Now we combine them:

$$P(x) = 1 \cdot L_1(x) + 4 \cdot L_2(x) + 9 \cdot L_3(x)$$

3. Simplify:

$$P(x) = 1 \cdot (\frac{x^2 - 5x + 6}{2}) + 4 \cdot (-x^2 + 4x - 3) + 9 \cdot (\frac{x^2 - 3x + 2}{2})$$

$$P(x) = x^2$$

And there you have it! We found that the polynomial passing through these points is x^2. To apply this to your assignment:

1. Create Lagrange basis polynomials for each given point.
2. Combine them as shown above.
3. Simplify the resulting polynomial.
4. For each x value, calculate P(x) to get your final answer.