Sudoku Solver (70 points)

A Sudoku puzzle consists of a 9-by-9 grid divided into 3-by-3 sub-grids. The objective is to fill the grid with digits from 1 to 9, such that:

- Each row contains all the digits from 1 to 9 without repetition.
- Each column contains all the digits from 1 to 9 without repetition.
- Each 3x3 sub-grid contains all the digits from 1 to 9 without repetition.

You are given a partially filled 9-by-9 grid where some cells contain numbers (1-9) and others are empty, represented by 'X'. Your task is to write a program that fills in the blanks to complete the puzzle according to the rules of Sudoku. The input grid is guaranteed to have a unique solution.

Input:
The input consists of a single 9-by-9 grid of characters, where each character represents a cell in the grid:

- Numbers from 1 to 9 represent filled cells.
- X represents an empty cell.
- Each character is separated by a single space.
- Each row of the grid is provided on a new line.

Output:
The output should be a single 9-by-9 grid of numbers, each separated by a single space, representing the solved Sudoku puzzle. Each row of the grid should be printed on a new line. Each cell should contain a digit from 1 to 9, ensuring that the solution follows the Sudoku rules.

Example

Input:

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

Output:

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

Password Checker (50 points)

A password is considered strong if all the below conditions are met:
It has at least 6 characters and at most 20 characters.
It contains at least one lowercase letter, at least one uppercase letter, at least one digit, and at least one special character.
It does not contain three repeating characters in a row (i.e., "Baaabb0" is weak, but "Baaba0" is strong).

Given a password, return the minimum number of steps required to make the provided password strong. If the password is already strong, return 0.

A step can be one of the following:
Insert one character to password,
Delete one character from password, or
Replace one character of password with another character.

Input
Each test case will consist of a password. The password may contain uppercase letters, lowercase letters, digits, and special characters. The available special characters are !@#$%^&*().

Output
For each test case, output the minimum number of steps required to make the password strong.

Examples
Input:
AA1

Output:
3

Input:
1337F3f2

Output:
1

Color Switches (60 points)

There is a rectangular grid with N rows and N columns, with ($0 < N < 100$). Each cell in the grid is one color of the set (Red, White, Blue). Two cells are called adjacent if they share a side. In one move you can choose any two adjacent cells of different colors and change them both to the third color. For example, if two adjacent cells have colors Red and White, then you can change both cells to Blue. You may not choose two non-adjacent cells or two cells of the same color. The goal is to switch the cells so that all the cells are of the same color. It doesn't matter which of the three colors you end.

Input
Each test case will consist of N lines and each line will have N entries where each entry is one of the three colors separated by a blank space. There will be 10 test cases.

Output
1 number that represents the minimum number of switches it will take to convert all the cells in the grid to a single color.

Examples:
Input:
Red White Blue
Blue White Red
White Red Blue

Red Red Red Red
Blue White Red Red
Red Red Red Red
Red Red Red Red

Output:
3
1

Sorting Efficiency (75 points)

There are many sorting algorithms that each have their own strengths and weaknesses. Some are very efficient at finding and fixing outliers while others are better at handling large amounts of random data. Given an array of numbers and a small list of sorting algorithms, write a program that determines which algorithm reaches a fully sorted list in the least number of steps. Then output the name of the sorting algorithm and the number of steps required to fully sort the array. If multiple algorithms reach a sorted array in the same number of steps, output all of their names followed by the number of steps (i.e. "merge bucket 52"). Numbers will be between 0 and 999 inclusively and you will always be trying to sort numbers in ascending order. Each algorithm will define what counts as a step.

You will be using the following sorting algorithms:

Bubble Sort – Traverse through the array starting at the first element and comparing it to the next element. Repeatedly swap adjacent elements until the current element bubbles up to the correct position. Repeat this process until all elements have been iterated over. Each swap counts as two steps.

Insertion Sort – Create a second array that is the same length as the unsorted array. This will be your sorted array. Select an element from the unsorted array and linearly search for the correct position in the sorted array. Continue until all elements are sorted correctly. Each insertion into the sorted array will be one step. If a number needs to be moved to allow a different number to take its place, that will count as an additional step; one for inserting the original number, and one for inserting the new number.

Merge Sort – Divide the original array into two smaller subarrays. Continue recursively dividing the arrays until each one contains only a single element. Then start merging the subarrays back together while sorting the elements at the same time. When merging two arrays that have more than one element, compare the first two elements and insert the smaller one into the first available spot in the merging array. Each insertion into any array will be counted as a step.

Bucket Sort – Numbers from the original array will be first sorted into buckets before being redistributed into a sorted array. The array of buckets will be the same length as the original array of numbers and the following equation can be used to determine which bucket an element should fall in:
ValueOfElement * NumberOfElements / (MaxElementValue + 1)
After each element has been put into a bucket, traverse the bucket array in ascending order and insert elements into the sorted array. If there are multiple elements in a single bucket, make sure to always pull the smallest value. Each insertion into the sorted array will count as a step. Every insertion into a bucket will count for as many steps as elements in the bucket. The first element added to a bucket is one step, the second element is two steps, the third element is three steps, etc. So three elements in a bucket will count as six steps total.

Examples

Input:
1, 3, 5, 7, 9, 2, 4, 6, 8
10, 20, 30, 40, 60, 70, 80, 90, 50, 100

Output:
insertion 19
bubble 8