

# **ICPC Practice Set 2**

## **Boulva**

You have just heard about a new game called Boulva that is being played by many of your friends. In this game, you start with a row of coins of various values. There is always an even number of coins at the beginning. You and your opponent alternate turns. In each turn, the player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of that coin in points. After the last coin has been selected, the player with the most points wins.

Being the smart programmer you are, you realize that it is possible to determine the maximum number of points that you can be guaranteed to win. That is, no matter what the opponent selects, you will have at least that number of points. For simplicity, you decide that you only want to use your program if it is your turn first. You may assume that all coins have a positive value.

### **Input:**

The input consists of multiple games in which you will determine your maximum number of guaranteed points. The first line will contain an integer indicating the number of games for which you will be running your program. Each game is shown in the input by two lines. The first contains an even integer,  $n$ , indicating the number of coins in the starting position. The next contains  $n$  integers representing the values of each coin on the line at the beginning of the game.

### **Output:**

For each case, you should output a single line indicating the case number and the computed maximum number of coins in the following format:

Case X: #

Where X is the case number and # is the maximum number of guaranteed points that your player can receive.

### **Sample Input:**

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

### **Sample Output:**

```
Case 1: 14
Case 2: 13
```

## Making Change

You have just begun working as a cashier at a new international market. The shop that you work at accepts money in many different currencies which causes a problem for you when you are making change for your customers.

To solve this, you decide to write a program that takes a list arbitrary coin values and gives the best way to make change for any amount of money. In this case, you assume that the best way to make change is the way that uses as few coins as possible. You may also assume that all coin values are positive integers.

### Input:

The input will consist of currency information and values to run through your program. The first line will be a number,  $n$ , representing the number of different coin values available in a certain currency. The next  $n$  lines will be the value of each coin that the name of that particular coin. The next line will contain a number,  $m$ , representing the number of values for which you will compute the best possible way to make change. Finally, the next  $m$  lines will contain the values.

The first coin value for the currency will always be 1 to ensure that it is possible to make change for any integer values.

### Output:

For each value, you will first output the case number. Next, you will output the name of each type of coin needed and the quantity in the following format:

```
Case X:  
NAME1 NUM1  
NAME2 NUM2  
...
```

Where  $X$  is the case number and  $NAME1$ ,  $NUM1$ ,  $NAME2$ , and  $NUM2$  are the names and numbers of the first and second values of coins needed. Output the number even if it is 0 and output the coin values in the order in which they are given.

### Example Input:

```
4  
1 Penny  
5 Nickel  
10 Dime  
25 Quarter  
2  
77  
18
```

**Example Output:**

Case 1:

Penny 2

Nickel 0

Dime 0

Quarter 3

Case 2:

Penny 3

Nickel 1

Dime 1

Quarter 0

## Programming is fun

As a skilled programmer, you see everything in a programming way. You see that given any string of characters, you can figure out how many times it says the word “programming”. It's easy to look through and find a 'p', and then an 'r', and then an 'o', and so on.

Your task is to write a program that can take any text and print out how many times that text contains the word “programming”. To be more precise, given a text string, you are to determine how many times the string “programming” appears as a sub-sequence of that string. In other words, find a sequence  $s$  of increasing indices into the input string such that the concatenation of `input[s[0]]`, `input[s[1]]`, ..., `input[s[10]]` is the string “programming”.

### Input:

The first line of input will contain the number of test cases,  $n$ . The next  $n$  lines of input contain one test case each. Each test case is a single line of text, containing only lower-case letters and spaces. No line will start or end with a space.

### Output:

For each test case, output a single line containing your answer in the following format:

Case X: #

Where X is the case number and # is your answer

### Example Input:

```
3
prog hhlmn ramming
pprrooggrraammiinng
programing
```

### Example Output:

```
Case 1: 1
Case 2: 256
Case 3: 0
```