

Speed Limit

Source file name: speed.c, speed.cpp or speed.java

Input: speed.in

Output: standar output

Bill and Ted are taking a road trip. But the odometer in their car is broken, so they don't know how many miles they have driven. Fortunately, Bill has a working stopwatch, so they can record their speed and the total time they have driven. Unfortunately, their record keeping strategy is a little odd, so they need help computing the total distance driven. You are to write a program to do this computation.

For example, if their log shows

Speed in miles per hour	Total elapsed time in hours
20	2
30	6
10	7

this means they drove 2 hours at 20 miles per hour, then $6-2=4$ hours at 30 miles per hour, then $7-6=1$ hour at 10 miles per hour. The distance driven is then $(2)(20) + (4)(30) + (1)(10) = 40 + 120 + 10 = 170$ miles. Note that the total elapsed time is always since the beginning of the trip, not since the previous entry in their log.

Input

The input consists of one or more data sets. Each set starts with a line containing an integer n , $1 \leq n \leq 10$, followed by n pairs of values, one pair per line. The first value in a pair, s , is the speed in miles per hour and the second value, t , is the total elapsed time. Both s and t are integers, $1 \leq s \leq 90$ and $1 \leq t \leq 12$. The values for t are always in strictly increasing order. A value of -1 for n signals the end of the input.

Output

For each input set, print the distance driven, followed by a space, followed by the word "miles".

Sample input	Sample Output
3	170 miles
20 2	180 miles
30 6	90 miles
10 7	
2	
60 1	
30 5	
4	
15 1	
25 2	
30 3	
10 5	
-1	

Symmetric Order

Source file name: order.c, order.cpp or order.java

Input: order.in

Output: standar output

In your job at Albatross Circus Management (yes, it's run by a bunch of clowns), you have just finished writing a program whose output is a list of names in nondescending order by length (so that each name is at least as long as the one preceding it). However, your boss does not like the way the output looks, and instead wants the output to appear more symmetric, with the shorter strings at the top and bottom and the longer strings in the middle. His rule is that each pair of names belongs on opposite ends of the list, and the first name in the pair is always in the top part of the list. In the first example set below, Bo and Pat are the first pair, Jean and Kevin the second pair, etc.

Input

The input consists of one or more sets of strings, followed by a final line containing only the value 0. Each set starts with a line containing an integer, n, which is the number of strings in the set, followed by n strings, one per line, sorted in nondescending order by length. None of the strings contain spaces. There is at least one and no more than 15 strings per set. Each string is at most 25 characters long.

Output

For each input set print "SET n" on a line, where n starts at 1, followed by the output set as shown in the sample output.

Sample input	Sample Output
7 Bo Pat Jean Kevin Claude William Marybeth	SET 1 Bo Jean Claude Marybeth William Kevin Pat
6 Jim Ben Zoe Joey Frederick Annabelle 0	SET 2 Jim Zoe Frederick Annabelle Joey Ben

Rounders

Source file name: rounders.c, rounders.cpp or rounders.java

Input: rounders.in

Output: standar output

For a given number, if greater than ten, round it to the nearest ten, then (if that result is greater than 100) take the result and round it to the nearest hundred, then (if that result is greater than 1000) take that number and round it to the nearest thousand, and so on ...

Input

Input to this problem will begin with a line containing a single integer n indicating the number of integers to round. The next n lines each contain a single integer x ($0 \leq x \leq 999999999$).

Output

For each integer in the input, display the rounded integer on its own line.

Note: Round up on fives.

Sample input	Sample Output
9	20
15	10
14	4
4	5
5	100
99	10000000
12345678	50000000
44444445	2000
1445	500
446	

Optimal Parking

Source file name: parking.c, parking.cpp or parking.java

Input: parking.in

Output: standar output

When shopping on Long Street, Michael usually parks his car at some random location, and then walks to the stores he needs. Can you help Michael choose a place to park which minimizes the distance he needs to walk on his shopping round?

Long Street is a straight line, where all positions are integer. You pay for parking in a specific slot, which is an integer position on Long Street. Michael does not want to pay for more than one parking though. He is very strong, and does not mind carrying all the bags around.

Input

The first line of input gives the number of test cases, $1 \leq t \leq 100$. There are two lines for each test case. The first gives the number of stores Michael wants to visit, $1 \leq n \leq 20$, and the second gives their n integer positions on Long Street, $0 \leq X_i \leq 99$.

Output

Output for each test case a line with the minimal distance Michael must walk given optimal parking.

Sample input	Sample Output
2	152
4	70
24 13 89 37	
6	
7 30 41 14 39 42	

Electrical Outlets

Source file name: outlets.c, outlets.cpp or outlets.java

Input: outlets.in

Output: standar output

Roy has just moved into a new apartment. Well, actually the apartment itself is not very new, even dating back to the days before people had electricity in their houses. Because of this, Roy's apartment has only one single wall outlet, so Roy can only power one of his electrical appliances at a time.

Roy likes to watch TV as he works on his computer, and to listen to his HiFi system (on high volume) while he vacuums, so using just the single outlet is not an option. Actually, he wants to have all his appliances connected to a powered outlet, all the time. The answer, of course, is power strips, and Roy has some old ones that he used in his old apartment. However, that apartment had many more wall outlets, so he is not sure whether his power strips will provide him with enough outlets now.

Your task is to help Roy compute how many appliances he can provide with electricity, given a set of power strips. Note that without any power strips, Roy can power one single appliance through the wall outlet. Also, remember that a power strip has to be powered itself to be of any use.

Input

Input will start with a single integer $1 \leq N \leq 20$, indicating the number of test cases to follow. Then follow N lines, each describing a test case. Each test case starts with an integer $1 \leq K \leq 10$, indicating the number of power strips in the test case. Then follow, on the same line, K integers separated by single spaces, $O_1 O_2 \dots O_k$, where $2 \leq O_i \leq 10$, indicating the number of outlets in each power strip.

Output

Output one line per test case, with the maximum number of appliances that can be powered.

Sample input	Sample Output
3	7
3 2 3 4	31
10 4 4 4 4 4 4 4 4 4	37
4 10 10 10 10	

7 segments

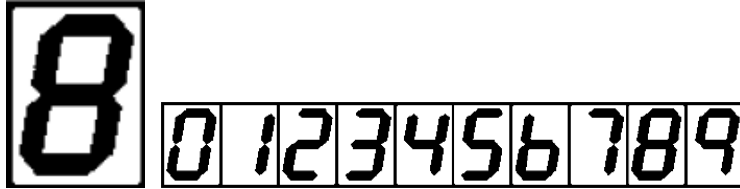
Source file name: segments.c, segments.cpp or segments.java

Input: segments.in

Output: standar output

A 7 segments display is a component built with diodes LED (Light emitter diode) which allow to show decimal numbers.

Next figure shows a 7 segments display and then shows the representation of each digit.

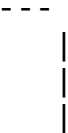
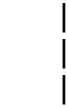



Input

Input consists in several cases. Each one, contain a decimal number and other decimal number between 1 and 5 separated by a space.

Output

For each case, you must print the first number with the number of characters determined by the second number for each segment.

Sample input	Sample Output
3 3 8 2	  

Making Book

Source file name: book.c, book.cpp or book.java

Input: book.in

Output: standar output

A printer - who still uses moveable type - is preparing to print a set of pages for a book. These pages are to be numbered, as usual. The printer needs to know how many instances of each decimal digit will be required to set up the page numbers in the section of the book to be printed.

For example, if pages 10, 11, 12, 13, 14 and 15 are to be printed, computing the number of digits is relatively simple: just look at the page numbers that will appear, and count the number of times each digit appears. The digit 0 appears only once, the digit 1 appears 7 times, the digits 2, 3, 4 and 5 each appear once, and 6, 7, 8 and 9 don't appear at all.

Your task in this problem is to provide the printer with the appropriate counts of the digits. You will be given the numbers of the two pages that identify the section of the book to be printed. You may safely assume that all pages in that section are to be numbered, that no leading zeroes will be printed, that page numbers are positive, and that no page will have more than three digits in its page number.

Input

There will be multiple cases to consider. The input for each case has two integers, **A** and **B**, each of which is guaranteed to be positive. These identify the pages to be printed. That is, each integer **P** between **A** and **B**, including **A** and **B**, is to be printed. A single zero will follow the input for the last case.

Output

For each input case, display the case number (1, 2, . . .) and the number of occurrences of each decimal digit 0 through 9 in the specified range of page numbers. Display your results in the format shown in the examples below.

Sample input	Sample Output
10 15	Case 1: 0:1 1:7 2:1 3:1 4:1 5:1 6:0 7:0 8:0 9:0
912 912	Case 2: 0:0 1:1 2:1 3:0 4:0 5:0 6:0 7:0 8:0 9:1
900 999	Case 3: 0:20 1:20 2:20 3:20 4:20 5:20 6:20 7:20 8:20 9:120
0	

Quicksum

Source file name: quicksum.c, quicksum.cpp or quicksum.java

Input: quicksum.in

Output: standar output

A checksum is an algorithm that scans a packet of data and returns a single number. The idea is that if the packet is changed, the checksum will also change, so checksums are often used for detecting transmission errors, validating document contents, and in many other situations where it is necessary to detect undesirable changes in data. For this problem, you will implement a checksum algorithm called Quicksum. A Quicksum packet allows only uppercase letters and spaces. It always begins and ends with an uppercase letter. Otherwise, spaces and letters can occur in any combination, including consecutive spaces.

A Quicksum is the sum of the products of each character's position in the packet times the character's value. A space has a value of zero, while letters have a value equal to their position in the alphabet. So, A = 1, B = 2, etc., through Z = 26. Here are example Quicksum calculations for the packets "ACM" and "MID CENTRAL":

ACM: $1 \times 1 + 2 \times 3 + 3 \times 13 = 46$

MID CENTRAL: $1 \times 13 + 2 \times 9 + 3 \times 4 + 4 \times 0 + 5 \times 3 + 6 \times 5 + 7 \times 14 + 8 \times 20 + 9 \times 18 + 10 \times 1 + 11 \times 12 = 650$

Input

The input consists of one or more packets followed by a line containing only # that signals the end of the input. Each packet is on a line by itself, does not begin or end with a space, and contains from 1 to 255 characters.

Output

For each packet, output its Quicksum on a separate line in the output.

Sample input	Sample Output
ACM	46
MID CENTRAL	650
REGIONAL PROGRAMMING CONTEST	4690
ACN	49
A C M	75
ABC	14
BBC	15
#	