Problem 1 – Happy Hunting

Professor Plum's wife loves egg hunts. But her nearsightedness forces her to use a non-optimal search strategy. She always goes to the egg nearest to her current location, even if it means backtracking during a hunt. Determine the order in which Mrs. Plum gathers eggs for a collection of egg hunts.

There are some things you should know. Two eggs will never be the same distance away. And Mrs. Plum lives in a one-dimensional world.

Input

The first line contains the number of hunts for which you have data. Each of the following lines contain the information about an egg hunt. The first number on each line is Mrs. Plum's current location in her 1-D world. The second number is the number of eggs in the hunt. The remaining numbers on the line are the locations of the eggs in the 1-D world. For example, in the first hunt in the example input below, Mrs. Plum is initially located at position 14, and the eggs are located at positions 20, 32, 10, and -1.

```
2
14 4 20 32 10 -1
50 5 56 3 8 82 203
```

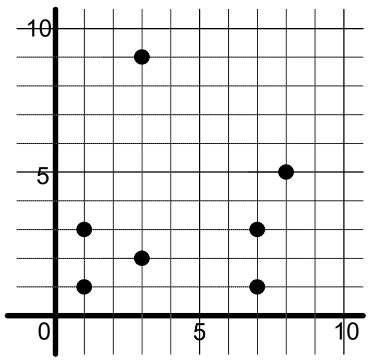
Output

For each hunt, print to standard output a case label and the locations of the eggs in the order Mrs. Plum collects them. For the example input given above, the output is:

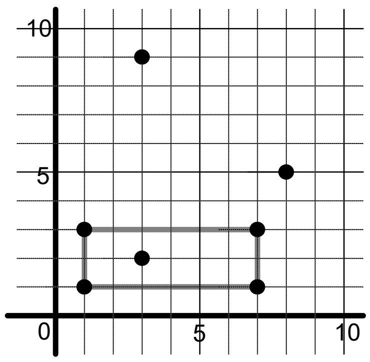
```
Case 1: 10 20 32 -1
Case 2: 56 82 100 8 3 203
```

Problem 2 – Zukei Puzzle

In a Zukei puzzle, the player is given a set of seemingly random coordinates in the Cartesian plane. The player must find a geometric shape whose vertices are in the set. In this problem, the player must find only rectangles:



(a) Random coordinates?



(b) Nope, a rectangle!

Figure 1: A Zukei puzzle with seven vertices.

Identify which vertices form rectangles in several Zukei puzzles given as input. Assume that only one rectangle is hidden within each puzzle and that rectangle sides are aligned with the X and Y axes – they are not rotated.

Input

The first line contains the number of puzzles. Each of the following lines contain the information about a puzzle. A puzzle line starts with the number of 2-D vertices followed by pairs of integers for the vertex coordinates, listed in XY order. For example, the first puzzle in the example input below has 7 vertices. The first vertex is (1, 1), and the final vertex is (8, 5).

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

Output

For each puzzle, print to standard output a case label and the vertices that form the rectangle in the puzzle. Lower vertices (i.e., smaller Y) appear before higher ones. In the case of a tie, the vertex farther to the left (i.e., smaller X) is printed first. For the example input given above, the output is:

```
Case 1: 1 1 7 1 1 3 7 3
Case 2: 5 4 8 4 5 9 8 9
```

Problem 3 – Alternative Routes

As a college student Professor Plum worked one summer as a traveling salesperson. Unfortunately, Professor Plum was fired for taking too long to plan his routes. As he walked home from his (former) office, he took great solace in counting the number of routes he might take to arrive home. For example, if his home was 2 blocks south and 3 blocks east, he has exactly 10 routes to choose from:

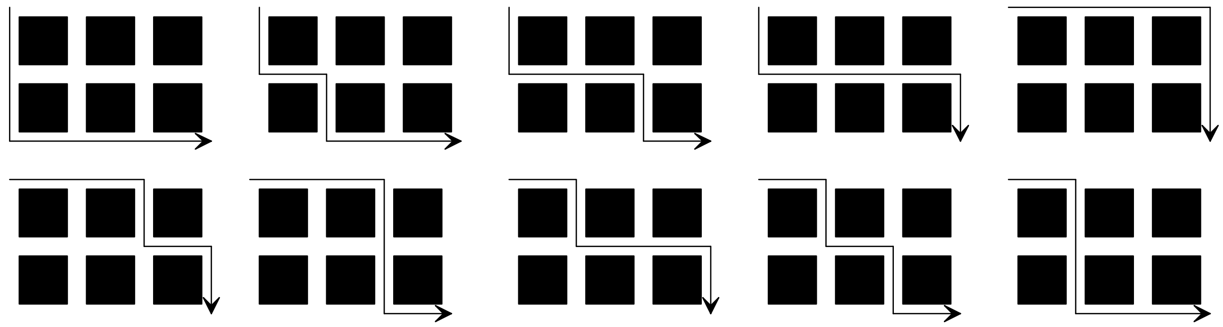


Figure 2: The possible routes from the office at the top-left corner to Plum's home at the bottom-right.

Professor Plum strives to be efficient, so he'll never take a turn that will lead him away from his home. Compute the number of routes that Professor Plum can take for a collection of distances between the office and his home. The number of routes will always fit in a 64-bit integer.

Input

The first line contains the number of neighborhood configurations. Each of the following lines specifies a neighborhood configuration by two integers: neighborhood's width followed by its height. For example, the first neighborhood in the example input below is 3x2, as shown in the figure above.

```
2
3 2
20 4
```

Output

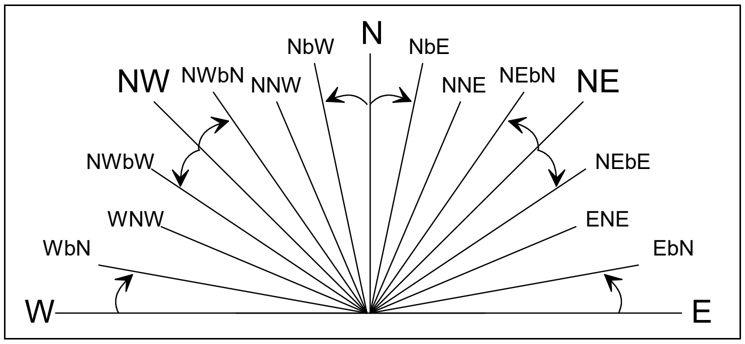
For each neighborhood, print to standard output a case label and the number of routes Profess Plum may take to get home. For the example input given above, the output is:

```
Case 1: 10
Case 2: 10626
```

Problem 4 – Golf Pointers

Professor Plum is trying to teach his friend Professor East to play golf. Professor East can hit the ball along ways (300+ yards), so he likes to "cut dog-legs", i.e., tee-off directly for the hole even if the fairway turns to the left or right. Professor East being unfamiliar with the course wants Professor Plum to specify a direction to the hole from the tee. Professor Plum would like to specify an angle from straight down the fairway at the tee, e.g., '20 degrees to the right", or "15 degrees to the left." Unfortunately, Professor East is not very good at angles, but being an "old navy man" would prefer a compass direction.

Professor Plum is familiar with the 4 *cardinal* directions (N,E,S,W) and the 8 *principal* compass directions: N, NE, E, SE, S, SW, W, NW. He is also familiar with the 16 compass directions: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW which have 22.5 degrees between them. To get a finer granularity, Professor East explains the 32 compass directions which includes 16 more compass directions between the original 16: N, NbE, NNE, NEbN NE, NEbE, ENE, EbN, E, EbS, ESE, SEbE, SE, SEbS, SSE, SbE, S, SbW, SSW, SWbS, SW, SWbW, WSW, WbS, W, WbN, WNW, NWbW, NW, NWbN, NNW, NbW. The name XbY stands for "X by Y" which is short for "from principal direction X toward cardinal direction Y.



Professor Plum doesn't think that 32 compass points is enough being 11.25 degrees apart, so he proposes adding 32 more compass points in between these labeled "XtY" short for X toward cardinal direction Y. Here the X's are the 16 original compass directions. From N to E in the clockwise direction the compass points are: N, NtE, NbE, NNEtN, NNE, NNEtE, NEbN, NEtN, NE, NEtE, NEbE, ENEtN, ENE, ENEtE, EbN, EtN, E.

Professor Plum wants you to write a program that takes inputs like '20 degrees to the right" or '15 degrees to the left" and determines the closed of the 64 compass points where N is always assumed to be straight down the fairway at the tee.

Input

The first line contains the number of directions to convert. Each of the following lines specifies a direction as "X degrees to the Y," where X is an integer from 0 to 90 and Y is either "left" or "right." The below sample input has 5 directions to convert.

```
5
20 degrees to the right
15 degrees to the left
0 degrees to the left
35 degrees to the left
18 degrees to the right
```

Output

For each direction, print to standard output a case label and the closed of the 64 compass points. For the example input given above, the output is:

```
Case 1: NNE
Case 2: NNWtN
Case 3: N
Case 4: NWbN
Case 5: NNEtN
```

Problem 5 – MICS L

Professor Plum likes the idea of MICS 2017 being the 50th anniversary, or 'L" in Roman numerals. He wants you to write a program to generate ASCII art printing "MICS L" for a sign to tape on the back of the MICS's trip van. Since he does not know the dimensions of the sign, so he wants your program to take as input positive integer scaling factors.

Scaling Factor	Letter Dimension of MICS (# chars × # chars)	Line Width of All Letters (# chars)	Blank Lines Between Letters MICS	Letter Dimension of L (# chars × # chars)	Blanks Between MICS and L
1	5 × 5	1	1	23 × 20	5
2	10 × 10	2	2	46 × 40	10
3	15 × 15	3	3	69 × 60	15
10	50 × 50	10	10	230 × 200	50

A scaling factor of 1 would produce:

```
Case 1:
M   M     L
MM MM     L
M M M     L
M   M     L
M   M     L
          L
IIIII     L
  I       L
  I       L
  I       L
IIIII     L
          L
CCCCC     L
C         L
C         L
C         L
CCCCC     L
          L
SSSSS     L
S         L
SSSSS     L
    S     L
SSSSS     LLLLLLLLLLLLLLLLLLLL
```

A scaling factor of 2 would produce:

```
Case 2:
MM      MM          LL
MMM    MMM          LL
MMMM  MMMM          LL
MM MMMM MM          LL
MM  MM  MM          LL
MM      MM          LL
MM      MM          LL
MM      MM          LL
MM      MM          LL
MM      MM          LL
                    LL
                    LL
IIIIIIIIII          LL
IIIIIIIIII          LL
    II              LL
    II              LL
    II              LL
    II              LL
    II              LL
    II              LL
IIIIIIIIII          LL
IIIIIIIIII          LL
                    LL
                    LL
CCCCCCCCCC          LL
CCCCCCCCCC          LL
CC                  LL
CC                  LL
CC                  LL
CC                  LL
CC                  LL
CC                  LL
CCCCCCCCCC          LL
CCCCCCCCCC          LL
                    LL
                    LL
SSSSSSSSSS          LL
SSSSSSSSSS          LL
SS                  LL
SS                  LL
SSSSSSSSSS          LL
SSSSSSSSSS          LL
        SS          LL
        SS          LL
SSSSSSSSSS          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
SSSSSSSSSS          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

Input

The first line contains the number of scaling factors. Each of the following lines contains a single positive integer scaling factor. The below sample input has 3 scaling factors.

```
3
1
2
3
```

Output

The output should contain the ASCII art for each sign corresponding to the scaling factor specified by the input.

NOTE: All lines for a sign should be the same length by padding shorter lines with blanks. No blank lines are between cases. Output for above input is spread cross the three "boxes" (see next page too).

A scaling factor of 3 would produce:

```
Case 3:
MMM         MMM               LLL
MMMM       MMMM               LLL
MMMMM     MMMMM               LLL
MMMMMM   MMMMMM               LLL
MMM MMM MMM MMM               LLL
MMM  MMMMM  MMM               LLL
MMM   MMM   MMM               LLL
MMM    M    MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
MMM         MMM               LLL
                              LLL
                              LLL
                              LLL
IIIIIIIIIIIIIII               LLL
IIIIIIIIIIIIIII               LLL
IIIIIIIIIIIIIII               LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
      III                     LLL
IIIIIIIIIIIIIII               LLL
IIIIIIIIIIIIIII               LLL
IIIIIIIIIIIIIII               LLL
                              LLL
                              LLL
                              LLL
CCCCCCCCCCCCCCC               LLL
CCCCCCCCCCCCCCC               LLL
CCCCCCCCCCCCCCC               LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCC                           LLL
CCCCCCCCCCCCCCC               LLL
CCCCCCCCCCCCCCC               LLL
CCCCCCCCCCCCCCC               LLL
                              LLL
                              LLL
                              LLL
SSSSSSSSSSSSSSS               LLL
SSSSSSSSSSSSSSS               LLL
SSSSSSSSSSSSSSS               LLL
SSS                           LLL
SSS                           LLL
SSS                           LLL
SSSSSSSSSSSSSSS               LLL
SSSSSSSSSSSSSSS               LLL
SSSSSSSSSSSSSSS               LLL
            SSS               LLL
            SSS               LLL
            SSS               LLL
SSSSSSSSSSSSSSS               LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
SSSSSSSSSSSSSSS               LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
SSSSSSSSSSSSSSS               LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

Problem 6 – Odd-Even Directional Sort

Professor Plum likes to travel. On a recent trip he noticed that even Interstate highways run east-west and odd Interstates run north-south. Thus, on a recent test he had his students write the following "odd-even directional sort" to rearrange positive integers such that:

- odd values should be interleaved with even values exactly the same way after sorting as before sorting,
- odd values are sorted in ascending order (smallest to largest),
- even values are sorted in descending order (largest to smallest),

Input

The first line contains the number of sorts to perform. Each of the following lines contain the information about a sort. A sort line starts with the number values to be sorted followed by a sequence of positive integers to sort. For example, the first sort in the example input below contains 13 values: 8, 2, ..., 10. Each sort sequence consists of no more than 1,000,000 positive integers. The number of values in each sort (e.g., 13) should NOT to be included in the sort.

```
3
13 8 2 11 6 3 6 5 15 4 7 1 9 10
8 9 7 5 3 1 2 4 6
10 9 3 2 1 5 4 6 7 8 10
```

Output

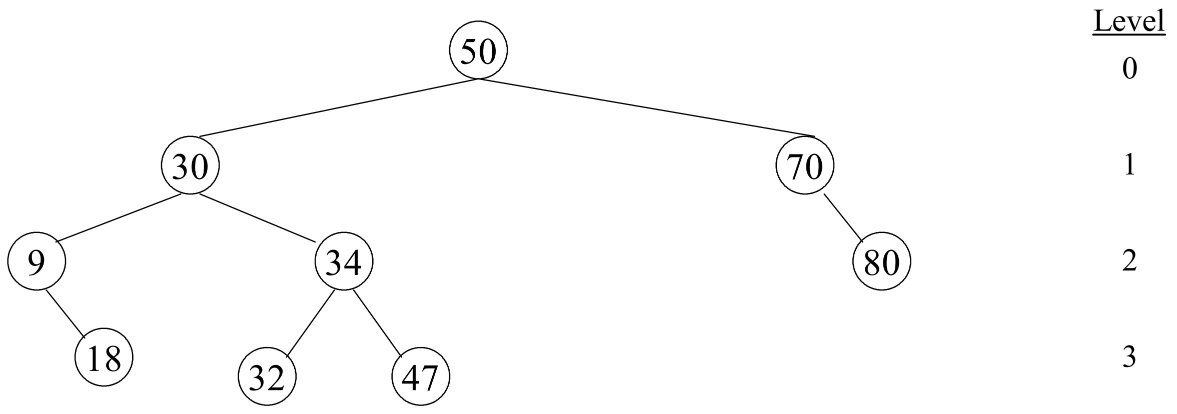
For each sort, print to standard output a case label followed by the sequence of positive integers rearranged in "odd-even directional" sorted order as defined above. For the example input given above, the output is:

```
Case 1: 10 8 1 6 3 6 5 7 4 9 11 15 2
Case 2: 1 3 5 7 9 6 4 2
Case 3: 1 3 10 5 7 8 6 9 4 2
```

Problem 7 – BST Height

Trees are particularly annoying to Professor Plum. He likes to fly kites in the center of campus, but the wind keeps blowing his kites into the trees. Professor Plum has spent enough time observing trees to notice that some are taller than others.

Professor Plum often teaches Data Structures so he knows that the same thing can happen in binary search trees (BSTs). Recall that for each node in a Binary Search Tree (BST) all values in the left-subtree are < the root node and all values in the right-subtree are > the root node. The shape of a BST depends on the order in which values are added. For example, if we start with an empty BST and insert the sequence of values: 50, 70, 30, 80, 34, 32, 9, 47, 18, then we get the BST:



He wants you to write a program to determine the height of the BST given a sequence of inserted values. The height of the BST is defined to be the maximum level of the tree. For the above input sequence, the program should report a BST height of 3.

Input

The first line contains the number of BSTs. Each of the following lines contain the information about a BST. A BST line starts with the number values to be inserted into the BST followed by a sequence of integers to insert. For example, the first BST in the example input below matches the above BST.

```
3
9 50 70 30 80 34 32 9 47 18
5 2 4 6 8 10
12 4 50 30 20 10 70 65 8 90 100 130 120
```

Output

For each BST, print to standard output a case label and the height of the BST. For the example input given above, the output is:

```
Case 1: 3
Case 2: 4
Case 3: 6
```

Problem 8 – Wandering Mind

Professor Plums likes recursion, but his students typically find it confusing. During a recent faculty meeting his mind wandered, and he invented the following recursive mathematical function, F(n):

$$F(n) = n \text{ for all value of } n \leq -3$$
$$F(n) = 2n \text{ for all value of } -3 < n < 3$$
$$F(n) = F(n-6) + F(n-4) + F(n-1) \text{ for all values of } n \geq 3.$$

He wants you to write a program to compute values of the function F(n).

Input

The first line contains the number of n values to run through the function F(n). Each of the following lines contain a single integer value of n. All of the values of n and corresponding F(n) values will fit into a 64-bit signed integer. The below sample input contains three n values.

```
3
5
9
-8
```

Output

For each n value, print to standard output a case label and the value of F(n) as defined above. For the example input given above, the output is:

```
Case 1: -5
Case 2: -7
Case 3: -8
```

Problem 9 – Secret Messages

Professor Plum's wife enjoys reading "spy" novels. She likes to send him encoded messages. He hates it, but at least her encryption scheme is fairly simple. She uses a form of substitution cipher based on an integer key and the follow sequence of 78 characters: (Note: blank-space character between the "K" and "l")

```
aA0bB1cC2dD3eE4fF5gG6hH7iI8jJ9kK lL,mM.nN?oO/pP;qQ:rR'sS"tT!uU@vV$wW%xX&yY-zZ=
```

The integer key is used to encrypt **only the first letter** in the message by "shifting" down the above sequence by that amount. For example, shifting 5 from 'B' encrypts as 'd'. Shifting past the right end of the sequence wraps back to the beginning of the sequence. For example, shifting 5 from 'Z' encrypts as 'b'.

The sequence position of the first letter in the message is used as the shift amount for the second letter in the message. If the first letter was a 'B,' then 4 is used to shift the second letter in the message since 'B' is at position 4 (start with 'a' at position 0, 'A' at position 1, etc.). The sequence position of the second letter in the message is used as the shift amount for the third letter, etc. Any message character not in the above sequence (e.g., '<', '{') is copied to the encrypted message without modification with the previous shift amount carrying over to the next letter in the message that's in the above sequence. Consider the following example with key 5:

Four Line Message to Encrypt with Key = 5	Encrypted Message
`Be by the Union at 6 PM!`	`dF/,z83al/fHvwb& t3RRa1g`
`<We'll plan MICS 2018...>`	`<qA$2w$=aln&x@ @dNDCKVZZ>`
`- Sam`	`LJdSm`

Input

The first line contains the integer key used to encrypt the original message. The second line contains the sequence of 78 characters. The third line contains the number of lines in the message. The following lines contain an encrypted message.

```
5
aA0bB1cC2dD3eE4fF5gG6hH7iI8jJ9kK lL,mM.nN?oO/pP;qQ:rR'sS"tT!uU@vV$wW%xX&yY-zZ=
4
dF/,z83al/fHvwb& t3RRa1g

<qA$2w$=aln&x@ @dNDCKVZZ>
LJdSm
```

Output

The output will be the decrypted message. For the example input given above, the output is:

```
Be by the Union at 6 PM!

<We'll plan MICS 2018...>
- Sam
```