Problem 1 — Roll Over and Die

A standard 6-sided die has the numbers 1 through 6, represented with spots, on each side of the die in such a way that the sum of opposite sides is 7. Suppose we have an infinite checker board oriented so the rows run east-west and the columns run north-south. Furthermore, imagine that each square on the board is exactly the dimensions of one side of our die.

Initially, our die is sitting on a square so that the 1 is on the upwards facing side, the 2 is on the side facing north and the 3 is on the side facing west. (We'll call this the *starting position*.) We then proceed to roll the die from square to square, taking care never to slide it. After a series of rolls, we wish to know what number is on the side facing upwards.

There will be multiple problem instances for this problem. The first line of input will contain the integer n indicating how many problems will follow. Each of the next n lines will contain information for one problem instance. Each line will have a positive integer m (≤ 50) followed by a single space, followed by a string of m characters. Each character will be either `N`, `S`, `E` or `W`. The character indicates which direction to roll the die. We assume the die starts in standard position at the beginning of each problem instance. You should produce one line of output for each problem instance. After executing the m rolls, you are to output the number that appears on the side facing upwards, left-justified on the line.

Example Input:

```
3
5 NNNNN
6 ESSWNN
8 SSSENNNW
```

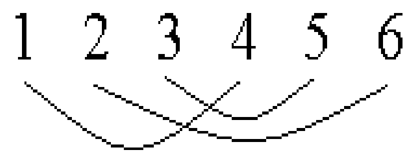
Corresponding Output:

```
5
6
5
```

Problem 2 — Crossing Number

A *perfect matching* of the even integer n is a pairing of all positive integers $1, 2, \ldots, n$. For example $(1, 4)$, $(2, 6)$,$(3, 5)$ is a perfect matching for 6.

If we write the write the integers $1, 2, \ldots, n$ in a row and draw an arc between the paired numbers, we can count the number of times arcs cross. This is known as the *crossing number* of the perfect matching. The diagram below shows that the crossing number of the perfect matching shown above is 2. You may only draw arcs below the list of numbers. You only cross arcs when you are forced to and at most two arcs can intersect at a point.



There will be multiple problem instances for this problem. The information for each problem instance will be on one line. Each of these lines will be of the form

$$m\ n_1\ n_2\ n_3\ n_4\ \cdots n_{m-1}\ n_m$$

where m is an even integer less than 27 and the n_i are a permutation of $1, 2, \ldots, m$ where n_1 and n_2 are matched, n_3 and n_4 are matched and so on. A value of $m = 0$ indicates the end of input and should generate no output.

For each problem instance you are to produce one line of output of the form

`The crossing number is` n`.`

where n is the appropriate number.

Example Input:

```
6 1 4 2 6 3 5
10 2 3 1 8 4 7 6 10 5 9
0
```

Corresponding Output:

```
The crossing number is 2.
The crossing number is 5.
```

Problem 3 — Life at the Top

The buildings in The Big Avocado, as the city is known, are erected according to a very precisely laid out plan. According to the zoning ordinances, there is exactly one building per block and the blocks are all the same length. Furthermore, all the really big buildings in The Big Avocado are along one street. In fact, they are all along one side of one street (the side with the great view). This street is perfectly straight.

On the top floor of each skyscraper is a penthouse and all the occupants of the penthouses are great friends and enjoy nothing better than a good talk with their pals. At those distances it'd be more like a good yell, so they use semaphore. Of course, if there is a skyscraper that is too tall between the two friends, they can't communicate. The problem is to determine if a given pair of friends can communicate. We'll give all heights in units of a block to make things easier to calculate. (So, for instance, a building might be 4 blocks high or 7 blocks high.)

For example, the diagram below shows a situation where the heights of the buildings down this street are 2, 3, 4, 3, 1, 3 and 4.

Then there is a clear line of sight from the top of building 1 to building 3, for example. (The line connecting the top of buildings 1 and 3 just touches the top of building 2; we'll say that this offers a clear line of sight.) However, building 3 and building 5 do not have a clear line of sight.

There will be multiple problem instances. Each problem instance will be given on two lines. The first line will contain the integer m (≤ 30) followed by m positive integers $n_1, n_2, \ldots, n_m$ where n_i is the height of the building at block i. (If $m = 0$, input terminates. In this case there will be no second line of input.) The second line will contain two positive integers k and j, $1 \leq k < j \leq m$. You are to determine if there is a clear line of sight between the top of building k to the top of building j.

For each problem instance, print one line of output: either `No` or `Yes`, accordingly. These messages should be left-justified on the line.

Example Input:

```
7 2 3 4 3 1 3 4
1 3
4 1 2 3 3
1 4
0
```

Corresponding Output:

```
Yes
No
```

Problem 4 — Simple Numbers

Simple numbers are those rationals with denominators that are powers of 2. They are arranged in a heirarchy that is diagrammed on the back of this sheet. Notice each simple number is written in lowest terms and, except for the outside branches of this tree (which are labelled with the integers), the two children of each simple number n are the nearest two simple numbers with denominators twice that of n and the smaller of the two is the left child.

Numbers are classified by their simplicity which corresponds to their depth in the tree. Thus 0 is the simplest number and 1/2 is simpler that -1/16 or 3 but has the same simplicity as 2.

We can use this simplicity to find `simple(`X, Y`)`, where X and Y are two finite sets of simple numbers. The value of `simple(`X, Y`)` is the simplest number strictly larger than all the numbers in X and smaller than all the numbers in Y. The value of `simple(`X, Y`)` is undefined if there is some number of X that is greater than or equal to some number in Y. Note that it is legal to have either X or Y (or both) be empty. For example, if $X = \{1/2, 3/4\}$ and $Y = \emptyset$, then `simple(`X, Y`)`=1.

Your program will read in two sets of numbers and print the value of `simple(`X, Y`)`, if it exists, or the message `undefined`, if it does not.

There will be multiple problem instances for this problem. The first line of input will be the positive integer n (≤ 20) indicating the number of problem instances to follow. Each problem instance will consist of two sets, first X then Y. Each set will be on one line and be of the form $m\ n_1\ n_2\ \cdots\ n_m$ where $m \leq 10$ and n_i is a real number between -100 and 100, inclusive, which is a simple number. (You may assume that the denominator of this simple number will not be so large as to cause round-off problems when the number is stored in a variable of type double.)

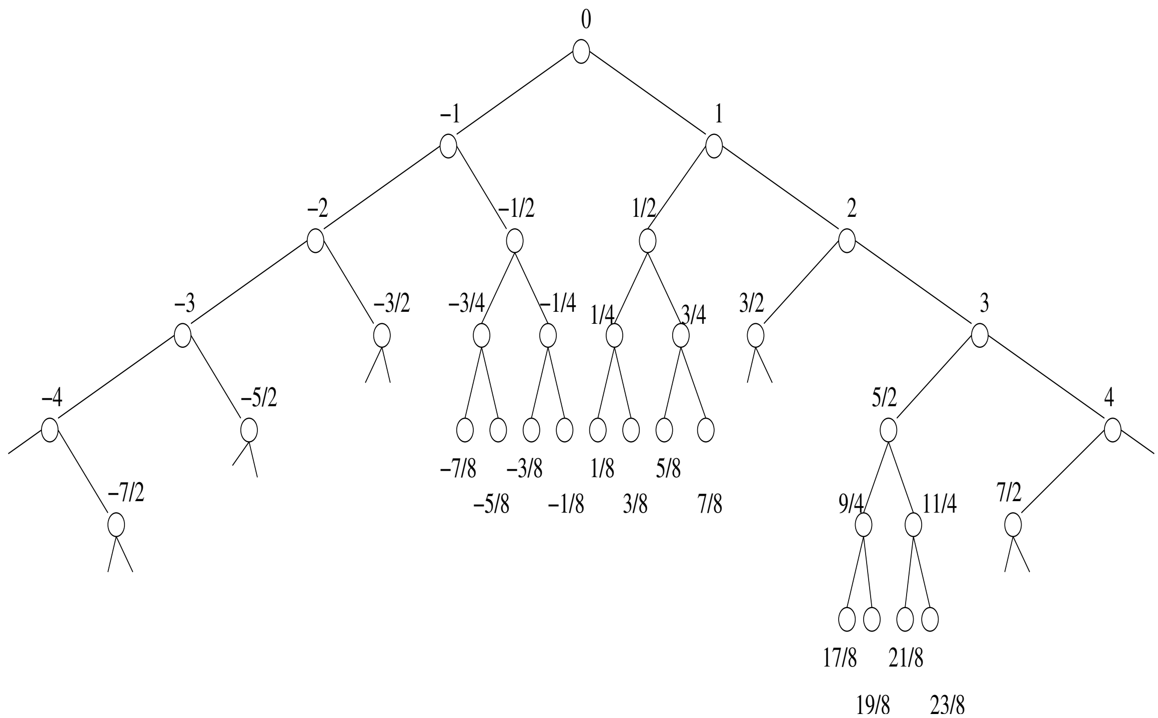
Output should be the value of `simple(`X, Y`)` printed as a floating point number (use no special formatting), if the value of `simple(`X, Y`)` is defined, or the statement `undefined` if the value of `simple(`X, Y`)` is undefined.

Example Input:

```
3
3 0.125 -1.5 0.09375
4 4.5 0.75 2.25 0.875
2 0.5 -0.5
2 1 -3
2 0.5 0.75
0
```

Corresponding Output:

```
0.5
undefined
1
```


0
−1
1
−2
−1/2
1/2
2
−3
−3/2
−3/4
−1/4
1/4
3/4
3/2
3
−4
−5/2
5/2
4
−7/8
−3/8
1/8
5/8
−5/8
−1/8
3/8
7/8
−7/2
9/4
11/4
7/2
17/8
21/8
19/8
23/8

Problem 5 — Cribbage Scorer

The card game Cribbage has several modes of scoring. One is to count points in your hand. To score, you use combinations of the 4 cards in your hand plus the one common card. Points are tallied as follows: 2 points for each set of cards that total to 15 points, 1 point for each card in a run of 3 or more consecutive cards, 2 points for each pair. (You also get a point for "nobs" and a flush, but we won't count those here.) Since we aren't considering suits in this version of Cribbage counting, we'll only input the denominations. Face cards (Jack, Queen and King) each count 10 points and an Ace counts 1 point and is considered the lowest card. (So, it cannot be used in a queen-king-ace run, only an ace-two-three run.)

For example, if the 5 cards you hold are 5 5 5 Jack 7, then you score 8 points for 15's (the three 5-Jack combinations plus the 5-5-5 combination) and 6 points for pairs (the three different pairs of 5's) for a total of 14 points. The hand 2-3-4-5-8 scores 4 points for 15's (the 3-4-8 and 2-5-8 combinations) and 4 points for the 2-3-4-5 run for a total of 8 points.

When scoring runs, you always score the longest possible run. For example, 2-3-4-5 will score 4 points; you may not break it into 2 runs of 3: 2-3-4 and 3-4-5. However, if you use different collections of card for runs, you may count all such runs. For example, J-Q-Q-K has 2 runs of 3 and J-Q-Q-K-K has 4 runs of 3. (Of course there are also pairs to count in the last two examples.)

You will be given the denominations of 5 cards and should print out the point value for the hand.

There will be multiple problem instances for this problem. The first line will contain the positive integer n (≤ 20). Each of the next n lines will be of the form c_1 c_2 c_3 c_4 c_5, where each c_i represents the denomination of a card: either the digit 2, 3, 4, 5, 6, 7, 8 or 9 or the character T, J, Q, K or A.

Each input line should generate exactly one output line of the form

`The value of hand` i `is` p `points.`

where i is the number of the hand (start numbering at 1) and p is the appropriate point value.

Example Input:

```
3
5 5 J 5 7
3 2 4 5 8
K Q T 2 7
```

Corresponding Output:

```
The value of hand 1 is 14 points.
The value of hand 2 is 8 points.
The value of hand 3 is 0 points.
```