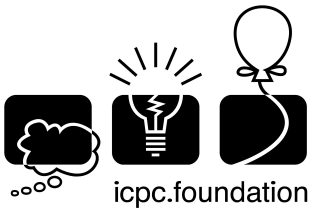


icpc International Collegiate Programming Contest

XXXVIII Maratón Nacional de Programación
Colombia - ACIS / REDIS / CCPL 2024
ICPC

Problems

(This set contains 11 problems; problem pages are numbered from 1 to 18)

General Information. Unless otherwise stated, the conditions stated below hold for all the problems. However, since some problems may have specific requirements, it is important to read the problem statements carefully.

Program name. Each source file (your solution!) must be called

`<codename>.c`, `<codename>.cpp`, `<codename>.java`, or `<codename>.py`

as instructed below each problem title.

Input.

1. The input must be read from the standard input.
2. In most problems, the input can contain several test cases. Each test case is described using a number of lines specific to the problem.
3. In most cases, when a line of input contains several values, they are separated by single blanks. No other spaces appear in the input and there are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. If no end condition is given, then the end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

Output.

1. The output must be written to the standard output.
2. The result of each test case must appear in the output using a number of lines, which depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, if no particular instructions are given, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

A: Arctic Virus

Source file name: `arctic.c`, `arctic.cpp`, `arctic.java`, *or* `arctic.py`
Author: Camilo Rocha

The year is 2045 and climate change has reached an irreversible tipping point. The polar ice caps are melting at an alarming rate, causing global sea levels to rise and destabilizing ecosystems. Scientists around the world have been racing to find solutions to slow the thawing of the poles. Then, one day, they discover something extraordinary hidden deep within the ice --a network of ancient, highly advanced technology. These dormant machines, left behind by an ancient civilization, have the power to control the Earth's temperature.

Reactivating this technology requires solving a series of complex computational problems to restore balance to the planet's climate. Ultimately, these problems reduce to finding specific instances of an ancient virus that can unlock the machines. Unfortunately, this virus no longer ''walks'' the Earth, so advanced in-vivo laboratory experimentation is required to recreate instances of the arctic virus that can trigger the unlocking process.

The virus consists of a chain of bases --adenine (A), cytosine (C), guanine (G), and thymine (T)-- and evolves in one of the following forms:

$$\varphi ::= A \mid T \mid \varphi C \mid A\varphi \mid A\varphi^{-1} \mid G\varphi^{-1}C.$$

In its *simple stage*, the arctic virus takes one of two configurations: A and T. Any other form is considered a *mutation*, which can occur in four ways:

- The notation φC means that the virus φ mutates by adding C to the end of its chain of bases.
- The notation $A\varphi$ means that the virus φ mutates by appending A to the beginning its chain of bases.
- The notation $A\varphi^{-1}$ means that the virus φ mutates by appending A to the beginning of its reversed chain of bases.
- The notation $G\varphi^{-1}C$ means that the virus φ mutates by appending G to the beginning and C to the end of its reversed chain of bases.

For example, A, AT, $GTAC$, and $ACGTCCGA$ are configurations of the virus, while TTG, $CGGAT$, and $GAGAT$ are not.

Your task is to help the scientists unlock the secrets of this ancient technology by writing code to identify configurations of the arctic virus. In particular, you are tasked with designing and implementing an algorithm to detect if a chain of bases created in the laboratory corresponds to the virus, in any of its configurations, or not. This will allow the scientists to verify the results of in-vivo experimentation as quickly and efficiently as possible. Failure means the complete melting of the poles and catastrophic consequences for humanity.

Input

The input consists of several test cases. Each testcase comprises exactly one line of the form n s, with $1 \leq n \leq 1\,000$, and s, with $|s| = n$, a sequence comprising only the characters `A`, `C`, `G`, and `T` representing the four bases that can appear in the virus.

The input must be read from standard input.

Output

For each testcase, output a single line with:

- '`simple`' if s represents a simple stage of the virus;

- '`mutation`' if s represents a mutation of the virus; and
- '`doomed`' if s is not an instance of the arctic virus.

The output must be written to standard output.

Sample Input	**Sample Output**
1 A	simple
2 AT	mutation
4 GTAC	mutation
8 ACGTCCGA	mutation
3 TTG	doomed
5 CGGAT	doomed
5 GAGAT	doomed

B: The Bridge at Night

Source file name: `bridge.c`, `bridge.cpp`, `bridge.java`, *or* `bridge.py`
Author: Rafael García

A group of people want to cross a suspension bridge. Due to the weight of the individuals and the condition of the bridge, no more than two people can cross at the same time. Additionally, it is nighttime and they have only one lamp to light the way. Therefore, whenever two people cross the bridge, one person must return with the lamp if there are still people pending to cross.

The challenge is further complicated by the fact that each person takes a different amount of time to cross the bridge and when two people cross together, they must do so at the pace of the slower individual. What is the shortest time required for the entire group to cross the bridge?

For example, suppose three people, A, B, and C, want to cross the bridge. Their crossing times are 1, 2, and 5 minutes, respectively. The minimum time required for the entire group to cross the bridge is 8 minutes. For instance, A and B can cross first (2 minutes), A returns (1 minute), and then A and C cross together (5 minutes).

Input

The input consists of several test cases. For each test case, the first line contains an integer N ($1 \leq N \leq 30$), representing the number of people in the group. The next N lines each contain an integer t ($1 \leq t \leq 20$), where the k-th line ($1 \leq k \leq N$) represents the time (in minutes) required for the k-th person to cross the bridge. The input ends with a line containing the value 0, which should not be processed.

The input must be read from standard input.

Output

For each test case, output a single line with the minimum time required for the group of N people to cross the bridge.

The output must be written to standard output.

Sample Input	Sample Output
1 10 2 3 20 3 1 2 5 0	10 20 8

C: **SquareDiff**

Source file name: `sqdf.c`, `sqdf.cpp`, `sqdf.java`, *or* `sqdf.py`
Author: Rodrigo Cardoso

Taking a break from her advanced number theory research, Alana began playing with square differences:

- $1^2 - 0^2 = 1$
- $6^2 - 4^2 = 20$
- $10^2 - 5^2 = 75$

However, she has not found two integers x and y such that $x^2 - y^2 = 2$. It might be the case that such a pair does not exist. This made her wonder if there are other numbers that are equally rare.

Alana wants to determine whether a given integer can be expressed as the difference of two squares of integers. Can you help her by writing a program to solve this problem?

Input

The input consists of several test cases. Each test case is defined by a single line containing a positive integer N ($0 < N < 40\,000$), which is the number to test. The input ends with a line containing 0, which should not be processed.

The input must be read from standard input.

Output

For each test case, print a single line with one character: '`Y`' if N can be expressed as the difference of two squares of integers and '`N`' otherwise.

The output must be written to standard output.

Sample Input	**Sample Output**
1 2 3 20 0	Y N Y Y

D: Drug Test

Source file name: `drugtest.c`, `drugtest.cpp`, `drugtest.java`, *or* `drugtest.py`
Author: Rodrigo Cardoso

Active Customizable International Surveys (ACIS) is a company specialized in surveys to determine drug impact and efficacy. They have designed an experiment to test X, a new drug to be used worldwide. The experiment is based on the classical technique of using different concentrations of the drug to measure and compare its effects on test individuals.

Packages of $k \geq 0$ pills, called *k-packages*, are used to define different concentrations of the drug. Each k-package is either filled with pills that contain the active ingredient of X (called *active* package) or filled with placebo pills without X's active ingredient (called *placebo* package). A package without pills (i.e., a 0-package) is called *empty* and is considered a placebo package.

Experiments consider a maximum number of pills per package N and are designed with two conditions:

c1 Each test individual must receive exactly two packages, an active one and a placebo one; if their corresponding sizes are a and p, then $a + p$ must be a power of 2.

c2 For each size k, $0 \leq k \leq N$, at least one individual must receive a k-package.

Your task is to determine if a given package size corresponds to an active or to a placebo package. Indeed, ACIS researchers have proved that there is only one way to design the experiment fulfilling the given conditions.

Consider a scenario with $N = 12$, where packages $1, 2, 4, 5, 8, 9, 10$ are active and packages $0, 3, 6, 7, 11, 12$ are placebo. The experiment may be accomplished by the following pairs of packages (active, placebo):

$$(1, 0), (1, 3), (1, 7), (2, 0), (2, 6), (4, 0), (4, 12), (5, 3), (5, 11), (8, 0), (9, 7), (10, 6).$$

Note that every pair (a, p) in the list is such that a is an active package, p is a placebo package, and $a + p$ is a power of 2. On the other hand, every number from 0 to 12, inclusive, appears in one pair at least. Hence, if the queries are for packages of sizes $1, 3, 5, 7, 9$, the answer is active for $1, 5, 9$ and placebo for $3, 7$.

Input

The input consists of several test cases. Each case is defined by two lines. The first line contains two blank-separated integers N and Q, ($0 < N < 10\,000$ and $0 < Q \leq 100$), the maximum number of pills per package and the number of queries, respectively. The second line contains Q non-negative integer numbers $q_1, q_2, \ldots, q_Q$, ($0 \leq q_k \leq N$ and $0 < k \leq Q$). The input ends with a line containing two blank-separated 0 values.

The input must be read from standard input.

Output

For each test case, output one line with a string $a_1 a_2 \cdots a_Q$ (without blanks between characters), where a_k is `A` if q_k corresponds to the size of an active package and `P` if q_k corresponds to the size of a placebo package.

The output must be written to standard output.

Sample Input	Sample Output
4 3 0 4 2 12 5 1 3 5 7 9 0 0	PAA APAPA

E: Spacebar Tokenizer

Source file name: `spacebar.c`, `spacebar.cpp`, `spacebar.java`, *or* `spacebar.py`
Author: Juan Corena

Spacebar Tokenizer is a startup developing a state-of-the-art tokenizer. Its unique design helps the fastest programmers in the world avoid wasting time hitting the spacebar as they type.

To achieve this goal, the tokenizer uses a scoring system based on a large language model. This model assigns an integer score to each valid token in the language and a score of 0 to any token that is not recognized. When tokenizing a sentence, the tokenizer sums the scores of each token according to the model. An optimal tokenization is one that yields the highest total score out of all possible ways to tokenize the given sentence.

For example: *ilovespacebartokenizer* could be tokenized in several ways including, but not limited to: ''*i love space bar tokenizer*'', ''*i love spacebar tokenizer*'', and ''*ilovespace bartokenizer*''. If the language model assigns a score of 5 to the token *spacebar*, 2 to the token *bar*, and 3 to the token *tokenizer*, then the second tokenization yields the highest score among the 3 example tokenizations, with a total score of 8. Note that there may be cases where multiple tokenizations result in the same maximum score.

The table below shows the scores for the sample tokenizations. The tokens contributing to the final score are in bold:

Tokenization	Score
i love space **bar tokenizer**	5
i love **spacebar tokenizer**	8
ilovespace bartokenizer	0

Your task is to create a program that outputs the score that a tokenizer should output, when it behaves optimally as described.

Input

The input consists of several test cases. Each test case consists of a language model providing the score for each token, and a series of sentences that need to be tokenized optimally according to the language model.

Each test case starts with a line containing two integers $1 \leq m \leq 1\,000$ and $1 \leq s \leq 100$ separated by a single space. Each of the next m lines contains, separated by a blank, a single token in undercase English alphabet and a positive integer number denoting the score that token adds to the overall score of the tokenization. The length of each token is at least 1 and at most 100 characters, the token can be assumed to be unique. Then s lines follow, each contains a sequence a_i for $1 \leq i \leq s$, of undercase English characters to be tokenized, where $1 \leq |a_i| \leq 1\,500$.

The input ends with two blank-separated 0s.

The input must be read from standard input.

Output

For each test case and a_i, output a single line with the optimal tokenization score of a_i according to the corresponding model.

The output must be written to standard output.

Sample Input	**Sample Output**
7 2 space 5 bar 7 spacebar 13 tokenizer 5 i 2 love 4 ipo 8 ilovespacebartokenizer theonlytokenizeripo 0 0	24 13

F: Turnswitch

Source file name: `turnswitch.c`, `turnswitch.cpp`, `turnswitch.java`, *or* `turnswitch.py`
Author: Julián Badillo

The starship Century Hawk is in the middle of a loosing battle with a squad of space pirates, possibly after realizing their business with Captain Hanna Lonely was actually, sort of a scam. There is no escape but to jump to hyperspace. Unfortunately, the Century Hawk has been at the receiving end of heavy fire, and though Captain Lonely is a skilled pilot, the space pirates managed to hit home before she could shield up.

*"It must be the hyperdrive regulator **Turnswitch**, it's not responding!"* - yells the second in command.

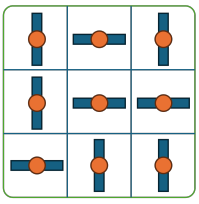
"You, newbie, go to the hyperdrive and fix it up!" - Captain Lonely orders.

As the newbie - and the only crew member of the Century Hawk with any programming skills - your task is to fix the **Turnswitch**, so the hyperdrive is back online.

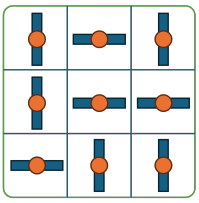
The regulator **Turnswitch** is an $N \times N$ matrix of switches, each switch can be either in horizontal or vertical position.

You discover that if you turn any switch -from horizontal to vertical or viceversa-, all the directly adjacent swiches (up, down, left, right) will also turn. If a switch is on an edge of the matrix, or in a corner, the positions outside the matrix are ignored.

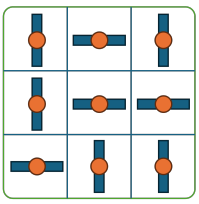
For instance let's consider the following 3×3 **Turnswitch**:



If you turn the switch at the **very center**, the **Turnswitch** will become:



Now, if you turn the switch of the **bottom left corner**, the **Turnswitch** will become:



The Century Hawk can jump to hyperdrive if the switches in the **Turnswitch** are either all horizontal, or all vertical -it is a ship build under a very liberal set of standards. Your task is to find the **least amount of switches** to turn in order to make the Turnswitch ready for the jump.

The fate of the Century Hawk's crew is in your hands.

Input

The input consist of several test cases. Each test case starts with an integer N ($1 \leq N \leq 10$), the size of the Turnswitch. The following N lines contain the Turnswitch matrix. Each line contains N characters, either "|" (the vertical pipe) or "-" (the minus sign), representing the state of the switches. There is a valid solution for each test case. The last line of input will be a 0 (zero) and should not be processed.

The input must be read from standard input.

Output

For each test case, print a single line with the minimum number of switches that need to be turned in order to make the Turnswitch ready for the jump, e.g. all the switches become either horizontal or vertical.

The output must be written to standard output.

Sample Input	**Sample Output**
3 -|| --- |-| 3 ||- |-| -|| 4 -|-| ||-| ||-| ||-- 4 |||- |--- -|-- -|-| 0	3 2 3 3

G: **Signal Coverage**

Source file name: `signal.c`, `signal.cpp`, `signal.java`, *or* `signal.py`
Author: Rafael García

The Nlogonia government is in negotiations with a telecommunications provider to cover a region of the country.

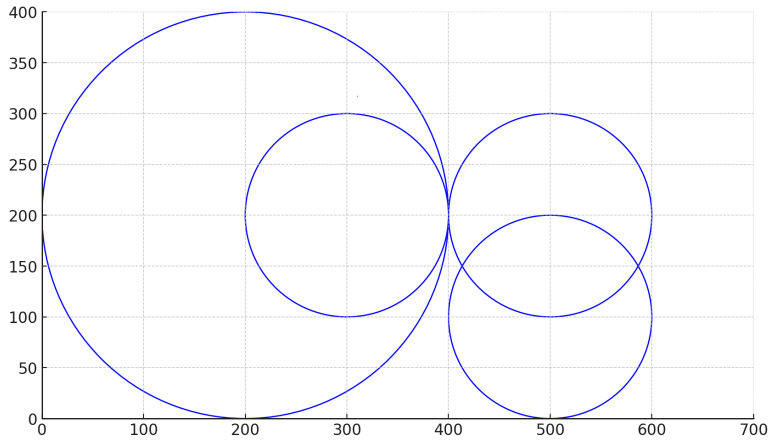
The commitment includes installing n antennas. Each antenna has a coverage radius (which defines a circular area with its border included), a planned location on the map of Nlogonia, and can be active over a given time range. More formally, the n antennas are numbered from 1 to n. The coverage radius of antenna i is r_i, its planned location is (x_i, y_i), it goes into operation at 00:00 on day b_i and goes out of operation at 00:00 on day $e_i + 1$.

For the benefit of both parties (the economy of Nlogonia and the profit for the supplier), two types of restrictions have been imposed:

- Restriction imposed by Nlogonia: No point should be covered by more than one antenna at the same time.
- Restriction imposed by the supplier (pair restrictions): A list of antenna pairs will be provided. For every listed pair, at least one of the antennas must be installed.

Your task is to determine a list of antennas to be installed that meet the restrictions outlined in the agreement, or to conclude that it is impossible to do so.

For instance, consider the 4 antennas illustrated in the figure below. Antenna number 1 has a coverage radius of 200 and is centered at the point $(200, 200)$. The other antennas each have a coverage radius of 100 and are located at $(300, 200)$, $(500, 200)$ and $(500, 100)$, respectively. Assume that all four antennas share the same activation time period. The pair restrictions are $(1, 2)$, $(3, 4)$ and $(1, 4)$. In this case, it is possible to satisfy the restrictions by activating antennas 2 and 4.



Input

The input consists of several lines. The first line contains the number of test cases.

Each case is described as follows:

The first line contains two integers: n, the number of antennas ($2 \leq n \leq 10\,000$), and m, the number of pair restrictions ($1 \leq m \leq 10\,000$). In the next n lines, the i-th line contains five integers: r_i, x_i, y_i, b_i and e_i ($1 \leq r_i, x_i, y_i \leq 5\,000$ and $1 \leq b_i \leq e_i \leq 5\,000$), decribing the i-th antenna. The next m lines describes the pair

restrictions: the j-th line contains two integers: u_j, v_j ($1 \leq u_j, v_j \leq n$ and $u_j \neq v_j$), describing the j-th pair restriction.

The input must be read from standard input.

Output

For each case, if it is impossible to find a list of antennas that satisfy the constraints, output one line with the word `Impossible`. Otherwise, output one line containing an n-long string where the i-th symbol (counting from left to right) is `1` if the i-th antenna is to be installed, and `0` otherwise. If multiple solutions exist, the output corresponding to any one of them should be printed.

The output must be written to standard output.

Sample Input	**Sample Output**
3 2 1 1000 500 500 100 200 100 400 300 20 101 1 2 4 3 200 200 200 1 10 100 300 200 1 10 100 500 200 1 10 100 500 100 1 10 1 2 3 4 1 4 4 3 100 200 300 1 100 100 300 300 1 50 100 200 200 1 25 100 300 200 10 70 1 2 2 3 3 4	01 0101 Impossible

H: **Only1s0s**

Source file name: `only1s0s.c`, `only1s0s.cpp`, `only1s0s.java`, *or* `only1s0s.py`
Author: Rodrigo Cardoso

Alana is a very curious number theory researcher. Last week, she discovered a --perhaps-- useless property about positive integers: for every such number N, there exists an integer M, a multiple of N, that consists only of the digits 1 and 0 in decimal notation. This number M is called an `only1s0s` number.

For instance:

- If $N = 4$, then $M = 10000$ and $\frac{M}{N} = 2500$.
- If $N = 14$, then $M = 10010$ and $\frac{M}{N} = 715$.

Alana has proven that there are many possible solutions for any N. However, she is interested only in the smallest possible M for a given N. Moreover, she wants to determine the value D such that $M = N \cdot D$, where M is the minimum `only1s0s` number.

Can you help her?

Input

The input consists of several test cases. Each test case is defined by one line containing a positive integer N ($0 < N < 10^5$). The input ends with a line containing 0.

The input must be read from standard input.

Output

For each test case, output a single line containing the positive integer D, such that $N \cdot D$ is the smallest `only1s0s` number that is a multiple of N.

The output must be written to standard output.

Sample Input	**Sample Output**
4 14 13 0	25 715 77

I: Omens

Source file name: `omens.c`, `omens.cpp`, `omens.java`, *or* `omens.py`
Author: Julián Badillo

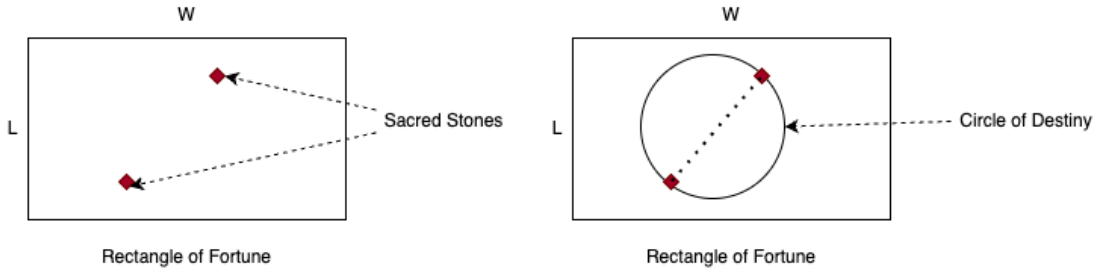
Hanna Lonely, captain of the starship Century Hawk, and her crew, have arrived to the planet C0-3, carrying precious (and dubiously legal) cargo for the Kingdom of Fomana.

The Fomanians are a very superstitious people. Once a year (that's what they call their cicle around their star, CO), they celebrate the *Omens Day*, which happens to be the next day Captain Lonely and her crew arrived to C0-3. On this day, the Royal Oracle performs the divination ritual, by throwing a pair of *Sacred Stones* on the *Rectangle of Fortune*.

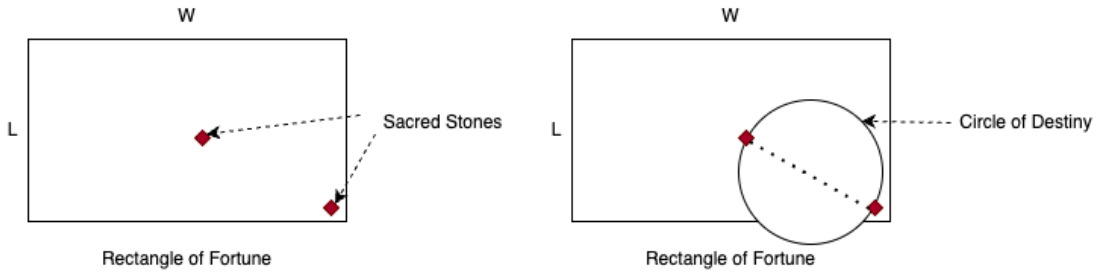
The *Rectangle of Fortune* is an $L \times W$ rectangle, precisely drawn on the sands of the *Sacred Beach*. Once the Royal Oracle has thrown the *Sacred Stones*, she proceeds to draw the *Circle of Destiny*, in which the stones are in exactly oposite sides; the diameter of the circle is the distance between the two stones.

Fomanians believe it's a *good omen*, when the resulting *Circle of Destiny* is completely contained within the *Rectangle of Fortune*, clearly meaning their destiny in good fortune. On the other hand, a *bad omen* is when the *Circle of Destiny* is partially outside the *Rectangle of Fortune*.

This is an example of *good omen*:



This is an example of *bad omen*:



The nature of the *Sacred Stones* is to always fall within the *Rectangle of Fortune*, and you can assume they are equally likely to fall anywhere within the rectangle.

The recently appointed *Royal Oracle* is quite progressive. She went to the Galactic University, and wants Captain Lonely to help her calculate the probability a thrown resulting in a good omen, given the dimensions of the *Rectangle of Fortune*. As the most junior member of the crew, you are tasked with this calculation.

Input

The input consists of several test cases. A case is defined with a line with two positive integer values L and W, $1 \leq L, W \leq 1\,000$, representing the dimensions of the *Rectangle of Fortune*.

```
L W
```

The end of the input is signaled with a line
`0 0`
that should be not processed.

The input must be read from standard input.

Output

For each test case output a line with the probability of the *Circle of Destiny* being completely contained within the *Rectangle of Fortune*, rounded to 4 decimal places.

The output must be written to standard output.

Sample Input	**Sample Output**

`1 1`
`20 40`
`40 30`
`0 0`
	`0.5236`
	`0.3927`
	`0.4909`

J: Lumina

Source file name: `lumina.c`, `lumina.cpp`, `lumina.java`, *or* `lumina.py`
Author: David Yepes

In the magical kingdom of Lumina, the streets are lit by luminous gems embedded in the ground. Each gem emits a radiant aura that can activate nearby gems, creating a chain reaction that illuminates all the gems. However, a powerful dark spell has extinguished all the gems in the kingdom.

The king of Lumina has tasked you with finding the most efficient way to reignite all the gems. Your goal is to determine the minimum number of gems that need to be manually activated so that, through the chain reaction, every gem becomes illuminated.

You are provided with a map of the kingdom, where each gem is represented as a point on the Cartesian plane. Each gem has a radius that indicates the distance its aura can reach to activate other nearby gems. Two gems are considered ''nearby'' if one gem lies within the radius of illumination of the other.

For example, suppose you have 3 gems with the following information:

- Gem 1: Coordinates **(X: -1, Y: 0)** with radius 4.
- Gem 2: Coordinates **(X: 2, Y: 1)** with radius 2.
- Gem 3: Coordinates **(X: 4, Y: -2)** with radius 1.

In this case:

- Gem 1 has a large aura with a radius of 4, covering a significant area.
- Gem 2's aura, with a radius of 2, can also activate nearby gems but is smaller than Gem 1's aura.
- Gem 3 has the smallest aura, with a radius of 1, affecting only a small area.

If you manually activate Gem 1, its large aura is enough to cover Gem 2, but Gem 3 is not covered by the aura. Therefore, in this case, it is necessary to manually activate two gems.

Input

The input consists of multiple test cases. Each test case starts with an integer N ($1 \leq N \leq 5\,000$) representing the number of gems in that test case. Then, N lines follow, each containing three blank-separated integers X, Y, and R, satisfying $-10^9 \leq X, Y \leq 10^9$ and $0 \leq R \leq 10^9$, where X is the x-coordinate of the gem, Y is the y-coordinate of the gem, and R is the radius of the gem's aura. The input terminates with a line containing a single 0, which should not be processed.

The input must be read from standard input.

Output

For each test case, output a single line containing the minimum number of gems that need to be manually activated to ensure that, through the chain reaction, all N gems become illuminated.

The output must be written to standard output.

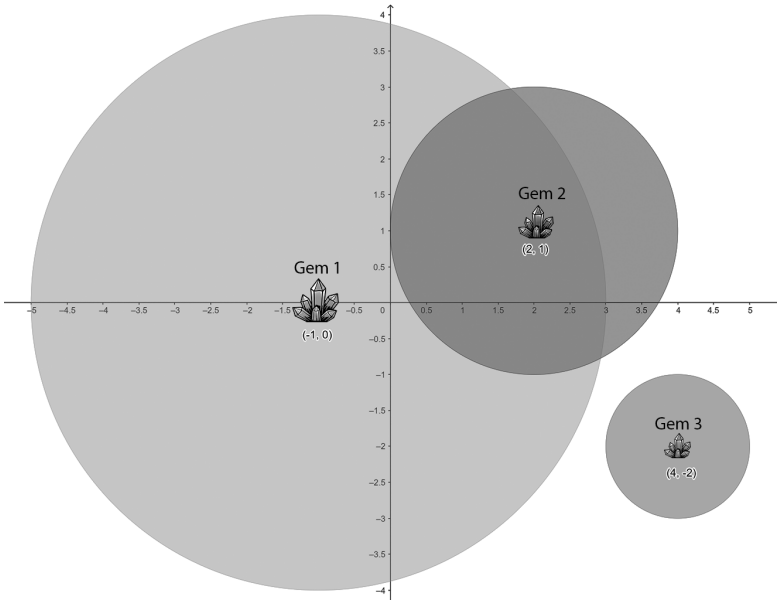


Figure 1: Illustration of the gems and their aura radii.

Sample Input	**Sample Output**
4 0 0 5 1 1 1 -1 -1 1 2 2 1 4 0 0 2 1 1 1 -1 -1 1 2 2 1 3 -1 0 4 2 1 2 4 -2 1 0	1 2 2

K: **Skyline**

Source file name: `skyline.c`, `skyline.cpp`, `skyline.java`, *or* `skyline.py`
Author: Rodrigo Cardoso

Lucy Diamond is a young architect pursuing a Master's degree in city development. As part of her research, she is studying how different city skylines compare. To advance with her analysis, Lucy has devised a method to estimate how the heights of buildings in a city are organized. Now she needs your assistance in writing a computer program to perform the necessary calculations.

A skyline of N buildings can be represented by a sequence of N building heights, denoted as $h_1, h_2, \ldots, h_N$, where $N \geq 1$. For any two buildings at positions i and j in the skyline, with $1 \leq i < j \leq N$, Lucy defines a height disorder whenever $h_i > h_j$. Let *HD* represent the total number of height disorders in the entire skyline and let *PHD* represent the potential number of height disorders for a skyline of that size. Lucy's measure for skyline organization is calculated as the ratio $\frac{HD}{PHD}$ when $N \geq 2$. In the case where $N = 1$, this measure is defined as 0, since no comparisons between building heights can be made.

As an example, consider a skyline of 4 buildings with heights $28, 30, -29, 28$ (note that a negative height indicates a depression in the landscape). In this case, there are 3 height disorders, corresponding to the pairs $(1, 3)$, $(2, 3)$, and $(2, 4)$. Since the potential number of height disorders for a skyline of 4 buildings is 6, Lucy's measure for this skyline is $\frac{3}{6}$.

Input

The input consists of several test cases. A case begins with a line containing a single integer positive number N, $0 < N < 100\,000$, the number of buildings in the skyline. Then, a line follows with N blank-separated integer numbers, representing the heights of the skyline buildings. Each height h is in the range $-20 < h < 300$. The end of input is signaled with a line containing 0.

Output

For each test case, print one answer line with the decimal number corresponding to the value $\frac{HD}{PHD}$ rounded to 3 decimal places.

The output must be written to standard output.

Sample Input	**Sample Output**
1 17 4 28 30 -29 28 0	0.000 0.500