Thirty-fourth Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	Filename
Colorful Neighborhood	colorful
Silly Strings	silly
Ahmad's Mod	mod
In a Jam	jam
Why Don't We Use Giant Fans to Deal With Hurricanes?	winds
Blockdoku	block
Lucky Charms	lucky
Typing Contest	typing
Waterfall Quest	waterfall
Bigloo Building	bigloo
Gamers on the Bus	bus
Sock Sort	socks

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving Bigloo Building:
Call your program file: bigloo.c, bigloo.cpp, bigloo.java, or bigloo.py
Call your Java class: bigloo

Colorful Neighborhood

Filename: `colorful`

In Rainbow World there are n colored houses. These houses are connected by m bi-directional paths, although it may not be possible to get from any house to any other house. People are very friendly, and they constantly visit their neighbors' houses just to say hello! Let's denote the color of a house as an uppercase letter, between 'A' to 'Z' (inclusive).

The people of Rainbow World also have a tradition! They draw some color out of a bucket (some people may draw more than once), and they then visit a neighbor whose house has that color.

The Problem:

The people of Rainbow World are people, so naturally they would like to fulfill the tradition with the least amount of effort - by visiting a neighbor that is closest to their home. Help them find their destinations!

The Input:

The first line of the input begins with a single positive integer, t, representing the number of neighborhoods. For each neighborhood, multiple lines follow.

- The first line contains three positive integers, n, m and k ($n \leq 10^5$; $m \leq 10^5$; $k \leq 10^5$), denoting the number of houses in the neighborhood, the number of paths between houses, and the total number of draws the neighbors made, respectively.
- Then, a string of n uppercase letters follows on its own line, the i^{th} character denoting the color of the i^{th} house.
- Next, m pairs of integers follow, u and v ($1 \leq u \leq n$; $1 \leq v \leq n$; $u \neq v$), denoting that there is a direct path between house u and house v.
- Lastly, k lines follow, representing the draws. Each consists of an integer, x ($1 \leq x \leq n$) followed by a single space and an uppercase letter c. This means that the person living in house x must visit a person whose house color is c. It is guaranteed that house x does not have color c and that color c is a valid color.

The Output:

For each neighborhood, output a single line "`Neighborhood #i:`" where `i` is the number of the neighborhood in the input (starting with 1), followed by k lines, each consisting of a single integer, the number of the house that should be visited. If there are multiple houses that minimize the distance, output the one with the smallest index. If a house of color c is not reachable from house x, output -1. Output a blank line after the answers to each neighborhood's queries.

Sample Input:

```
2
7 4 4
ABCDBFF
1 2
2 3
6 5
4 5
4 F
7 B
6 B
1 B
4 5 8
ABAC
1 2
2 3
3 4
4 1
3 1
1 B
1 C
2 A
2 C
3 B
3 C
4 A
4 B
```

Sample Output:

```
Neighborhood #1:
6
-1
5
2

Neighborhood #2:
2
4
1
4
2
4
1
2
```

Silly Strings

Filename: `silly`

Farmer John recently inherited a large plot of land dead in the center of Iowa. While surveying the land, he stumbled across a number of large strings consisting only of lowercase English letters left by the local hooligans. Farmer John detests many of these strings and wishes to make them more welcoming. He considers a string to be welcoming if he can remove a single character and the resulting string can be split into two portions that are anagrams of one another. Two strings are anagrams of each other if their characters can be rearranged such that the strings become equal. Farmer John would like to know how many options he has for making a given string more welcoming.

For example, the string `abaddab` may be split after removing the first or second `a` giving, respectively:

1. `bad dab`
2. `abd dab`

Within each pair of strings, the two strings are anagrams of each other. Furthermore, removing any other character instead (other than the first or second `a`) will not make the string `abaddab` welcoming.

The Problem:

For a given string, *s*, help Farmer John find out how many different ways he can remove a single character from *s* such that the remaining string is welcoming. Two ways are considered different if the characters he removes in each of them are at different positions.

The Input:

The first line will contain a single integer, *t*, representing the number of strings to check. The input for each string follows on two lines. The first line for each string contains a single integer, n ($2 \leq n \leq 200{,}000$), representing the length of the string that Farmer John is currently considering. The second line of each string contains the string, *s*, of length n.

The Output:

For each string, output a single line with an integer representing the number of locations such that removing the character at location i ($1 \leq i \leq n$) in *s* splits *s* into two strings, *l* and *r*, where *l* and *r* are anagrams of each other.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
7
abaddab
10
impossible
```

Sample Output:

```
2
0
```

Ahmad's Mod

Filename: `mod`

It's Ahmad's birthday today! Ahmad's rather strange aunt, Odd Maude, gave Ahmad an array of n integers (the same gift he gets every year). However, many of the numbers in the array are different! Ahmad wants to mod[ify] the array by taking every element and applying the modulo operator (mod) with a special number m (called Ahmad's Mod). Ahmad wants to choose m such that every element is equal under modulo m. He also wants m to be as big as possible.

Note that the modulo operator can be applied to two numbers, a and b, and the result is a single number, the remainder of the integer division a / b (for example, the remainder of 5 / 3 is 2). Thus, for two numbers, a_1 and a_2, to be "equal under modulo m," it would be required that $a_1 \bmod m = a_2 \bmod m$.

The Problem:

Given the array Odd Maude will give to Ahmad, pick the mod, m, such that m is as large as possible, and every element in Ahmad's array is the same when the mod m is applied to them.

The Input:

The first line will contain a single positive integer, t, representing the number of arrays of which to compute the mod m. Each array will consist of two lines. The first line for each array will contain a single integer, n ($2 \leq n \leq 10^5$), representing the length of the array that Maude has given Ahmad. The second line will contain n space-separated integers, i ($1 \leq i \leq 10^9$), representing the elements in Ahmad's array.

The Output:

For each array, output Ahmad's mod, m, the largest mod such that every number in Ahmad's array modulo m is the same. If m is unbounded, instead output "`Thank you, Maude!`" as Ahmad would be even more grateful than usual for such a beautiful array.

Sample Input:

```
2
3
1337 404 4447
2
20 20
```

Sample Output:

```
311
Thank you, Maude!
```

In a Jam

Filename: `jam`

A very hungry caterpillar has a hard time leafing food alone. Just yesterday he got himself into a real pickle (thankfully, he managed to chew his way out of that one). Today, he is in a jam and he needs your help finding his way out.

As it turns out, n drops of jam were spilled on the picnic table that the very hungry caterpillar is on. The i^{th} drop of jam has spread out and now covers a circle of radius r_i with center (x_i, y_i). Drops may overlap each other but no more than two outlines of drops will intersect at the same point. The caterpillar is located at the origin, (0,0). What is the shortest distance he needs to travel to reach a part of the table that isn't covered in jam?

The Problem:

Given the locations and sizes of all the jam droplets on the table, find the shortest distance the very hungry caterpillar needs to travel to reach a part of the table that isn't sticky and covered in jam.

The Input:

The input begins with a line containing a single integer, t, representing the number of tables to consider. Each table is defined across multiple lines. The first line for each table will contain a single integer, n ($1 \le n \le 100$), representing the number of jam droplets. On the next n lines, each contains three integers, x_i, y_i and r_i ($-1{,}000 \le x_i \le 1{,}000$; $-1{,}000 \le y_i \le 1{,}000$; $1 \le r_i \le 1{,}000$), representing the center, (x_i, y_i), and radius, r_i, of the circle of jam, respectively. The caterpillar is guaranteed to be within some jam (and will not be on an edge of some jam).

The Output:

For each table, output a single line containing a real number representing the minimum distance that the very hungry caterpillar needs to travel. The values should be output with three decimal places and rounded (0.0024 rounds to 0.002; 0.0025 rounds to 0.003).

Sample Input:

```
2
2
-1 0 2
10 1 9
3
0 1 2
-1 0 2
400 1 400
```

Sample Output:

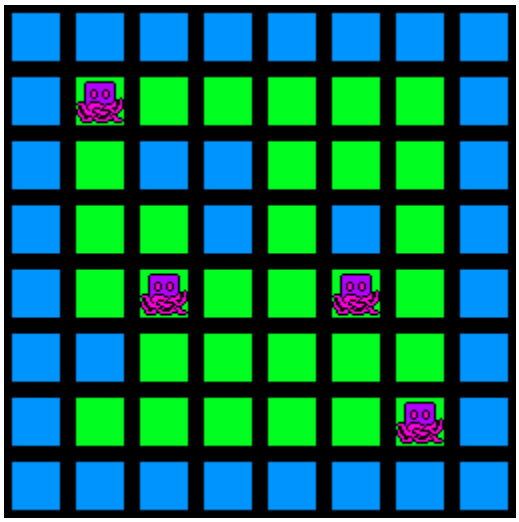
```
1.000
1.727
```

Why Don't We Use Giant Fans to Deal With Hurricanes?

Filename: `winds`

The Earth is under attack by Aliens! Anybody who has watched the movie "Signs" knows that Aliens are weak to water. In a last ditch effort to save humanity, the world military has cornered the Aliens on an island and set up giant fans on big aircraft carriers surrounding it.

The island can be represented as a square *n* x *n* grid. There are three types of cells. Cells with no Aliens on them are denoted by 'W' (water) or 'L' (land). An Alien's position is represented by the letter 'A'. Since it is an island, the borders of the grid are guaranteed to be water cells.



In this scenario, it is optimal to push the aliens upwards once and then to the right once.

The military arranges their four fans facing north, south, east, and west, and can only activate one at a time. When a fan activates, it pushes all Aliens one cell in that direction. If an Alien is pushed into water, it instantly drowns. Aliens do not move unless a fan is activated. They just stand there. Menacingly.

While the world is facing an existential threat, the military still wishes to conserve energy in order to prevent another existential threat in the form of climate change. For this reason, you have been hired to find the minimum number of times a fan needs to be activated to kill all Aliens.

The Problem:

Determine the minimum number of times a fan must be activated in order to kill all Aliens.

The Input:

The first line of the input will begin with a single positive integer, t, representing the number of scenarios. For each scenario, multiple lines will follow. The first line contains a single integer, n ($3 \leq n \leq 10$), representing the dimensions of the island. Then, on the following lines there are n rows each consisting of n characters, as described in above, describing the state of the island. It is guaranteed that there is at least one alien.

The Output:

For each scenario, output a single line with an integer representing the minimum number of fan activations.

Sample Input:

```
3
8
WWWWWWWW
WALLLLLW
WLWWLLLW
WLLWLWLW
WLALLALW
WWLLLLLW
WLLLLLAW
WWWWWWWW
4
WWWW
WLAW
WAWW
WWWW
3
WWW
WAW
WWW
```

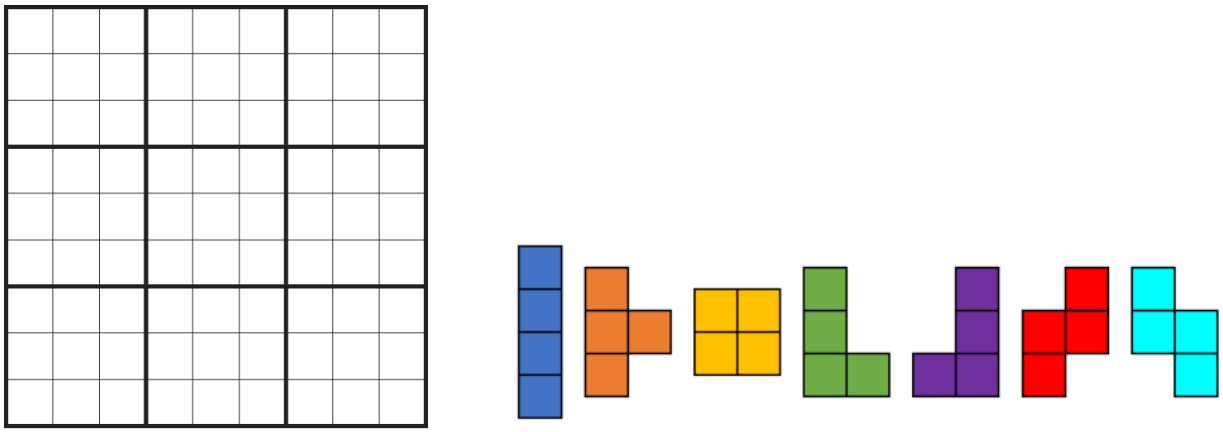
Sample Output:

```
2
1
1
```

Blockdoku

Filename: `block`

Blockdoku is a game that is similar to Sudoku and Tetris. It is played on a 9x9 grid that is empty initially. To start the game, the player is offered 3 random tetrominoes, which can be used in any order. The player may place them anywhere on the grid, one at a time, as long as they fit entirely within the grid and are not placed on top of any other pieces. The tetrominoes may not be rotated. Once they place all 3 tetrominoes on the board, they are given 3 more to use. If at any time a row, column, or one of the nine 3x3 sub-grids is completely filled, all 9 squares in that row, column, or sub-grid are erased from the board (some of the tetrominoes may be broken apart at this point). Points are awarded for each piece placed and every row, column, or sub-grid filled. The game ends when there are no possible moves remaining.



Left: the 9x9 grid with the nine 3x3 sub-grids outlined. Right: the 7 possible tetrominoes, which may be given in any orientation

The Problem:

For this problem, you will be given several board states and 3 tetrominoes to use. You must determine if it's possible to place some combination of the 3 pieces on the board such that any row, column, or sub-grid becomes filled. You may use 1, 2, or all 3 of the given pieces.

The Input:

The first line will contain an integer, p, denoting the number of puzzles. Each puzzle will consist of several lines of input. The first 9 lines will be the current state of the grid, represented using '`.`' and '`#`'. Each row will be exactly 9 characters. '`.`' indicates a free space and '`#`' indicates part of a tetromino. It is guaranteed that no row, column, or 3x3 sub-grid will already be filled. Following this will be the 3 tetrominoes (all from the set of 7 possible tetrominoes) that may be used. For each tetromino, there will first be a line containing 2 integers, r and c ($1 \leq r \leq 4$; $1 \leq c \leq 4$), representing the number of rows and columns, respectively. The following r lines will each contain c characters, of the same characters as the board. This will represent the tetromino.

The Output:

For each puzzle, output a single line with "`Yes`" if one or more rows, columns, or sub-grids can be filled using any combination of the 3 tetrominoes, or "`No`" otherwise.

Sample Input:

```
2
.....##.#
.........
........#
........#
........#
........#
......#.#
......#.#
......#..
3 2
#.
##
.#
3 2
.#
.#
##
2 3
###
.#.
#.#.#..##
#.#..####
##.##...#
..##.####
######.#.
#.##.#..#
###.#.###
.#.###...
#.##.####
1 4
####
1 4
####
1 4
####
```

Sample Output:

```
Yes
No
```

Lucky Charms

Filename: `lucky`

Every St. Patrick's Day, Daniel and his family go to the amazing clover harvesting festival in Clovertown! They spend the entire afternoon there collecting clovers for their lucky charms. A single lucky charm is made of a single four-leaf clover. If a clover does not have four leaves, it's not lucky!

A measure of how lucky a member of the family will be in the following year is determined by how many lucky charms they can make using the four-leaf clovers they've found. Daniel is feeling pretty lucky this year and thinks he has collected a record-breaking number of four-leaf clovers, but his collection of clovers is a mess! As a solution to this annual problem, write a program to help Daniel organize his pile and figure out how many lucky charms he can make.

The Problem:

Given the number of leaves on each clover Daniel has collected, determine how many lucky charms he can make.

The Input:

The input will begin with a line containing a single positive integer, t, representing the number of years for which Daniel needs help. The following t pairs of lines will begin with an integer, n ($1 \leq n \leq 10^5$), representing the number of clovers Daniel collected for the year, followed by a single line containing n space-separated integers, a_i ($1 \leq a_i \leq 4$), indicating the number of leaves on the i-th clover.

The Output:

For each year in the input, output a single line with an integer, representing the number of lucky charms Daniel can make.

Sample Input:

```
2
8
1 1 3 4 2 1 3 4
10
4 4 4 4 3 2 1 4 4 4
```

Sample Output:

```
2
7
```

Typing Contest

Filename: `typing`

Sharon always types capital letters by pressing the Caps Lock key, even if capitalizing only a single letter. While he is smart enough to use Caps Lock optimally (he will not toggle it off just to have to toggle it back on immediately), he is often judged by his peers, as sometimes it is better to use the Shift key while typing in order to minimize key presses.

For those of you who may not know, pressing on the Caps Lock key toggles capitalization for alphabetic keys. If Caps Lock is toggled on, pressing an alphabetic key on the keyboard will produce its capitalized version. When toggled off, it will produce the lowercase version.

Furthermore, holding the Shift key requires one additional key press and allows you to write letters of the opposite case (for alphabetic keys) while it continues to be held. If Caps Lock is on while the Shift key is held, it writes lowercase letters. Otherwise, it writes capital letters. Holding the Shift key while pressing non-alphabetic keys results in the same non-alphabetic character (i.e. the Shift key has no effect). Releasing the Shift key requires no key press and will write letters normally (this may be useful to know if you are planning on writing code).

Now, Sharon is stubborn and unwilling to change. While Sharon is known to be the fastest typer in the West, Internet Cowboys have stepped up to challenge him in a typing contest. Sharon will only use the Caps Lock key to switch between capitalization, while the challengers may use both Shift and Caps Lock. Both will start with Caps Lock toggled off.

The Problem:

Given the number of milliseconds it takes Sharon and his challenger to press any key (including shift and caps lock) on the keyboard, determine who will finish typing the text faster. The text has been determined so that there will always be a clear winner. Note that it is allowed to leave Caps Lock on at the end of the challenge.

The Input:

The first line of the input will begin with a single, positive integer, t, representing the number of challengers. For each challenger, two lines will follow. The first contains two integers, s and c ($1 \leq s \leq c \leq 100$), representing the number of milliseconds it takes Sharon and the challenger to press a key on the keyboard, respectively. The next line contains a single string (of at most 100 characters in length) consisting of only capital and lowercase letters as well as spaces. This string is the text used for the contest.

The Output:

For each challenger, output a single line with either "`Sharon has won`" if Sharon wins, or "`Sharon is gone`" if Sharon loses.

Sample Input:

```
3
2 5
Hello World
10 11
AbAcAbAdAbAcAbA
5 9
wanna BUY a BOX
```

Sample Output:

```
Sharon has won
Sharon is gone
Sharon has won
```

Waterfall Quest

Filename: `waterfall`

Emily is a brave adventurer who has heard rumors of treasure at the bottom of a waterfall. The waterfall has n horizontal rocky platforms protruding from the cliffside, which Emily can step on. As adventurous as Emily is, she loves reminiscing about her adventures so she takes a picture of the beautiful valley below from each platform she reaches on her way down!

Emily has a plan to reach the bottom. Equipped with a single-use rope, she will ride a raft down the river current from upstream to the top of the waterfall and land on any platform to start. Then, she can drop down to any platform (or to the bottom of the waterfall) that is at least partially below her current platform. She also has the option to use her rope in order to swing to any platform whose elevation is strictly lower than her current platform (regardless of distance). However, once she uses the rope, she cannot use it again.

The Problem:

Since Emily loves taking pictures, she would like to take them on as many platforms as possible. Find the maximum number of platforms Emily can visit on her way to the bottom.

The Input:

The first line of the input will begin with a single, positive integer, t, representing the number of waterfalls. For each waterfall, multiple lines follow. The first consists of a single integer, n ($1 \leq n \leq 10^5$), representing the number of platforms. Then, n lines follow, each consisting of three integers, x, y and s ($1 \leq x \leq 10^9$; $1 \leq y \leq 10^9$; $1 \leq s \leq 10^9$), representing the left endpoint of the platform, its elevation, and its size. Formally, each platform is a line segment between the points (x, y) and $(x+s, y)$. It is guaranteed that no two platforms intersect and no two platforms share endpoints.

The Output:

For each waterfall, output a single line with an integer representing the length of the longest path (in terms of the maximum number of platforms) Emily can take on her way down.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
13
5 15 4
2 12 6
9 12 2
1 6 4
7 6 3
13 9 4
4 5 2
9 5 2
14 4 2
3 4 2
12 3 3
15 2 2
14 1 1
2
1 2 4
8 2 9
```

Sample Output:

```
8
1
```

Bigloo Building

Filename: `bigloo`

Tyler, Alex R. Coleman, and a prenatal blob have embarked on a Spring Break trip to Iceland. While they made sure to add lots of cool stops to their itinerary, they completely forgot to make sleeping arrangements! Fortunately, at their first waypoint, they've stumbled upon an abandoned stash of pre-built igloo layers. Each igloo layer consists of a ring of snowy blocks, frozen solid.

In a predictable stroke of genius, Alex suggests that because they can't thaw the igloo layers in time to fashion a custom igloo, they can instead try to stack the existing igloo layers to fabricate a multi-layer igloo stack. Tyler has measured the diameter of each ring, and determined that one ring may be safely placed atop another if its diameter is *exactly* one unit smaller than the layer below it. Any ring may be used as the bottommost layer of the igloo stack.

The prenatal blob loves to juggle, so the igloo stack must be a certain number of layers tall. Dusk is falling, the closest hot spring is hundreds of kilometers away, and a blizzard is about to blow in. They need to figure something out fast. Can you help?

The Problem:

Given the diameters of several pre-built igloo layers, determine whether an igloo with the required number of layers can be built and, if so, find the maximum diameter of the base.

The Input:

The first line of the input will contain a positive integer, t, representing the number of scenarios. The first line of each scenario will contain two integers, n and h ($1 \leq h \leq n \leq 1{,}000$), representing the number of igloo layers available, and the required number of layers in an igloo stack, respectively. The second line of each scenario will contain n space-separated positive integers, representing the diameters of each igloo ring. The diameter of each ring will not exceed 10^7.

The Output:

Output a single integer representing the largest possible diameter of the base layer of the stack of igloo rings satisfying the required number of layers. However, if it is impossible to build any such igloo, instead output "`We're gonna freeze out here!`"

Sample Input:

```
2
10 3
1 4 5 6 10 2 9 11 20 13
8 6
1 2 3 4 5 7 8 9
```

Sample Output:

```
11
We're gonna freeze out here!
```

Gamers on the Bus

Filename: `bus`

The Programming Team is going to a competition in Miami, Florida. Lior and Sharon decided to drive to the contest separately from the team. Due to a lack of coordination, they have arrived earlier than expected and decided to play a game to pass the time.

The rules are simple: each person takes turns guessing where the bus is. Sharon thinks the playing field is pretty fair because he believes the only knowledge they both have is how much time has passed since the bus left Orlando. But Lior knows better; he packed a map of Florida for the trip and also knows all possible routes between Orlando and Miami! Lior is given another advantage, as he is well-acquainted with the driver of the bus and knows he will arrive in the shortest time possible. Unfortunately, to not raise suspicion that he may be cheating, Lior doesn't have time to calculate the exact location of the bus, so he needs the help of a fellow programmer to write a program for him.

The Problem:

Given a map of Florida represented as roads and intersections on an x-y plane, help write a program that provides the exact location of the bus given the time elapsed since the bus left Orlando or tells Lior to take his best guess if the answer is ambiguous.

The Input:

The input begins with a line with a single integer, t, representing the number of trips. Each trip is defined across multiple lines. The first line of each trip begins with two integers, n ($2 \le n \le 10^5$) and m ($1 \le m \le 10^5$), representing the number of intersections and roads, respectively. The following n lines contain two integers, x and y ($-1{,}000 \le x \le 1{,}000$; $-1{,}000 \le y \le 1{,}000$), describing the location of the i-th intersection.

This is followed by m lines which each contains two integers, a and b ($0 \le a < n$; $0 \le b < n$), describing a road between intersections a and b. Orlando is intersection 0, and Miami is intersection n-1. All roads are bidirectional and straight lines (since there are no hills in Florida). It is guaranteed that no roads cross unless they are connected by an intersection.

On the next line of input, an integer, q ($1 \le q \le 10^5$), is given, denoting the number of turns Lior has to make a guess. The following q lines contain a single, non-negative integer, e ($e \le 10^{18}$), which dictates the time elapsed since the bus left Orlando. It's safe to assume that the bus moves at a pace of one unit per minute (the driver is very safe) and the bus has not arrived in Miami, yet.

The Output:

For each query on a line by itself, output two numbers, b_x and b_y, separated by a single space, representing the coordinates of the location that Lior should guess to give him the best chance of winning the round given time t. The values should be output rounded to two decimal places (0.024 rounds to 0.02; 0.025 rounds to 0.03). If it is not possible to provide an exact guess, output "`Best of Luck`" instead.

Sample Input:

```
1
6 7
0 0
4 0
2 2
0 2
2 0
4 4
0 3
4 0
1 5
2 4
5 2
2 3
1 4
2
2
5
```

Sample Output:

```
Best of Luck
2.71 2.71
```

Sock Sort

Filename: `socks`

Today is a wonderful day. The sun is shining, the birds are chirping, and you have lots of interesting programming problems ahead of you. There's just one thing bothering you: your socks aren't matched!

Currently, you have $2n$ socks in an unmatched pile. n times, you will do the following:

- Take the top sock out of the pile of unmatched socks.
- Of the socks left in that pile, find the highest one that matches the sock you just took out and remove it from the pile.
- Put the matching pair of socks on the top of your pile of matched socks.

What will your matched pile look like once you're done? You better hurry so you have time to get dressed for the programming contest!

The Problem:

Given your initial pile of unmatched socks, determine what the final pile of matched socks will look like.

The Input:

The first line will contain a single, positive integer, s, representing the number of scenarios. For each scenario, the first line will contain a single integer, n ($1 \leq n \leq 10^5$), representing the number of pairs of socks you own. The next line will contain $2n$ positive integers describing the style of each sock from top to bottom. These integers will be no greater than n, and there will be an even number of socks of each style.

The Output:

For each pile, output a single line containing $2n$ integers representing the matched pile of socks once you are done, in order of top to bottom.

Sample Input:

```
2
5
1 3 1 5 1 1 5 3 4 4
4
1 1 1 2 1 1 1 2
```

Sample Output:

```
4 4 1 1 5 5 3 3 1 1
1 1 2 2 1 1 1 1
```