

# Practice Problems

*Computer Programming and Utilization*

PARAM RATHOUR

CS101

# Practice Problems

## Computer Programming and Utilization

Param Rathour  
<https://paramrathour.github.io/CS101>

Last update: 2023-03-08 04:59:11+05:30

### Disclaimer

These are **optional** problems. As these problems are pretty involving, my advice to you would be to first solve exercises given in slides, lab optional questions and get comfortable with the course content.

I have created these problems such that you will learn something new from each problem. Each section builds on the next; so, try to solve the problems only using the **topics mentioned in that section and previous sections**. They will suffice to solve these problems. Don't forget to look at the **starter code** (it will be in blue) for each problem which takes care of input and output behaviours (and sometimes hints). I have also prepared **model solutions** for each problem, they are available on request.

### Acknowledgements

Many thanks to [Numberphile](#), [3Blue1Brown](#), [Mathologer](#), [PBS Infinite Series](#), [Veritasium](#) and countless other YouTube channels for developing my love for mathematics and their *Fun Videos* further inspiring me to create these problems. Also thank you [Wikipedia](#) and [The On-Line Encyclopedia of Integer Sequences](#) for freely providing their vast resources and detailed information about concepts which helped me frame these problems. Many numbers, phrases, equations and graphics are directly taken from it and modified as per my wish. I would also like to thank [Project Euler](#), [CSES](#), [Codeforces](#) and many other online programming practice communities which motivated me to further pursue programming and create problems. I faced lots of T<sub>E</sub>Xnical issues while setting up this document and I thank [T<sub>E</sub>X-<sub>La</sub>TeX Stack Exchange](#) community for their support and many thanks to [L<sup>A</sup>T<sub>E</sub>X Draw](#) for their stylish cover page. Thanks to the CS101 professors, my fellow TAs, tutees, and others for their valuable suggestions on improving these problems. And, lastly thanks to you, reader; These problems are the result of my hard work over the years. I hope they help you in some way or the other and you enjoy solving them :).

### Simplecpp Graphics

Also we will be using [Simplecpp](#) for initial problem sets (till 8). Why? because [Introductory Programming: Let Us Cut through the Clutter!](#) The course book is [An Introduction to Programming through C++](#) by Abhiram G. Ranade. Apart from C++, Simplecpp graphics are an interesting approach to introductory programming. Check out [Turtle Graphics – Wikipedia](#) and [Simplecpp Gallery](#) for some fascinating examples. Graphics problems in this problem set are – [Star Spiral](#), [Peace](#), [Modular Times Table](#), [Regular Star Polygon](#), [Hilbert Curve](#), [Thue-Morse Sequence](#), [Recaman's Sequence](#), [Farey Sequence](#), [Dragon Curve](#), [Sierpiński Arrowhead Curve](#), [Sierpiński Triangle](#) and [Barnsley's Fern](#).

Here are additional chapters of the book on Simplecpp graphics demonstrating its power.  
(It is just a list, you are not expected to understand/study things, CS101 is for a reason :P)

**Chapter 1** Turtle graphics

**Chapter 5** Coordinate based graphics, shapes besides turtles

**Chapter 15.2.3** Polygons

**Chapter 19** Gravitational simulation

**Chapter 20** Events, Frames, Snake game

**Chapter 24.2** Layout of math formulae

**Chapter 26** Composite class

**Chapter 28** Airport simulation

## How to write a program? (5Cs)

- Carefully go through the problem statement
- Check your understanding of problem using solved examples and practice testcases
- Compose the programming approach on paper
- Consolidate your approach by verifying its correctness on testcases by doing dry runs
- Code it up (finally!)

## Good Programming Practices

- Write documentation clearly explaining
  - what the program does,
  - how to use it,
  - what quantities it takes as input, and
  - what quantities it returns as output.
- Use appropriate variable/function names.
- Extensive internal comments explaining how the program works.
- Complete error handling with informative error messages.  
For example, if  $a = b = 0$ , then the `gcd(a, b)` routine should return the error message “`gcd(0,0)` is undefined” instead of going into an infinite loop or returning a “division by zero” error.

## Tips

- Some data types that you should keep in mind are:
  - `bool`
  - `char`
  - `short int`, `int`, `long int`, `long long int` and their unsigned counterparts
  - `float`, `double`, `long double`
- Choose appropriate variable data types according to constraints. Example, if a variable is always an integer then it should be assigned an `int` data type.
- Whenever possible prefer integer data types over floating point data types which aren't accurate due to floating point errors. Some problems that look like they will need floating point numbers but are solvable using integers are **Triangle Types**, **Friendly Pair** and **Newton Interpolation**.
- Use **type conversion** to your advantage to
  - make your program unambiguous.
  - compute expressions containing variables of different data types.
- Find more tips at <https://paramrathour.github.io/CS101/tips>

## Get comfortable with Dry Runs

The most important step in debugging

- Select a testcase
- Manually go through the code to trace the value of variables
- Check if the values of variables matches with their expected values
  - If they do not match for any variable at any time then your program is incorrect, consider debugging/rewriting it
  - If they match for all variables at all times, Hurray your program is correct for the current testcases!
- Now repeat the procedure for a different testcase :)

# Contents

<b>1</b>	<b>Prodigal Patterns</b>	<b>6</b>
1.1	Star Spiral	6
1.2	Peace	7
1.3	Butterfly	8
1.4	Alphabetical Floyd's Triangle	9
1.5	Bernoulli's Triangle	10
1.6	Modular Times Table	11
<b>2</b>	<b>Expression Obsession</b>	<b>12</b>
2.1	Harmonic Number	12
2.2	Wallis Product	13
2.3	Tetration	14
2.4	Ramanujan's Nested Radical	15
2.5	Simple Continued Fractions	16
2.6	Ramanujan's $\sqrt{\frac{\pi e}{2}}$ Formula	17
2.7	Viète's $\pi$ Formula	18
2.8	Hölder Mean	19
2.9	Shoelace Formula	20
2.10	Simpson's Rule	21
<b>3</b>	<b>Traditional Conditionals</b>	<b>22</b>
3.1	Triangle Types	22
3.1.1	By Side	22
3.1.2	By Angle	22
3.2	Clock Angle	23
3.3	Fleur Delacour	24
3.4	Doomsday Algorithm	25
<b>4</b>	<b>Iteration Domination</b>	<b>26</b>
4.1	Pisano Period	26
4.2	Palindromic Number	27
4.3	Kempner Series	28
4.4	Base -2	29
4.5	Base Conversion	30
<b>5</b>	<b>Function Admiration</b>	<b>31</b>
5.1	Collatz Conjecture	31
5.2	Friendly Pair	32
5.3	Gauss Circle Problem	33
5.4	Euler's Totient Function	34
5.5	Regular Star Polygon	35
<b>6</b>	<b>Recursion Salvation</b>	<b>36</b>
6.1	Ackermann Function	36
6.2	Horner's Method	37
6.3	Modular Exponentiation	38
6.4	Partitions	39
6.5	Hereditary Representation	40
<b>7</b>	<b>Paths Paranoia (More Recursion?)</b>	<b>41</b>
7.1	Staircase Walk	41
7.2	Dyck Path	42
7.3	Delannoy Number	43
7.4	Schröder Number	44
7.5	Motzkin Number	45
7.6	Hilbert Curve	46

<b>8</b>	<b>Sequence Eminence (Intro to Arrays)</b>	<b>47</b>
8.1	Josephus Problem . . . . .	47
8.2	Van Eck's Sequence . . . . .	48
8.3	Look-And-Say Sequence . . . . .	49
8.4	Thue-Morse Sequence . . . . .	50
8.5	Recaman's Sequence . . . . .	51
8.6	Farey Sequence . . . . .	52
<b>9</b>	<b>Array Leeway (2-D Arrays)</b>	<b>53</b>
9.1	Case Converter . . . . .	53
9.2	Spiral Grid . . . . .	54
9.3	Minesweeper . . . . .	55
9.4	Gray Code . . . . .	56
<b>10</b>	<b>Array Powerplay (More Arrays or Recursion?)</b>	<b>57</b>
10.1	Determinant of a Matrix . . . . .	57
10.2	Tower of Hanoi . . . . .	58
10.3	Quicksort . . . . .	59
<b>11</b>	<b>Programming Expositions</b>	<b>60</b>
11.1	Newton Interpolation . . . . .	60
11.2	ISBN . . . . .	61
11.3	Vigenère Cipher . . . . .	62
11.4	Linear Feedback Shift Register . . . . .	64
<b>12</b>	<b>Fractal Fun</b>	<b>65</b>
12.1	L-Systems . . . . .	65
12.1.1	Dragon Curve . . . . .	65
12.1.2	Sierpiński Arrowhead Curve . . . . .	65
12.2	Chaos Game (Iterated Function Systems) . . . . .	66
12.2.1	Sierpiński Triangle . . . . .	66
12.2.2	Barnsley's Fern . . . . .	66

## §1. Prodigious Patterns

**Topics.** `turtleSim` (*turtle simulator*) and its features `forward`, `right`, `left`, `penUp`, `penDown`, `repeat statement`, *variables and their data types* (`int`, `char`), *typecasting*.

### 1.1. Star Spiral

**Problem Statement:**

Draw the following Star Spiral.

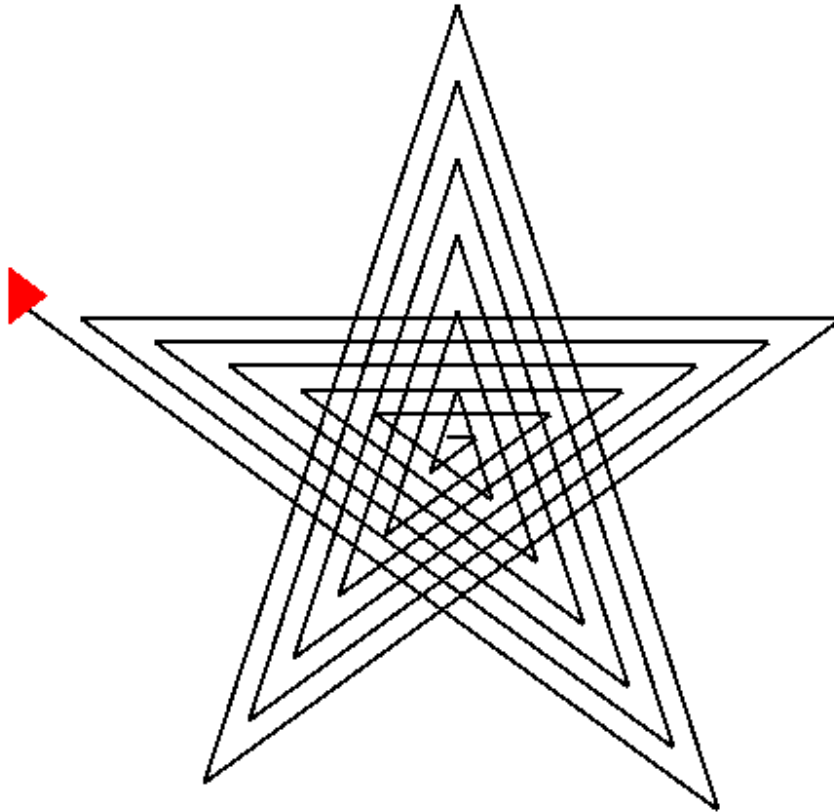


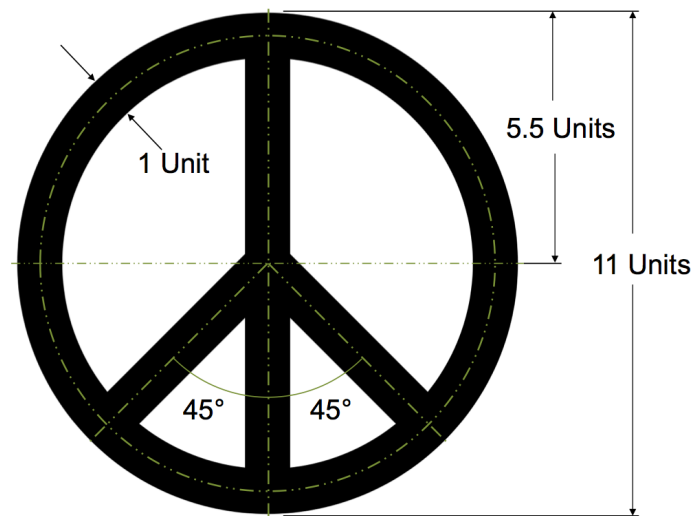
Figure 1: A Star Spiral of 30 sides

**Fun Video.** [Freaky Dot Patterns – Numberphile](#)

## 1.2. Peace

### Problem Statement:

Draw the outline of the Proportional Peace Sign according to measurements as shown in 2a.



(a) [Measurements](#) by Jerry S. Sadin, based on ([image](#) by [SchuminWeb](#))



(b) Output generated using Simplecpp

Figure 2: Peace Sign

The output image will look like 2b.

**Fun Video.** [Carl Sagan's Pale Blue Dot – carlsagandotcom](#)  
[Cosmos: Possible Worlds \(Carl Sagan's Monologue\) – Evil Dead](#)

### 1.3. Butterfly

**Problem Statement:**

Print the Butterfly pattern for a general  $n$ . See Starter code (below) for more details.

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1\ n_2\ \dots\ n_t$	(number of test cases, an integer) ( $t$ space seperated integers for each testcase)
<b>Output Format</b> Butterfly pattern	(each test case on a newline)
<b>Constraints</b> $1 \leq n_i \leq 10$	
<b>Sample Input</b> 5 1 2 3 4 5	
<b>Sample Output</b> <pre>*      * * * * *      *  *          * * *      * * * * * * * * * * * * * *      * * *          *  *              * * *          * * * * *      *      * * * * *          * * *              *  *                  * * *              * * * * *          * * * * * * *      *      * * * * *          * * *                  *  *                      * * *                  * * * * *              * * * * * * *          * * * * * * * *      *      * * * * *          * * *                      * *                          *</pre>	

**Fun Video.** [Chaos: The Science of the Butterfly Effect – Veritasium](#)



### 1.4. Alphabetical Floyd's Triangle

The alphabets are filled in alphabetical order ('A' to 'Z') and a newline is started after printing  $n$  alphabets on the  $n^{\text{th}}$  line. After 'Z', the alphabets "wrap around" to 'A'.

**Problem Statement:**

Print the left-aligned Alphabetical Floyd's Triangle for all given  $n$ . See Starter code (below) for more details.

<b>Starter Code</b>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1\ n_2\ \dots\ n_t$	( $t$ space seperated integers for each testcase)
<b>Output Format</b>	
Alphabetical Floyd's Triangle	(left-aligned, each test case on a newline)
<b>Constraints</b>	
$1 \leq n_i \leq 20$	
<b>Sample Input</b>	
5	
1 2 3 5 17	
<b>Sample Output</b>	
A	
A	
B C	
A	
B C	
D E F	
A	
B C	
D E F	
G H I J	
K L M N O	
A	
B C	
D E F	
G H I J	
K L M N O	
P Q R S T U	
V W X Y Z A B	
C D E F G H I J	
K L M N O P Q R S	
T U V W X Y Z A B C	
D E F G H I J K L M N	
O P Q R S T U V W X Y Z	
A B C D E F G H I J K L M	
N O P Q R S T U V W X Y Z A	
B C D E F G H I J K L M N O P	
Q R S T U V W X Y Z A B C D E F	
G H I J K L M N O P Q R S T U V W	

1.5. Bernoulli’s Triangle

You might have heard about [Pascal’s Triangle](#). The  $k^{\text{th}}$  element of row  $n$  of Bernoulli’s Triangle is obtained by as shown in 3 summing all elements of the row  $n$  (row 0 is the first row) until the  $k^{\text{th}}$  element (partial sums).

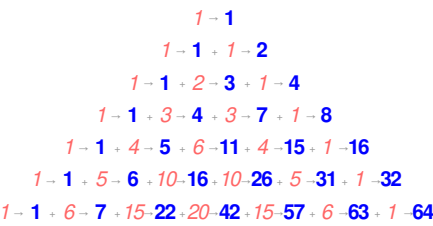


Figure 3: [Bernoulli’s triangle](#) from [Pascal’s triangle](#) (Image by [Cmglee](#) licensed under [CC BY-SA 4.0](#))

Problem Statement:

Print the left-aligned Bernoulli’s Triangle for all given  $n$ . See Starter code (below) for more details.

**Starter Code**

**Input Format**  
 $t$  (number of test cases, an integer)  
 $n_1\ n_2\ \dots\ n_t$  ( $t$  space seperated integers for each testcase)

**Output Format**  
Bernoulli’s Triangle (left-aligned, each test case on a newline)

**Constraints**  
 $0 \leq n_i \leq 20$

**Sample Input**  
4  
0 1 2 10

**Sample Output**  
1  
  
1  
1 2  
  
1  
1 2  
1 3 4  
  
1  
1 2  
1 3 4  
1 4 7 8  
1 5 11 15 16  
1 6 16 26 31 32  
1 7 22 42 57 63 64  
1 8 29 64 99 120 127 128  
1 9 37 93 163 219 247 255 256  
1 10 46 130 256 382 466 502 511 512  
1 11 56 176 386 638 848 968 1013 1023 1024

Fun Video. [Pascal’s Triangle – Numberphile](#)  
[What You Don’t Know About Pascal’s Triangle – Tipping Point Math](#)

1.6. Modular Times Table

Procedure to construct the Modular Times Table:

- Draw a circle which fits the entire “drawing canvas”.
- Imagine you have  $n$  equally-spaced points on the circumference of this circle. Number them 0 to  $n - 1$  anti-clockwise with 0 being the leftmost point.
- For each  $i \in \{0, 1, 2, \dots, n - 1\}$  connect the points representing  $i$  with the point for  $(m \cdot i) \% n$  with a straight line.

An example is shown in 4. Don't draw the numbers. They are just to visualise the construction.

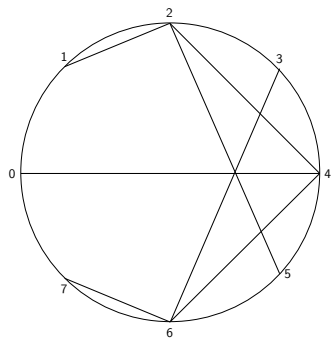


Figure 4: Times Table for  $(n, m) = (8, 2)$

**Problem Statement:**  
For a given  $(n, m)$  pair ( $n > m$ ), construct the Modular Times Table.

<b>Starter Code</b>	
<b>Input Format</b> $n \quad m$	(two numbers)
<b>Output Format</b> The constructed Modular Times Table	
<b>Constraints</b> $3 \leq n \leq 500$ $1 < m < n$	(an integer) (a double, first try to solve the problem for an integer $m$ )
<b>Sample Input</b> See 5	
<b>Sample Output</b> See 5	

The output Modular Times Tables

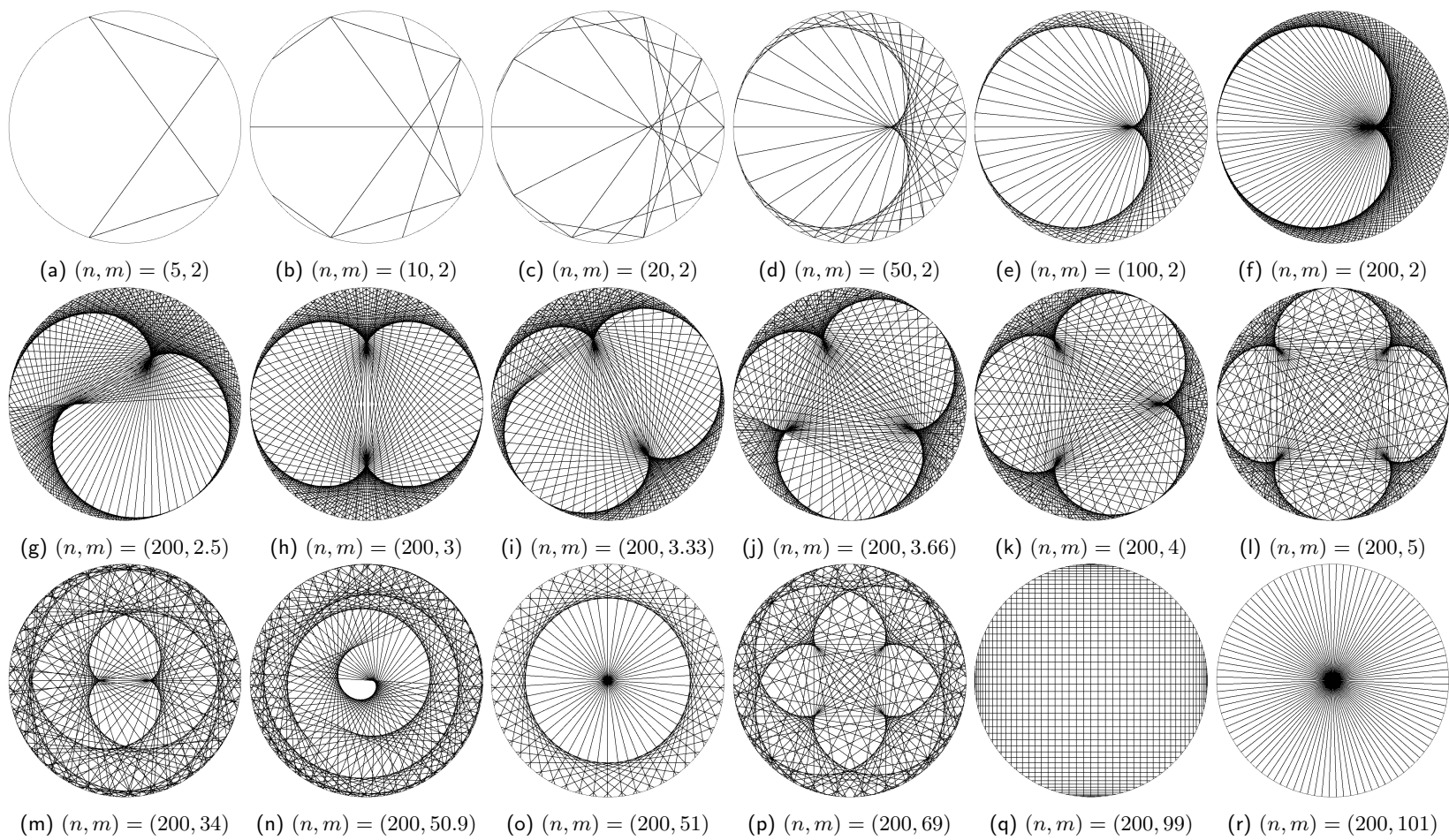


Figure 5: Modular Times Table

**Fun Video.** [Times Tables, Mandelbrot and the Heart of Mathematics](#)  
[Modular Times Tables](#)

## §2. Expression Obsession

**Topics.** repeat *statement*, *variables and their data types* (int, double), *mathematical functions* (min, max, sqrt, pow, log, sine...).

### 2.1. Harmonic Number

The  $n$ -th Harmonic Number ( $H_n$ ) is the sum of the reciprocals of the first  $n$  natural numbers.

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{i=1}^n \frac{1}{i} \quad (1)$$

**Fun Fact.** The Harmonic series diverges; i.e.,  $H_n \rightarrow \infty$  as  $n \rightarrow \infty$ .

**Problem Statement:**

Calculate  $H_n$  for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 \ n_2 \ \dots \ n_t$  ( $t$  space separated integers for each testcase)

##### Output Format

$H_{n_i}$  (each test case on a newline, accurate till 10 decimal places)

##### Constraints

$1 \leq n_i \leq 10^6$

##### Sample Input

11  
1 2 3 5 10 20 30 50 100 1000 1000000

##### Sample Output

1.0000000000  
1.5000000000  
1.8333333333  
2.2833333333  
2.9289682540  
3.5977396571  
3.9949871309  
4.4992053383  
5.1873775176  
7.4854708606  
14.3927267229

**Fun Video.** [The Harmonic Series – Tipping Point Math](#)

## 2.2. Wallis Product

$\pi/2$  is given by below infinite product formula. It is the ratio of product of even squares and odd squares

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots = \prod_{i=1}^{\infty} \left( \frac{2i}{2i-1} \cdot \frac{2i}{2i+1} \right) \quad (2)$$

Let's define  $\pi_n$  as  $n$ -th iteration of this infinite product as below

$$\frac{\pi_n}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots \frac{2n}{2n-1} \cdot \frac{2n}{2n+1} = \prod_{i=1}^n \left( \frac{2i}{2i-1} \cdot \frac{2i}{2i+1} \right)$$

### Problem Statement:

Calculate  $\pi_n$  for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 \ n_2 \ \dots \ n_t$  ( $t$  space separated integers for each testcase)

##### Output Format

$\pi_{n_i}$  (each test case on a newline, accurate till 10 decimal places)

##### Constraints

$1 \leq n_i \leq 10^6$

##### Sample Input

11  
1 2 3 5 10 20 30 50 100 1000 1000000

##### Sample Output

2.6666666667  
2.8444444444  
2.9257142857  
3.0021759546  
3.0677038066  
3.1035169615  
3.1159482859  
3.1260789002  
3.1337874906  
3.1408077460  
3.1415918682

**Fun Video.** [The Wallis product for pi, proved geometrically – 3Blue1Brown](#)  
[The World's Most Beautiful Formula For Pi – BriTheMathGuy](#)

### 2.3. Tetration

Problem 2.1 is about repeated additions whereas 2.2 is about repeated multiplication. Guess what's this problem about. Yes! It's repeated exponentiation. Tetration, the next [hyperoperation](#) after exponentiation defined as:

$${}^n a = \underbrace{a^{a^{\cdot^{\cdot^{\cdot^a}}}}}_n \text{ repeated exponentiation}$$

(3)

**Problem Statement:**

Calculate  ${}^n a$  for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

<b>Starter Code</b>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$a_1\ n_1\ a_2\ n_2\ \dots\ a_t\ n_t$	( $t$ space seperated pairs for each testcase)
<b>Output Format</b>	
${}^n a$	(each test case on a newline, accurate till 10 decimal places)
<b>Constraints</b>	
$0.05 \leq a_i \leq 3$	(a double)
$1 \leq n_i \leq 1000$	(an integer)
<b>Sample Input</b>	
10 1 1 1 2 2 1 2 2 2 3 3 2 3 3 1.41421356237 20 0.06598803584 1000 1.44466786101 1000	
<b>Sample Output</b>	
1.0000000000 1.0000000000 2.0000000000 4.0000000000 16.0000000000 27.0000000000 7625597484987.0000000000 1.9995856229 0.3968311347 2.7128728643	

**Fun Video.** [Tetration: The operation you were \(probably\) never taught – Taylor Series](#)  
[“Prove”  \$4 = 2\$  Using Infinite Exponents. Can You Spot The Mistake? – Mind Your Decisions](#)

## 2.4. Ramanujan's Nested Radical

$$r = \sqrt{1 + 2\sqrt{1 + 3\sqrt{1 + 4\sqrt{1 + \cdots}}}} = \lim_{n \rightarrow \infty} \sqrt{1 + 2\sqrt{1 + 3\sqrt{\cdots \sqrt{1 + n}}}} \quad (4)$$

Let's define  $r_n$  as  $n$ -th iteration of this infinite nested radical as below

$$r_n = \sqrt{1 + 2\sqrt{1 + 3\sqrt{\cdots \sqrt{1 + n}}}}$$

### Problem Statement:

Calculate  $r_n$  for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1 \ n_2 \ \dots \ n_t$	(number of test cases, an integer) ( $t$ space separated integers for each testcase)
<b>Output Format</b> $r_{n_i}$	(each test case on a newline, accurate till 10 decimal places)
<b>Constraints</b> $2 \leq n_i \leq 100$	
<b>Sample Input</b> 8 2 3 5 10 20 30 50 100	
<b>Sample Output</b> 1.7320508076 2.2360679775 2.7550532613 2.9899203606 2.9999878806 2.9999999868 3.0000000000 3.0000000000	

**Fun Video.** [Ramanujan: Knowing The Man Who Knew Infinity – singingbanana](#)  
[Ramanujan's infinite root and its crazy cousins – Mathologer](#)

**Fun Video.** *Infinite fractions and the most irrational number – Mathologer*



2.6. Ramanujan’s  $\sqrt{\frac{\pi e}{2}}$  Formula

This problem is a fusion of 2.5 and 2.1. It is recommended to solve them before proceeding to this problem.

$$\sqrt{\frac{\pi e}{2}} = \frac{1}{1 + \frac{1}{1 + \frac{2}{1 + \frac{3}{1 + \frac{4}{1 + \ddots}}}}} + \left\{ 1 + \frac{1}{1 \cdot 3} + \frac{1}{1 \cdot 3 \cdot 5} + \frac{1}{1 \cdot 3 \cdot 5 \cdot 7} + \frac{1}{1 \cdot 3 \cdot 5 \cdot 7 \cdot 9} + \cdots \right\}$$

(6)

Let’s define  $c_n$  as  $n$ -th convergent of this infinite continued fraction and sum as below

$$c_n = \sum_{i=0}^n \frac{a_i}{(2n+1)!!} \quad \text{where} \quad a_i = \begin{cases} 1 & i = 0 \\ i & i > 0 \end{cases} \Rightarrow \sqrt{\frac{\pi e}{2}} = \lim_{n \rightarrow \infty} c_n$$

**Note.**  $n!! \neq (n!)!$ ,  $n!!$  is *double factorial* of  $n$ .

Problem Statement:

Calculate  $c_n$  for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

Starter Code	
Input Format	
$t$	(number of test cases, an integer)
$n_1 \ n_2 \ \dots \ n_t$	( $t$ space seperated integers for each testcase)
Output Format	
$c_{n_i}$	(each test case on a newline, accurate till 10 decimal places)
Constraints	
$0 \leq n_i \leq 10^6$	
Sample Input	
12 0 1 2 3 5 10 20 30 50 100 1000 1000000	
Sample Output	
2.0000000000 1.8333333333 2.1500000000 2.0095238095 2.0422571580 2.0709281786 2.0667462769 2.0664199465 2.0663680635 2.0663656843 2.0663656771 2.0663656771	

Fun Video. [7 factorials you probably didn’t know – blackpenredpen](#)  
[The Man Who Knew Infinity – Tipping Point Math](#)

## 2.7. Viète's $\pi$ Formula

This problem is a fusion of 2.2 and 2.4. It is recommended to solve them before proceeding to this problem.

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots = \prod_{i=1}^{\infty} \frac{\overbrace{\sqrt{2+\sqrt{\cdots\sqrt{2+\sqrt{2+\sqrt{2+0}}}}}^{i \text{ 2's}}}}{2} \quad (7)$$

Let's define  $\pi_n$  as  $n$ -th iteration of this infinite nested radical as below

$$\frac{2}{\pi_n} = \prod_{i=1}^n \frac{\overbrace{\sqrt{2+\sqrt{\cdots\sqrt{2+\sqrt{2+\sqrt{2+0}}}}}^{i \text{ 2's}}}}{2}$$

### Problem Statement:

Calculate  $\pi_n$  for all test cases accurate till 15 decimal places. See Starter code (below) for more details.

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1 \ n_2 \ \dots \ n_t$	(number of test cases, an integer) ( $t$ space seperated integers for each testcase)
<b>Output Format</b> $\pi_{n_i}$	(each test case on a newline, accurate till 15 decimal places)
<b>Constraints</b> $1 \leq n_i \leq 50$	
<b>Sample Input</b> 8 1 2 3 5 10 20 30 50	
<b>Sample Output</b> 2.828427124746190 3.061467458920718 3.121445152258052 3.140331156954753 3.141591421511200 3.141592653588618 3.141592653589793 3.141592653589793	

Fun Video. [The Discovery That Transformed Pi – Veritasium](#)

## 2.8. Hölder Mean

Hölder mean is a generalized notion for aggregating sets of numbers.  
For any non-zero real number  $p$  and positive reals  $x_1, x_2, \dots, x_n$ , it is defined as

$$M_p(x_1, \dots, x_n) = \left( \frac{1}{n} \sum_{i=1}^n x_i^p \right)^{\frac{1}{p}} \quad (8)$$

Its special cases are

$$\begin{aligned} p = -\infty &\rightarrow M_{-\infty}(x_1, \dots, x_n) = \lim_{p \rightarrow -\infty} M_p(x_1, \dots, x_n) = \min\{x_1, \dots, x_n\} && \text{(minimum)} \\ p = -1 &\rightarrow M_{-1}(x_1, \dots, x_n) = \frac{n}{\frac{1}{x_1} + \dots + \frac{1}{x_n}} && \text{(harmonic mean)} \\ p = 0 &\rightarrow M_0(x_1, \dots, x_n) = \lim_{p \rightarrow 0} M_p(x_1, \dots, x_n) = \sqrt[n]{x_1 \cdots x_n} && \text{(geometric mean)} \\ p = 1 &\rightarrow M_1(x_1, \dots, x_n) = \frac{x_1 + \dots + x_n}{n} && \text{(arithmetic mean)} \quad (9) \\ p = 2 &\rightarrow M_2(x_1, \dots, x_n) = \sqrt{\frac{x_1^2 + \dots + x_n^2}{n}} && \text{(root mean square)} \\ p = 3 &\rightarrow M_3(x_1, \dots, x_n) = \sqrt[3]{\frac{x_1^3 + \dots + x_n^3}{n}} && \text{(cubic mean)} \\ p = +\infty &\rightarrow M_{+\infty}(x_1, \dots, x_n) = \lim_{p \rightarrow +\infty} M_p(x_1, \dots, x_n) = \max\{x_1, \dots, x_n\} && \text{(maximum)} \end{aligned}$$

### Problem Statement:

Calculate  $M_p(x_1, \dots, x_n)$  for all special cases ( $p = -\infty, -1, 0, 1, 2, 3, \infty$ ) and accurate till 5 decimal places.

<b>Starter Code</b>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_i \quad x_1 \quad x_2 \quad \dots \quad x_{n_i-1} \quad x_{n_i}$	( $n_i + 1$ space separated numbers for each testcase)
<b>Output Format</b>	
$M_p(x_1, \dots, x_n)$ for $p = \{-\infty, -1, 0, 1, 2, 3, \infty\}$ (each test case on a newline, accurate till 5 decimal places))	
<b>Constraints</b>	
$1 \leq n_i \leq 50$	(an integer)
$0 < x_i \leq 100$	(a double)
Also assume that the calculations are always within the range of double	
<b>Sample Input</b>	
4	
2 1 1	
5 1 2 3 4 5	
13 1 3 6 10 15 21 28 36 45 55 66 78 91	
33 1 3 6 2 7 13 20 12 21 11 22 10 23 9 24 8 25 43 62 42 63 41 18 42 17 43 16 44 15 45 14 46 79	
<b>Sample Output</b>	
1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000	
1.00000 2.18978 2.60517 3.00000 3.31662 3.55689 5.00000	
1.00000 7.00000 19.67642 35.00000 45.28797 52.26138 91.00000	
1.00000 9.31362 17.70339 25.66667 32.17424 37.42452 79.00000	
<b>More Test cases</b>	
<a href="#">Input</a> and <a href="#">Output</a> files	

## 2.9. Shoelace Formula

Shoelace Formula determines the area of a [simple polygon](#) whose vertices are given by Cartesian coordinates.

$$A = \frac{\begin{vmatrix} x_1 & x_2 & x_3 & \cdots & x_n & x_1 \\ y_1 & y_2 & y_3 & \cdots & y_n & y_1 \end{vmatrix}}{2} \quad (10)$$

which can be simplified as

$$A = \frac{\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix}}{2} \quad \text{where} \quad \begin{vmatrix} x_i & x_j \\ y_i & y_j \end{vmatrix} = x_i \cdot y_j - x_j \cdot y_i$$

### Problem Statement:

Calculate the area of a given  $n$ -sided polygon for all test cases accurate till 1 decimal place.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_i \quad x_1 \ y_1 \ x_2 \ y_2 \ \cdots \ x_n \ y_n$  ( $2n_i + 1$  space separated integers for each testcase)

##### Output Format

$A_i$  (each test case on a newline, accurate till 1 decimal places)

##### Constraints

$3 \leq n_i \leq 1000$   
 $-10^5 \leq x_i, y_i \leq 10^5$   
The given polygon is simple.

##### Sample Input

```
6
3 0 1 2 3 4 7
3 1 1 5 9 3 5
3 3 4 1 1 4 1
4 -2 4 -2 1 3 -3 4 4
8 458 695 621 483 877 469 1035 636 1061 825 875 1023 645 1033 485 853
10 443 861 470 506 754 432 910 446 952 485 1036 595 1101 721 1045 954 947 1009 712 1095
```

##### Sample Output

```
2.0
0.0
4.5
28.5
255931.0
325573.5
```

##### More Test cases

[Input](#) and [Output](#) files

**Fun Video.** [Gauss's magic shoelace area formula and its calculus companion](#)

## 2.10. Simpson's Rule

Simpson's Rule is a method in numerical integration (approximating definite integrals).

It approximates the area of  $f(x)$  in the interval  $[a, b]$  by area of parabola passing through  $a, \frac{a+b}{2}, b$ . as shown in 6.

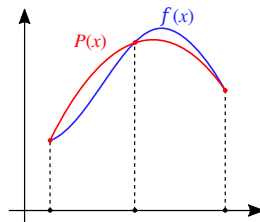


Figure 6: Approximating  $f(x)$  by a parabola  $P(x)$ . (Image by Popletibus licensed under CC BY-SA 4.0)

$$\int_a^b f(x) dx \approx \frac{b-a}{6} \left[ f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] \quad (11)$$

If 11 is applied to  $n$  equally spaced subdivisions in  $[a, b]$ , we get the *composite Simpson's rule* 12.

$$\int_a^b f(x) dx \approx \frac{\Delta x}{3} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots + 4f(x_{n-1}) + f(x_n)) \quad (12)$$

where each of the  $n+1$  ordinates is given by  $x_i = a + i\Delta x$  for  $i = 0, 1, \dots, n$  and  $\Delta x = \frac{b-a}{n}$

**Note.** *Simpson's rule can only be applied when an odd number of ordinates is chosen.*

**Problem Statement:**

$$\pi = \frac{22}{7} - \int_0^1 \frac{x^4(1-x)^4}{1+x^2} dx \quad (13)$$

Calculate  $\pi_n$  (approximate 13 using  $n$  ordinates) for all test cases (accurate till 15 decimal places).

<b>Starter Code</b>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1 \ n_2 \ \dots \ n_t$	( $t$ space separated integers for each testcase)
<b>Output Format</b>	
$\pi_{n_i}$	(each test case on a newline, accurate till 15 decimal places)
<b>Constraints</b>	
$0 < n_i < 500$ and $n_i$ is odd	
<b>Sample Input</b>	
10	
3 5 7 11 15 31 57 99 163 441	
<b>Sample Output</b>	
3.140773809523810	
3.141684884891457	
3.141601987350571	
3.141593090129691	
3.141592711563659	
3.141592654188570	
3.141592653603947	
3.141592653590286	
3.141592653589817	
3.141592653589793	

## §3. Traditional Conditionals

**Topics.** *if else statement, loop control statements (break, continue), more data types (bool, char) and, logical NOT, AND, OR operators (!, &&, || respectively) and previous sections.*

### 3.1. Triangle Types

Triangles can be classified using sides and angles as follows:

#### 3.1.1 By Side

**Scalene** All sides different

**Isosceles** Any two sides equal

**Equilateral** All sides equal

#### 3.1.2 By Angle

**Acute** All angles  $< 90^\circ$

**Right** One angle  $= 90^\circ$

**Obtuse** One angle  $> 90^\circ$

#### Problem Statement:

Given the three sides of the triangle  $a, b, c$ , output the type of triangle by side and angle. Also check the validity of given sides i.e., output "NOT A TRIANGLE" if the given sides does not form a triangle.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $a_i \ b_i \ c_i$  (three space separated integers for each testcase)

##### Output Format

Type by side & Type by angle (each test case on a newline)

##### Constraints

$1 \leq a, b, c \leq 100$

##### Sample Input

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

##### Sample Output

```
NOT A TRIANGLE
Scalene & Obtuse
Scalene & Right
Scalene & Acute
Isosceles & Acute
Isosceles & Obtuse
Equilateral & Acute
```

### 3.2. Clock Angle

**Problem Statement:**

Determine the pairwise angle between the hour, minute and second hand of a 24-hour clock at given time.  
Let

- $\angle_{HM}$  denote angle between hour hand and minute hand.
- $\angle_{HS}$  denote angle between hour hand and second hand.
- $\angle_{MS}$  denote angle between minute hand and second hand.

**Note.** Calculate the convex angle between pair of hands i.e.,  $0 \leq \angle_{ij} \leq 180$ .

**Starter Code**

**Input Format**

$t$  (number of test cases, an integer)  
Hours:Minutes:Seconds (three colon seperated integers for each testcase)

**Output Format**

$\angle_{HM}$   $\angle_{HS}$   $\angle_{MS}$  (three space seperated angles (in degrees, accurate till 4 decimal places)) on a newline

**Constraints**

Given time is a valid; i.e.,  $0 \leq \text{Hours} \leq 23$ ,  $0 \leq \text{Minutes} \leq 59$ ,  $0 \leq \text{Seconds} \leq 59$  (integers)

**Sample Input**

12  
00:00:00  
03:00:00  
21:45:00  
10:10:00  
03:16:36  
09:49:09  
19:38:18  
05:07:11  
11:07:05  
17:19:23  
23:19:17  
23:59:59

**Sample Output**

0.0000 0.0000 0.0000  
90.0000 90.0000 0.0000  
22.5000 67.5000 90.0000  
115.0000 55.0000 60.0000  
1.3000 117.7000 116.4000  
0.3250 119.4250 119.1000  
0.6500 121.1500 121.8000  
110.4917 87.5917 22.9000  
68.9583 56.4583 12.5000  
43.3917 21.6917 21.7000  
136.0583 122.3583 13.7000  
0.0917 5.9917 5.9000

**More Test cases**

[Input](#) and [Output](#) files

### 3.3. Fleur Delacour

Fleur Delacour has an interesting flower. She is also very busy, so she forgets to water the flower sometimes. The flower grows as follows:

- If the flower is watered in the  $i$ -th day, it grows by 1 unit.
- If the flower is watered in the  $i$ -th and in the  $(i - 1)$ -th day ( $i > 1$ ), then it grows by 5 units instead of 1.
- If the flower is not watered in the  $i$ -th day, it does not grow.
- If the flower isn't watered for two days in a row, it dies.

#### Problem Statement:

Calculate the flower's height after  $n$  days given information whether Fleur has watered the flower or not for  $n$  successive days. Take the flower's initial height as 1 unit.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_i \ a_1 \ a_2 \ \dots \ a_{n_i-1} \ a_{n_i}$  ( $n_i + 1$  space separated integers for each testcase)

##### Output Format

The flower's height after  $n_i$  days. If the flower dies, output  $-1$  (each test case on a newline)

##### Constraints

$1 \leq n_i \leq 100$

$$a_i = \begin{cases} 1 & \text{if Fleur waters the flower} \\ 0 & \text{if Fleur does not water the flower} \end{cases}$$

##### Sample Input

```
9
1 0
2 0 0
2 1 0
3 1 0 1
3 0 1 1
5 1 0 1 0 0
5 1 0 1 0 1
5 1 0 1 1 0
10 1 1 1 1 1 1 1 1 1 1
```

##### Sample Output

```
1
-1
2
3
7
-1
4
8
47
```

##### More Test cases

[Input](#) and [Output](#) files

**Note.** Verify your program on even more testcases from [here](#).



### 3.4. Doomsday Algorithm

The Doomsday Algorithm is a method for determining the day of the week for a given date. It takes advantage of some easy-to-remember-dates called *Doomsdates* falling on the same day called *Doomsdays* for a given year. Eg., 3/1 (4/1 leap years), Last Day of Feb, 14/3 (Pi Day), 4/4, 6/6, 8/8, 10/10, 12/12, 9/5, 5/9, 11/7, 7/11.

Watch the [Fun Video](#) or go through the [Wikipedia Article](#) to understand the approach. In short the steps are:

- Find the anchor day for the century.
- Calculate the anchor day for the year (according to the century).
- Select the date (*Doomsdate*) of the given month that falls on doomsday (according to the year).
- Count days between the *Doomsdate* and given date which gives the answer.

#### Problem Statement:

Write a function that calculates the day of the week for any particular date in the past or future.

Consider Gregorian calendar (AD)

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
DD/MM/YYYY (Date Month Year) (three slash seperated integers for each testcase)

##### Output Format

Day of the Week (each test case on a newline)

##### Constraints

$1 \leq \text{Date} \leq 99$ ,  $1 \leq \text{Month} \leq 99$ ,  $1 \leq \text{Year} \leq 9999$  (integers)

##### Sample Input

8  
01/01/0001  
19/02/1627  
29/02/1700  
15/04/1707  
22/12/1887  
23/06/1912  
01/01/2000  
15/03/2020

##### Sample Output

Monday  
Friday  
INVALID DATE!  
Friday  
Thursday  
Sunday  
Saturday  
Sunday

##### More Test cases

[Input](#) and [Output](#) files

**Fun Video.** [The Doomsday Algorithm – Numberphile](#)

## §4. Iteration Domination

**Topics.** for, while & do while loops and previous sections.

### 4.1. Pisano Period

The Fibonacci numbers are the numbers in the integer sequence defined by the following recurrence relation

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \quad n \in \mathbb{Z} \quad (\text{Yes! They can be extended to negative numbers}) \end{aligned} \tag{14}$$

For any integer  $n$ , the sequence of Fibonacci numbers  $F_i \% n$  is periodic.

The Pisano period, denoted  $\pi(n)$ , is the length of the period of this sequence.

For example, the sequence of Fibonacci numbers modulo 3 begins:

0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, ... ([A082115](#))

This sequence has period 8, so  $\pi(3) = 8$ .

Basically, the remainders repeat when these numbers are divided by  $n$ . You have to find this period.

#### Problem Statement:

Find Pisano period of  $t$  numbers  $n_1, n_2, \dots, n_t$

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 \ n_2 \ \dots \ n_t$  ( $t$  space separated numbers for each testcase)

##### Output Format

$\pi(n_i)$  (each test case space separated)

##### Constraints

$1 < n_i \leq 1000$

##### Sample Input

17  
2 3 5 8 13 21 34 55 89 144 233 987 30 50 98 750 1000

##### Sample Output

3 8 20 12 28 16 36 20 44 24 52 32 120 300 336 3000 1500

**Fun Video.** [Fibonacci Mystery – Numberphile](#)

## 4.2. Palindromic Number

A non-negative integer is a Palindromic number if it remains the same when it's digits are reversed.

### Problem Statement:

Determine whether the given integer is a Palindrome for all test cases.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 \ n_2 \ \dots \ n_t$  ( $t$  space separated integers for each testcase)

##### Output Format

"yes" if  $n_i$  is a Palindrome else "no". (each test case on a newline)

##### Constraints

$0 \leq n_i \leq 10^9$

##### Sample Input

13  
1 7 15 22 196 666 1212 96096 111111 8801088 9256713 40040004 123454321

##### Sample Output

yes  
yes  
no  
yes  
no  
yes  
no  
no  
yes  
yes  
no  
no  
yes

**Fun Video.** [Why 02/02/2020 is the most palindromic date ever. – Stand-up Maths](#)  
[Every Number is the Sum of Three Palindromes – Numberphile](#)

### 4.3. Kempner Series

Kempner series is **Harmonic** series where all terms whose denominator contains 9 are excluded.

$$K_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{8} + \frac{1}{10} + \dots + \frac{1}{n} = \sum_{i=1}^n c_i \frac{1}{i} \text{ where } c_i = \begin{cases} 0 & \text{if } i\text{'s decimal expansion contains a 9} \\ 1 & \text{else} \end{cases} \quad (15)$$

**Fun Fact.** Unlike Harmonic series, the Kempner series **converges** to around 22.92.

This is because most large integers contain a 9, hence they will be excluded from the sum.

**Problem Statement:**

Calculate  $K_n$  for all test cases accurate till 10 decimal places.

#### Starter Code

##### Input Format

$t$

(number of test cases, an integer)

$n_1 \ n_2 \ \dots \ n_t$

( $t$  space separated integers for each testcase)

##### Output Format

$K_{n_i}$

(each test case on a newline, accurate till 10 decimal places)

##### Constraints

$1 \leq n_i \leq 10^6$

##### Sample Input

11

1 2 3 5 10 20 30 50 100 1000 1000000

##### Sample Output

1.0000000000

1.5000000000

1.8333333333

2.2833333333

2.8178571429

3.4339969671

3.7967616822

4.2549307007

4.7818487651

6.5907201903

11.0156518499

**Fun Video.** [3 is everywhere – Numberphile](#)

## 4.4. Base $-2$

By using  $-2$  as the base, both positive and negative integers can be expressed without an explicit sign or other irregularity. Just like positive integral bases, any base  $-2$  number can be represented as follows:

$$(a_n \dots a_2 a_1 a_0)_{(-2)} = a_n(-2)^n + \dots + a_2(-2)^2 + a_1(-2)^1 + a_0(-2)^0 \quad \text{where } a_i \text{ is either 0 or 1} \quad (16)$$

To find base  $-2$  representation of  $n$ , we repeatedly divide by  $-2$  until the quotient becomes 0 and the remainders generated (which are either 0 or 1) will be the digits of base  $-2$  representation.

$$n = \text{Quotient} \times (-2) + \text{Reminder} \quad \rightarrow \quad \text{Quotient} = \text{Quotient}_{\text{new}} \times (-2) + \text{Reminder}_{\text{new}}$$

For  $-3$ , the process is as shown below,

$$\begin{aligned} -3 &= 2 \times (-2) + 1 && \rightarrow a_0 = 1 \\ 2 &= -1 \times (-2) + 0 && \rightarrow a_1 = 0 \\ -1 &= 1 \times (-2) + 1 && \rightarrow a_2 = 1 \\ 1 &= 0 \times (-2) + 1 && \rightarrow a_3 = 1 \end{aligned}$$

Hence  $(-3)_{10} = (1101)_{(-2)}$ .

**Note.** C++'s `%` operator may give negative values when the dividend or divisor is negative.

For example,  $(-1)\%(2) = (-1)\%(-2) = -1 \neq 1$ .

### Problem Statement:

Convert the given decimal number into base  $-2$  for all test cases.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 \ n_2 \ \dots \ n_t$  ( $t$  space separated integers for each testcase)

##### Output Format

Converted base  $-2$  number (each test case on a newline)

##### Constraints

$-200 \leq n_i \leq 200$

##### Sample Input

10  
-4 -3 -2 -1 0 1 2 3 4 100

##### Sample Output

1100  
1101  
10  
11  
0  
1  
110  
111  
100  
110100100

##### More Test cases

[Input](#) and [Output](#) files

Fun Video. [Base 12 – Numberphile](#)

## 4.5. Base Conversion

In this problem, you will convert binary number to decimal and vice versa.

**Hint.** First solve the conversion problem for integers and then try to incorporate their fractional part.

(a) **Problem Statement:**

Convert  $t$  positive binary numbers  $(n_1, n_2, \dots, n_t)$  to decimal.

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1 \ n_2 \ \dots \ n_t$	( $t$ space seperated numbers for each testcase)
<hr/>	
<b>Output Format</b>	
Converted decimal number	(space seperated)
<hr/>	
<b>Constraints</b>	
$0 \leq n_i \leq 10^{15}$ , a maximum of 8 digits after binary point ('.')	(base 2, a double)
<hr/>	
<b>Sample Input</b>	
9	
1 111 110001 101010111 100101100001 1.00011001 11.001001 110.01 10110.01110101	
<hr/>	
<b>Sample Output</b>	
1.00000000 7.00000000 49.00000000 343.00000000 2401.00000000 1.09765625 3.14062500 6.25000000 22.45703125	

(b) **Problem Statement:**

Convert  $t$  positive decimal numbers  $(n_1, n_2, \dots, n_t)$  to binary.

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1 \ n_2 \ \dots \ n_t$	( $t$ space seperated numbers for each testcase)
<hr/>	
<b>Output Format</b>	
Converted binary number truncated till 8 decimal places	(space seperated)
<hr/>	
<b>Constraints</b>	
$0 \leq n_i \leq 2500$ , a maximum of 8 digits after decimal point ('.')	(base 10, a double)
<hr/>	
<b>Sample Input</b>	
9	
1 7 49 343 2401 1.1 3.1415 6.25 22.459	
<hr/>	
<b>Sample Output</b>	
1.00000000 111.00000000 110001.00000000 101010111.00000000 100101100001.00000000 1.00011001 11.00100100 110.01000000 10110.01110101	

**Fun Video.** [Dungeon Numbers – Numberphile](#)

## §5. Function Admiration

**Topics.** functions, *passing by value & reference and previous sections.*

For this problem set, try to modularise as much as possible; i.e., make functions for sensible repeating parts.

### 5.1. Collatz Conjecture

Consider the following operation on an arbitrary positive integer:

- If the number is even, divide it by two.
- If the number is odd, triple it and add one.

This operation can be defined using the function  $f$  as follows:

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n + 1 & \text{if } n \text{ is odd} \end{cases} \quad (17)$$

Also note that the function updates  $n$  itself.

Let  $\{a_i\}$  be the sequence of values  $n$  acquires by applying  $f$  repeatedly.

Collatz conjecture states that for every positive integer this procedure will eventually reach 1.

For example, if initial value of  $n = 3$ , 1 is reached in seven operations .

$$3 \xrightarrow[(1)]{3 \times 3 + 1} 10 \xrightarrow[(2)]{10/2} 5 \xrightarrow[(3)]{3 \times 5 + 1} 16 \xrightarrow[(4)]{16/2} 8 \xrightarrow[(5)]{8/2} 4 \xrightarrow[(6)]{4/2} 2 \xrightarrow[(7)]{2/2} 1$$

#### Problem Statement:

Your task is to return the number of operations required to reach 1<sup>1</sup> for arbitrary number of inputs.

<b>Starter Code</b>
<b>Input Format</b> $n_1 \ n_2 \ \dots n_i \ \dots -1$ (space separated arbitrary number of testcases, stop when input is negative)
<b>Output Format</b> number of operations required to reach 1 with initial value of $n = n_i$ (space separated for each test case)
<b>Constraints</b> $1 \leq n_i \leq 10^6$
<b>Function(s) to Implement</b> void f(long long &n) – updates value of $n$ int count_operations(long long n) – returns the number of operations required to reach 1
<b>Sample Input</b> 1 3 7 9 27 255 871 4255 77031 665215 837799 -1
<b>Sample Output</b> 0 7 16 19 111 47 178 201 350 441 524

**Fun Video.** [Collatz Conjecture: The Simplest Math Problem No One Can Solve – Veritasium](#)

<sup>1</sup>As of 2020, the conjecture has been checked by computer for all starting values up to  $2^{68} \approx 2.95 \times 10^{20}$ , so sequence from  $n$  will reach 1 for the given constraints.

## 5.2. Friendly Pair

Two positive integers form a Friendly pair if they have a common abundance index.

The abundance index of a number is the ratio of sum of divisors of that number and the number itself.

$$\text{abundance index} = \frac{\sigma(n)}{n} \quad \text{where } \sigma(n) \text{ is the sum of divisors of } n \quad (18)$$

For example, 6 and 28 form a friendly pair<sup>2</sup> as

$$\frac{\sigma(6)}{6} = \frac{1+2+3+6}{6} = \frac{12}{6} = 2 = \frac{56}{28} = \frac{1+2+4+7+14+28}{28} = \frac{\sigma(28)}{28}$$

### Problem Statement:

Given two numbers  $a, b$  check if they form a friendly pair.

Express the common abundance (if it exists) as a quotient  $p/q$  where  $p, q$  are integers and  $q \neq 0$ .

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $a_1 \ b_1 \ a_2 \ b_2 \ \dots \ a_t \ b_t$  ( $t$  space separated integer pairs for each testcase)

##### Output Format

Output the common abundance if  $a_i, b_i$  form a friendly pair else output  $-1$  (each test case on a newline)  
 $p_{a_i}/q_{a_i}$  (where common abundance  $= p_{a_i}/q_{a_i}$  and  $p_{a_i}, q_{a_i}$  are integers &  $q_{a_i} \neq 0$  in irreducible form)

##### Constraints

$1 < a_i, b_i \leq 10^9$

##### Function(s) to Implement

`long long sum_of_divisors(int n)` – returns the sum of divisors of  $n$   
`bool friendly_pair_check(int a, int b)` – outputs the common abundance index or  $-1$

##### Sample Input

```
10
6 28 10 20 30 140 30 2480 135 819 42 544635 1556 9285 4320 4680
693479556 8640 84729645 155315394
```

##### Sample Output

```
2
-1
12/5
12/5
16/9
16/7
-1
7/2
127/36
896/351
```

**Fun Video.** [A Video about the Number 10 – Numberphile](#)

<sup>2</sup>in fact, they are called perfect numbers as their abundance  $= 2$



### 5.3. Gauss Circle Problem

Consider a circle in the  $x - y$  plane with center at the origin and radius  $r \geq 0$  ( $r \in \mathbb{R}$  such that  $r^2 = n \in \mathbb{Z}$ ). Gauss's circle problem asks the number of lattice points  $N(r)$  in the interior or on the circumference of this circle. These points are of the form  $(x, y) \in \mathbb{Z}^2$  such that  $x^2 + y^2 \leq r^2 = n$ . Also, note that  $N(r) \sim \pi r^2$  (why?).

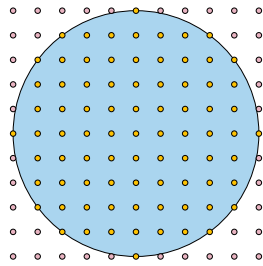


Figure 7: A circle with  $r = 5$  units bounding 81 integer points.  $N(r) = 81 \sim \pi r^2 \approx 78.54$

Consider the subproblem of finding  $M(i)$  – the number of  $(x, y) \in \mathbb{Z}^2$  such that  $x^2 + y^2 = i$  where  $i \in \{; 0, 1, \dots, n\}$ .

Clearly  $N(r) = \sum_{i=0}^{r^2} M(i) \rightarrow N(\sqrt{n}) = \sum_{i=0}^n M(i)$ . Now,

$$M(i) = 4 \sum_{j|i} \chi(j) \quad \text{where} \quad \chi(n) = \begin{cases} 1 & \text{if } n \% 4 = 1 \\ -1 & \text{if } n \% 4 = 3 \\ 0 & \text{else} \end{cases} \tag{19}$$

**Problem Statement:**

Calculate  $N(\sqrt{n})$  for a given  $n$ ; i.e. the number of lattice points  $(x, y)$  such that  $x^2 + y^2 \leq n$ .

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1 \ n_2 \ \dots \ n_t$	(number of test cases, an integer) ( $t$ space separated integers for each testcase)
<b>Output Format</b> $N(\sqrt{n_i})$	(each test case space separated)
<b>Constraints</b> $1 < n_i \leq 10^7$	
<b>Function(s) to Implement</b> int X(int n) – returns $\chi(n)$ int count_lattice_points(int n) – returns $M(n)$	
<b>Sample Input</b> 15 0 1 2 3 5 10 20 30 50 100 1000 10000 100000 1000000 10000000	
<b>Sample Output</b> 1 5 9 9 21 37 69 97 161 317 3149 31417 314197 3141549 31416025	

**Interesting Observation.** Does the last few outputs look familiar? How can this happen?  $\therefore$   
Also, if the last output took a long time then think how you can do the calculations faster?

**Fun Video.** [Pi hiding in prime regularities – 3Blue1Brown](#)  
[Your New Favorite Formula For Pi – BriTheMathGuy](#)

### 5.4. Euler’s Totient Function

Euler’s totient function  $\varphi(n)$  is the number of positive integers  $\leq n$  that are co-prime to  $n$ .  
A simple approach to calculating this function is to count the integers  $i$ ’s such that  $1 \leq i \leq n$  and  $\gcd(i, n) = 1$ .  
But there is a *better* way using the Euler’s Product Formula

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right) \qquad \text{For all primes } p \leq n \tag{20}$$

So, if  $n = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}$ , where  $p_1, p_2, \dots, p_r$  are the distinct primes dividing  $n$

$$\varphi(n) = p_1^{k_1-1}(p_1-1) p_2^{k_2-1}(p_2-1) \cdots p_r^{k_r-1}(p_r-1)$$

**Problem Statement:**  
Calculate  $\varphi(n)$  for a given  $n$

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1 \ n_2 \ \dots \ n_t$	(number of test cases, an integer) ( $t$ space seperated integers for each testcase)
<b>Output Format</b> $\varphi(n_i)$	(each test case on a newline)
<b>Constraints</b> $1 < n_i \leq 10^9$	
<b>Function(s) to Implement</b> <code>int totient(int n) – returns <math>\varphi(n)</math></code>	
<b>Sample Input</b> 13 1 4 8 20 44 69 97 120 2520 55440 277200 720720 88888888	
<b>Sample Output</b> 1 2 4 8 20 44 96 32 576 11520 57600 138240 12690687	

### 5.5. Regular Star Polygon

A regular star polygon is a self-intersecting, equilateral equiangular polygon. It is denoted by Schläfli symbol  $\{n/m\}$  where  $n$  is the number of vertices and  $m$  is the density (sum of the turn angles of all the vertices  $360^\circ$ ).

**Construction via vertex connection** Connect every  $m^{\text{th}}$  point out of  $n$  points regularly spaced on a circle. For example, check out the demo videos for constructing  $\{7, 2\}$  and  $\{7, 3\}$ .

So a seven-pointed star can be obtained in two-ways,

By connecting vertex 1 to 3, then 3 to 5, then 5 to 7, then 7 to 2, then 2 to 4, then 4 to 6, then 6 to 1 or by

By connecting vertex 1 to 4, then 4 to 7, then 7 to 3, then 3 to 6, then 6 to 2, then 2 to 5, then 5 to 1.

**Problem Statement:**

Construct the  $\{n/m\}$  regular star polygon for given  $n, m$ .

Starter Code

Input Format

$m$   $n$

(2 space seperated integers)

Output Format

Regular Star Polygon with Schläfli symbol  $\{n/m\}$

Constraints

$1 \leq n \leq 50, 1 \leq m < n/2$

Function(s) to Implement

void regular\_star\_polygon(int n, int m) – draws the corresponding regular star polygon

Sample Input

See 8.

Sample Output

See 8.

	3	4	5	6	n	7	8	9	10	11	12
1											
2											
m											
3											
4											
5											

Figure 8: Some inputs  $m, n$  and their corresponding star polygons in a tabular fashion.

**Fun Video.** [The 3-4-7 miracle. Why is this one not super famous? – Mathologer](#)

## §6. Recursion Salvation

**Topics.** recursive functions and previous sections.

### 5 Simple Steps for Solving Any Recursive Problem

(Courtesy – [Reducible](#))

- What's the simplest possible input?
- Play around with examples and visualize!
- Relate hard cases to simpler cases
- Generalize the pattern
- Write code by combining recursive pattern with base case

#### 6.1. Ackermann Function

Ackermann Function is defined as follows

$$\begin{aligned}A(0, n) &= n + 1 \\A(m, 0) &= A(m - 1, 1) \\A(m, n) &= A(m - 1, A(m, n - 1))\end{aligned}\tag{21}$$

**Problem Statement:**

Calculate  $A(m, n)$  (given  $m, n$ ) for all test cases.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $m_1\ n_1\ \ m_2\ n_2\ \ \dots\ \ m_t\ n_t$  ( $t$  space separated integer pairs for each testcase)

##### Output Format

$A(m_i, n_i)$  (each on a newline)

##### Constraints

$m_i, n_i$  are positive integers such that  $A(m_i, n_i)$  is within the range of int

##### Function(s) to Implement

int A(int m, int n) – returns  $A(m, n)$

##### Sample Input

10  
0 0 0 5 1 0 1 3 2 4 3 1 3 3 3 9 4 0 4 1

##### Sample Output

1  
6  
2  
5  
11  
13  
61  
4093  
13  
65533

**Interesting Observation.** Was your program able to compute the last output? Why not? How to fix this?

**Fun Video.** [The Most Difficult Program to Compute? – Computerphile](#)

## 6.2. Horner's Method

Consider, the problem of evaluating a polynomial given its coefficients.

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \cdots + a_n \cdot x^n$$

A naive method is to evaluate  $x^0, x^1, x^2, \dots, x^n$  independently, then multiply  $x^i$  with  $a_i$  and add all results.

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x \cdot x + a_3 \cdot x \cdot x \cdot x + \cdots + a_n \underbrace{x \cdot x \cdots x}_{n \text{ times}}$$

This approach takes  $1 + 2 + \cdots + n = n(n+1)/2$  multiplications and  $n$  additions.

It can be improved by using the precalculated  $x^{i-1}$  and multiplying it by  $x$  to get  $x^i$ . This reduces the number of multiplications significantly to  $2n - 1$  while keeping the number of additions  $n$ .

$$f(x) = a_0 + a_1 \cdot x^0 \cdot x + a_2 \cdot x^1 \cdot x + a_3 \cdot x^2 \cdot x + \cdots + a_n x^{n-1} \cdot x$$

But surprisingly there is an even better way! Horner's Method as described in 22, is an optimal algorithm for polynomial evaluation needing only  $n$  multiplications and  $n$  additions.

$$f(x) = a_0 + x \left( a_1 + x \left( a_2 + x \left( a_3 + \cdots + x (a_{n-1} + x a_n) \cdots \right) \right) \right) \quad (22)$$

### Problem Statement:

Evaluate polynomial given by coefficients at  $x$  using Horner's Method for all test cases.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $x_i \quad n_i \quad a_0 \ a_1 \ a_2 \cdots a_{n_i}$  ( $n_i + 3$  space separated integers for each testcase)

##### Output Format

$f(x_i)$  (each on a newline)

##### Constraints

$1 < x_i, n_i, a_i \leq 10^4$

Also assume that the calculations are always within the range of long long

##### Function(s) to Implement

long long f(const int &x, int a, int b) – returns  $f(x)$ , you are also given two extra parameters.

##### Sample Input

```
6
1 0 1
2 1 -3 2
2 2 15 -8 7
3 3 2 -1 -3 4
5 6 21 10 19 47 48 9 27
3 14 -1 59 265 -35 8 -97 -932 38 4 -62 -643 38 -3 27 950
```

##### Sample Output

```
1
1
27
80
486421
4552224296
```

**Interesting Observation.** If recursion was not allowed do you think it would be possible to solve this problem given it's input order was  $(a_0 \ a_1 \ a_2 \cdots a_{n_i})$ ? Problem 2.5 had inputs in reverse order  $a_{n_i} \ a_{n_i-1} \ \dots \ a_1 \ a_0$ . By taking inspiration from recursion, solve it when the inputs are in correct order  $(a_0 \ a_1 \ a_2 \cdots a_{n_i})$ .

**Fun Video.** [How Imaginary Numbers Were Invented – Veritasium](#)

6.3. Modular Exponentiation

Consider the problem of calculating  $x^y \pmod k$  (i.e. the remainder when  $x^y$  is divided by  $k$ ).  
A naive approach is to keep multiplying by  $x$  (and take  $\pmod k$ ) until we reach  $x^y$ .<sup>3</sup>

$$x \pmod k \rightarrow x^2 \pmod k \rightarrow x^3 \pmod k \rightarrow x^4 \pmod k \rightarrow \cdots \rightarrow x^y \pmod k$$

We can use a much faster method which involves *repeated squaring* of  $x \pmod k$

$$x \pmod k \rightarrow x^2 \pmod k \rightarrow x^4 \pmod k \rightarrow x^8 \pmod k \rightarrow \cdots \rightarrow x^{2^{\lceil \log y \rceil}} \pmod k \tag{23}$$

The idea is to multiply some of the above numbers and get  $x^y \pmod k$ .  
This is achieved by choosing all powers that have 1 in binary representation of  $y$ .  
For example,

$$x^{25} = x^{11001_2} = x^{1000_2} \cdot x^{100_2} \cdot x^{1_2} = x^{16} \cdot x^8 \cdot x^1$$

which gives,

$$x^{25} \pmod k = ((x^{16} \pmod k) \cdot (x^8 \pmod k) \cdot (x^1 \pmod k)) \pmod k$$

- (a) **Problem Statement:**  
Calculate  $x^y \pmod k$  using the above method for  $n$   $(x, y, k)$  triples. Take  $k = 10^9 + 7$ . [why this number?](#)

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b> $t$ $x_1\ y_1\ \ \ x_2\ y_2\ \ \ \dots\ \ \ x_t\ y_t$	(number of test cases, an integer) ( $t$ space seperated integer pairs for each testcase)
<hr/>	
<b>Output Format</b> $x_i^{y_i} \pmod k$	(each test case on a newline)
<hr/>	
<b>Constraints</b> $1 < x_i, y_i \leq 10^9$	
<hr/>	
<b>Function(s) to Implement</b> int mod_exp(int x, int y, int k) – returns $x^y \pmod k$	
<hr/>	
<b>Sample Input</b> 5 3 4 2 8 123 123 129612095 411099530 241615980 487174929	
<hr/>	
<b>Sample Output</b> 81 256 921450052 276067146 838400234	

**Note.** Before proceeding to next task, verify your program on more testcases from [here](#).

- (b) **Problem Statement:**  
Calculate  $x^y \pmod k$  using the above method for  $n$   $(x, y, k)$  triples. Take  $k = 10^9 + 7$ . [why this number?](#)

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b> $t$ $x_1\ y_1\ z_1\ \ \ x_2\ y_2\ z_2\ \ \ \dots\ \ \ x_t\ y_t z_t$	(number of test cases, an integer) ( $t$ space seperated triples for each testcase)
<hr/>	
<b>Output Format</b> $x_i^{y_i z_i} \pmod k$	(each test case on a newline)
<hr/>	
<b>Constraints</b> $1 < x_i, y_i, z_i \leq 10^9$	
<hr/>	
<b>Function(s) to Implement</b> int mod_super_exp(int x, int y, int z, int k) – returns $x^{y^z} \pmod k$	
<hr/>	
<b>Sample Input</b> 5 3 7 1 15 2 2 3 4 5 427077162 725488735 969284582 690776228 346821890 923815306	
<hr/>	
<b>Sample Output</b> 2187 50625 763327764 464425025 534369328	

**Note.** Verify your program on more testcases from [here](#).

**Fun Video.** [Square & Multiply Algorithm - Computerphile](#)

<sup>3</sup>this works because  $(a \cdot b) \pmod m = ((a \pmod m) \cdot (b \pmod m)) \pmod m$

## 6.4. Partitions

A partition of a natural number  $n$  is a way of decomposing  $n$  as sum of natural numbers  $\leq n$ .

For example, there are 5 partitions of 4 given by  $\{4, 3 + 1, 2 + 2, 2 + 1 + 1, 1 + 1 + 1 + 1\}$ .

Let us denote the number of partitions of  $n$  by  $P(n)$ .

Now, we move to a seemingly unrelated theorem.

**Theorem 1** (Pentagonal Number Theorem). *PNT relates the product and series representations of the [Euler function](#)*

$$\prod_{n=1}^{\infty} (1 - x^n) = \sum_{k=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2} = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k+1)/2} + x^{k(3k-1)/2}) \quad (24)$$

In other words,

$$(1 - x)(1 - x^2)(1 - x^3) \cdots = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - \cdots$$

The exponents  $1, 2, 5, 7, 12, \dots$  on the right hand side are called (generalized) pentagonal numbers ([A001318](#)).

They are given by the formula  $p_k = k(3k - 1)/2$  for  $k = 1, -1, 2, -2, 3, -3, \dots$

Equation [24](#) implies a recurrence relation for calculating  $P(n)$  given by

$$P(n) = P(n - 1) + P(n - 2) - P(n - 5) - P(n - 7) + \cdots = \sum_{k \neq 0} (-1)^{k-1} P(n - p_k) \quad (25)$$

### Problem Statement:

Calculate  $P(n)$  for all test cases using [24](#) or otherwise :).

#### Starter Code

##### Input Format

$t$

(number of test cases, an integer)

$n_1 \ n_2 \ \dots \ n_t$

( $t$  space separated integers for each testcase)

##### Output Format

$P(n_i)$

(each test case on a newline)

##### Constraints

$1 \leq n_i \leq 40$

##### Function(s) to Implement

int P(int n) – returns  $P(n)$

##### Sample Input

9

1 2 3 4 5 10 20 30 40

##### Sample Output

1

2

3

5

7

42

627

5604

37338

**Fun Video.** [Partitions – Numberphile](#)

[The hardest What comes next \(Euler's pentagonal formula\) – Mathologer](#)

6.5. Hereditary Representation

The usual base  $b$  representation is of a natural number is given by

$$n_b = a_0 \cdot b^0 + a_1 \cdot b^1 + \dots \quad \text{where } a_i\text{'s} \in \{0, 1, \dots, b - 1\}$$
 (26)

Here the power  $i$  of exponent  $b^i$  is in decimal but what if we continue to represent  $i$  in base  $b$  until we use only  $0, 1, 2, \dots, b - 1$  for all exponents of  $b$ .

This is the Hereditary Representation! Representing a natural number  $n_b$  in base  $b$  using only  $0, 1, 2, \dots, b - 1$  as exponents of  $b$ .

To generate this representation, find the usual base representation of the number and then represent its exponents also in the usual base representation. Keep repeating this until there is no exponent  $> b$ .

For example,

$$\begin{aligned} 666_2 &= 2^1 + 2^3 + 2^4 + 2^7 + 2^9 \\ &= 2^1 + 2^{2^0+2^1} + 2^{2^2} + 2^{2^0+2^1+2^2} + 2^{2^0+2^3} \\ &= 2^1 + 2^{2^0+2^1} + 2^{2^{2^1}} + 2^{2^0+2^1+2^{2^1}} + 2^{2^0+2^{2^0+2^1}} \end{aligned}$$
 (27)

Here are some more examples to get familiar,

$$\begin{aligned} 10_2 &= 2^1 + 2^{2^0+2^1} \\ 100_2 &= 2^{2^1} + 2^{2^0+2^{2^1}} + 2^{2^1+2^{2^1}} \\ 3435_3 &= 2 \cdot 3^1 + 3^{3^1} + 2 \cdot 3^{2 \cdot 3^0+3^1} + 3^{2 \cdot 3^1} + 3^{3^0+2 \cdot 3^1} \\ 754777787027_{10} &= 7 \cdot A^0 + 2 \cdot A^1 + 7 \cdot A^3 + 8 \cdot A^4 + 7 \cdot A^5 + 7 \cdot A^6 + 7 \cdot A^7 + 7 \cdot A^8 + 4 \cdot A^9 + 5 \cdot A^{A^1} + 7 \cdot A^{A^0+A^1} \end{aligned}$$

Problem Statement:

Output the Hereditary Representation of the input natural number  $n$  in base  $b$  ( $\geq 2$ ) following the below conventions:

- Use  $+$ ,  $*$  to denote addition (add space between operands), multiplication (no space between operands) respectively and  $b^{\{y\}}$  for  $b^y$  where  $y$  is some expression.
- The powers of base representation are in increasing order (first  $b^0$  then  $b^1$  then  $b^2$  and so on).
- Powers are displayed only when their coefficients are  $> 0$  (non-zero).
- Coefficients themselves are only displayed when they are  $> 1$ .
- The exponents between  $0$  and  $b - 1$  must not be simplified further. So,  $b$  is represented as  $b^{\{1\}}$  and not as  $b^{\{b^{\{0\}}\}}$ .
- For bases  $> 10$ , use capital alphabets ( $A, B, C, \dots, Z$ ) to denote  $(10, 11, 12, \dots, 35)$  respectively.

<b>Starter Code</b>
<div><div><b>Input Format</b></div><div><math>t</math><div>(number of test cases, an integer)</div></div><div><math>n_1 \ b_1 \quad n_2 \ b_2 \quad \dots \quad n_t \ b_t</math><div>(<math>t</math> space seperated pairs (number, base) for each testcase)</div></div></div>
<div><div><b>Output Format</b></div><div>Hereditary Representation of <math>n_i</math> in base <math>b_i</math><div>(each on a newline)</div></div></div>
<div><div><b>Constraints</b></div><div><math>1 &lt; n_i \leq 2 \cdot 10^{18}</math> <math>1 &lt; b_i \leq 35</math></div></div>
<div><div><b>Function(s) to Implement</b></div><div>void Hereditary (long long num, int base) – prints the required representation</div></div>
<div><div><b>Sample Input</b></div><div>9 2 2   10 2   100 2   666 3   3435 3   3816547290 4   3816547290 9   3816547290 35   1162849439785405935 10</div></div>
<div><div><b>Sample Output</b></div><div><math>2^{\{1\}}</math> <math>2^{\{1\}} + 2^{\{2^{\{0\}} + 2^{\{1\}}\}}</math> <math>2^{\{2^{\{1\}}\}} + 2^{\{2^{\{0\}} + 2^{\{2^{\{1\}}\}}\}} + 2^{\{2^{\{1\}} + 2^{\{2^{\{1\}}\}}\}}</math> <math>2^*3^{\{2\}} + 2^*3^{\{3^{\{0\}} + 3^{\{1\}}\}} + 2^*3^{\{2^*3^{\{0\}} + 3^{\{1\}}\}}</math> <math>2^*3^{\{1\}} + 3^{\{3^{\{1\}}\}} + 2^*3^{\{2^*3^{\{0\}} + 3^{\{1\}}\}} + 3^{\{2^*3^{\{1\}}\}} + 3^{\{3^{\{0\}} + 2^*3^{\{1\}}\}}</math> <math>2^*4^{\{0\}} + 2^*4^{\{1\}} + 4^{\{2\}} + 3^*4^{\{3\}} + 3^*4^{\{4^{\{1\}}\}} + 2^*4^{\{2^*4^{\{0\}} + 4^{\{1\}}\}} + 3^*4^{\{3^*4^{\{0\}} + 4^{\{1\}}\}} + 3^*4^{\{2^*4^{\{1\}}\}} + 2^*4^{\{4^{\{0\}} + 2^*4^{\{1\}}\}} + 3^*4^{\{2^*4^{\{0\}} + 2^*4^{\{1\}}\}} + 4^{\{3^*4^{\{0\}} + 2^*4^{\{1\}}\}} + 3^*4^{\{3^*4^{\{1\}}\}} + 2^*4^{\{2^*4^{\{0\}} + 3^*4^{\{1\}}\}} + 3^*4^{\{3^*4^{\{0\}} + 3^*4^{\{1\}}\}}</math> <math>2^*8^{\{0\}} + 3^*8^{\{1\}} + 7^*8^{\{2\}} + 8^{\{3\}} + 6^*8^{\{4\}} + 7^*8^{\{5\}} + 6^*8^{\{6\}} + 3^*8^{\{7\}} + 3^*8^{\{8^{\{1\}}\}} + 4^*8^{\{8^{\{0\}} + 8^{\{1\}}\}} + 3^*8^{\{2^*8^{\{0\}} + 8^{\{1\}}\}}</math> <math>5^*A^{\{0\}} + 3^*A^{\{1\}} + 9^*A^{\{2\}} + 5^*A^{\{3\}} + 4^*A^{\{5\}} + 5^*A^{\{6\}} + 8^*A^{\{7\}} + 7^*A^{\{8\}} + 9^*A^{\{9\}} + 3^*A^{\{A^{\{1\}}\}} + 4^*A^{\{A^{\{0\}} + A^{\{1\}}\}}</math> <math>+ 9^*A^{\{2^*A^{\{0\}} + A^{\{1\}}\}} + 4^*A^{\{3^*A^{\{0\}} + A^{\{1\}}\}} + 8^*A^{\{4^*A^{\{0\}} + A^{\{1\}}\}} + 2^*A^{\{5^*A^{\{0\}} + A^{\{1\}}\}} + 6^*A^{\{6^*A^{\{0\}} + A^{\{1\}}\}} +</math> <math>A^{\{7^*A^{\{0\}} + A^{\{1\}}\}} + A^{\{8^*A^{\{0\}} + A^{\{1\}}\}}</math></div></div>
<div><div><b>More Test cases</b></div><div><a href="#">Input</a> and <a href="#">Output</a> files</div></div>

Fun Video. [Kill the Mathematical Hydra – PBS Infinite Series](#)  
[How Infinity Explains the Finite – PBS Infinite Series](#)



## §7. Paths Paranoia (More Recursion?)

**Topics.** recurrence relations *and previous sections*.

### 7.1. Staircase Walk

Consider a grid with  $m$  horizontal lines and  $n$  vertical lines. A Staircase Walk is defined as the path from bottom-left corner of the grid to the top right corner by walking along the lines; so, the person is constrained to move only in positive  $x$  or positive  $y$  direction.

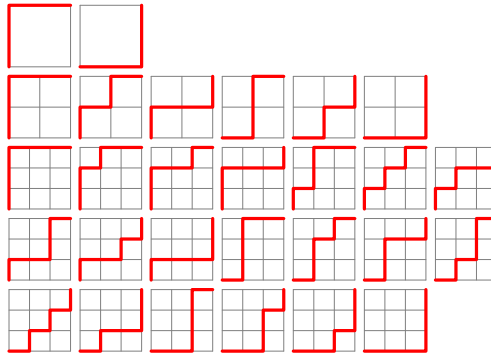


Figure 9: Example walks for case  $m = n = 1$  (#2),  $m = n = 2$  (#6),  $m = n = 3$  (#20) ([Image Source](#))

#### Problem Statement:

Find the number of possible *Staircase Walks* for a given  $m, n$  (for all test cases).

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $m_1 \ n_1 \ m_2 \ n_2 \ \dots \ m_t \ n_t$  ( $t$  space separated integer pairs for each testcase)

##### Output Format

Number of Staircase Walks for  $m_i, n_i$  (each test case on a newline)

##### Constraints

$1 \leq m_i, n_i \leq 15$

##### Function(s) to Implement

`int staircase_walks(int m, int n)` – returns the number of staircase walks for  $m, n$ .

##### Sample Input

```
6
1 1  2 5  6 3  7 10  13 8  15 15
```

##### Sample Output

```
1
5
21
5005
50388
40116600
```

**Fun Video.** [The Devil's Staircase – PBS Infinite Series](#)

[5 = 3 + 4? The Staircase Paradox. Spot The Mistake "Disproving" The Pythagorean Theorem – Mind Your Decisions](#)

## 7.2. Dyck Path

A Dyck Path is **Staircase Walk** ( $m = n$ ) when the path always stays *on or below the diagonal*.

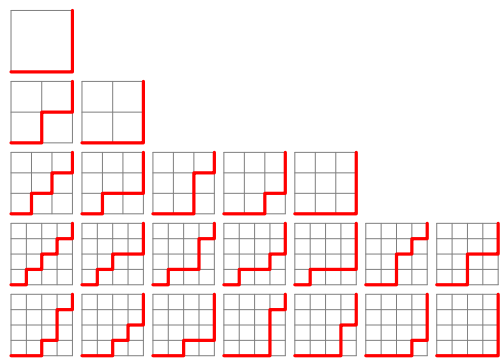


Figure 10: Example walks for case  $n = 1$  (#1),  $n = 2$  (#2),  $n = 3$  (#5),  $n = 4$  (#14) ([Image Source](#))

### Problem Statement:

Find the number of possible *Dyck Path* for a given  $n$  (for all test cases).

<b>Starter Code</b>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1 \ n_2 \ \dots \ n_t$	( $t$ space separated integers for each testcase)
<b>Output Format</b>	
Number of Dyck Paths for $n_i$	(each test case on a newline)
<b>Constraints</b>	
$1 \leq n_i \leq 15$	
<b>Function(s) to Implement</b>	
int dyck_paths(int n) – returns the number of possible staircase walks for $n$ .	
<b>Sample Input</b>	
15	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
<b>Sample Output</b>	
1	
2	
5	
14	
42	
132	
429	
1430	
4862	
16796	
58786	
208012	
742900	
2674440	
9694845	

### 7.3. Delannoy Number

Consider a grid with  $m$  horizontal lines and  $n$  vertical lines. A Delannoy Number is defined as the path from bottom-left corner of the grid to the top right corner by walking along the lines *or diagonally upwards*; so, the person is constrained to move only in positive  $x$  or positive  $y$  or positive  $x - y$  (i.e. along  $y = x$ ) direction.

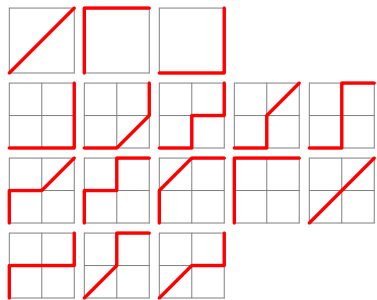


Figure 11: Example walks for case  $m = n = 1$  (#2),  $m = n = 2$  (#6),  $m = n = 3$  (#20) ([Image Source](#))

**Problem Statement:**

Find the number of possible *Delannoy Numbers* for a given  $m, n$  (for all test cases).

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$m_1\ n_1\ \quad m_2\ n_2\ \quad \dots\ \quad m_t\ n_t$	( $t$ space seperated integer pairs for each testcase)
<hr/>	
<b>Output Format</b>	
Number of Delannoy Numbers for $m_i, n_i$	(each test case on a newline)
<hr/>	
<b>Constraints</b>	
$1 \leq m_i, n_i \leq 13$	
<hr/>	
<b>Function(s) to Implement</b>	
<code>int delannoy_number(int m, int n)</code> – returns the number of Delannoy Numbers for $m, n$ .	
<hr/>	
<b>Sample Input</b>	
11 1 1 2 2 3 3 5 5 10 10 13 13 2 5 3 3 6 3 7 10 13 8	
<hr/>	
<b>Sample Output</b>	
3 13 63 1683 8097453 1409933619 61 63 377 433905 8405905	

### 7.4. Schröder Number

A Schroder Number is count of **Delannoy Walks** ( $m = n$ ) when the path always stays *on or below the diagonal*.

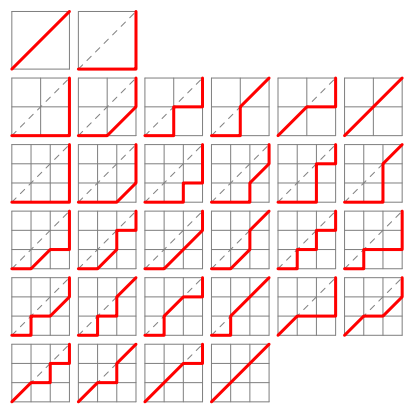


Figure 12: Example walks for case  $n = 1$  (#2),  $n = 2$  (#6),  $n = 3$  (#22) ([Image Source](#))

**Problem Statement:**

Find the *Schroder Number* for a given  $n$  (for all test cases).

<b>Starter Code</b>	
<b>Input Format</b> $t$ $n_1 \ n_2 \ \dots \ n_t$	(number of test cases, an integer) ( $t$ space seperated integers for each testcase)
<b>Output Format</b> Number of Schroder Numbers for $n_i$	(each test case on a newline)
<b>Constraints</b> $1 \leq n_i \leq 14$	
<b>Function(s) to Implement</b> <code>int schroder_number(int n)</code> – returns the number of possible delannoy walks for $n$ .	
<b>Sample Input</b> 14 1 2 3 4 5 6 7 8 9 10 11 12 13 14	
<b>Sample Output</b> 2 6 22 90 394 1806 8558 41586 206098 1037718 5293446 27297738 142078746 745387038	

### 7.5. Motzkin Number

Consider a grid with  $n$  horizontal lines and  $n$  vertical lines. A Motzkin Number is defined as the number of paths from bottom-left corner of the grid to the bottom-right corner which always stays on or above  $x$ -axis by walking horizontally forwards or diagonally upwards or diagonally downwards; so, the person is constrained to move only in positive  $x$  and along  $y = x$  or  $y = -x$  ( $y$  direction can be negative).

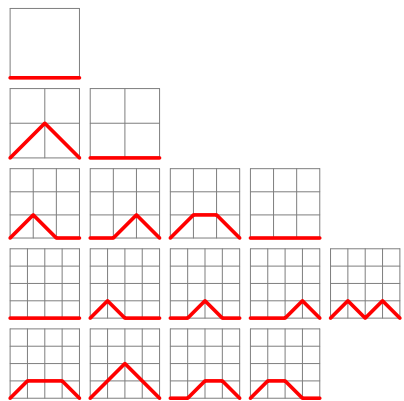


Figure 13: Example walks for case  $n = 1$  (#1),  $n = 2$  (#2),  $n = 3$  (#4),  $n = 4$  (#9) ([Image Source](#))

**Problem Statement:**

Find the *Motzkin Number* for a given  $n$  (for all test cases).

<b>Starter Code</b>	
<hr/>	
<b>Input Format</b>	
$t$	(number of test cases, an integer)
$n_1\ n_2\ \dots\ n_t$	( $t$ space seperated integers for each testcase)
<hr/>	
<b>Output Format</b>	
Number of Motzkin Numbers for $n_i$	(each test case on a newline)
<hr/>	
<b>Constraints</b>	
$1 \leq n_i \leq 20$	
<hr/>	
<b>Function(s) to Implement</b>	
<code>int motzkin_number(int n)</code> – returns the number of possible walks for $n$ .	
<hr/>	
<b>Sample Input</b>	
10 1 2 3 4 5 8 11 14 17 20	
<hr/>	
<b>Sample Output</b>	
1 2 4 9 21 323 5798 113634 2356779 50852019	

## 7.6. Hilbert Curve



Figure 14: Hilbert Curve ([Image Source](#))

### Problem Statement:

Take an integer as input and draw the corresponding iteration of this fractal using turtleSim

You may think along these lines

**Step 1** Find a simple pattern in these iterations.

**Step 2** Think how can you implement this pattern in an efficient way (here think in the number of lines of code you have to write. **Word of caution:** this is just one of the possible definitions of efficient code).

**Step 3** Write the code!

In case you are stuck, here's the starter code!

[Starter Code](#)

Feel free to discuss your thoughts.

**Fun Video.** [Hilbert's Curve: Is infinite math useful? – 3Blue1Brown](#)  
[Recursive PowerPoint Presentations \[Gone Fractal!\] – Stand-up Maths](#)

For more interesting recursive and fractal problems, check out [L-Systems](#).

## §8. Sequence Eminence (Intro to Arrays)

**Topics.** array traversal, manipulation and previous sections. Some problems can be solved without arrays too.

### 8.1. Josephus Problem

Suppose there are  $n$  terrorists around a circle facing towards the centre. They are numbered 1 to  $n$  along clockwise direction. Initially, terrorist 1 has the sword. Now, the terrorist with sword kills the  $k^{\text{th}}$  nearest alive terrorist to its left and passes the sword to  $(k + 1)^{\text{st}}$  nearest alive terrorist to its left. The process repeats. Basically, every  $k^{\text{th}}$  terrorist is killed until only one survives. Then the last terrorist is killed.

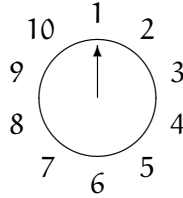


Figure 15: Example arrangement of 10 terrorists

For example, in the above arrangement,

when  $k = 1$ , 1 kills 2, 3 kills 4, 5 kills 6, 7 kills 8, 9 kills 10, 1 kills 3, 5 kills 7, 9 kills 1 and 5 kills 9. So, 5 survives; when  $k = 2$ , 1 kills 3, 4 kills 6, 7 kills 9, 10 kills 2, 4 kills 7, 8 kills 1, 4 kills 8, 10 kills 5 and 4 kills 10. So, 4 survives.

#### Problem Statement:

For a given  $n, k$  pair, and starting position 1, print the terrorists in the sequence they are killed.

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_1 k_1 \quad n_2 k_2 \quad \dots \quad n_t k_t$  ( $t$  space separated pairs (number of terrorists  $n$  and  $k$ ) for each testcase)

##### Output Format

Terrorists in the sequence they are killed (each test case on a newline)

##### Constraints

$1 \leq k_i \leq n_i \leq 100$

##### Sample Input

9  
1 1 2 1 4 1 4 2 8 1 8 3 10 2 16 7 50 25

##### Sample Output

1  
2 1  
2 4 3 1  
3 2 4 1  
2 4 6 8 3 7 5 1  
4 8 5 2 1 3 7 6  
3 6 9 2 7 1 8 5 10 4  
8 16 9 2 12 6 3 15 14 1 5 11 10 4 13 7  
26 2 29 6 34 12 41 20 50 32 14 46 30 15 49 37 23 11 3 43 36 28 24 21 19 22 27 35 42 1 10 33 4 25 7 44 38  
31 40 5 18 16 39 9 17 45 48 13 8 47

**Note.** Verify your program on even more testcases from [here](#).

**Fun Video.** [The Josephus Problem – Numberphile](#)

## 8.2. Van Eck's Sequence

The Van Eck's Sequence is defined as follows:

- $a_0 = 0$  then for  $n > 0$ ,
- $a_{n+1} = \begin{cases} n - m & \text{where } m \text{ the maximal index } < n \text{ exists, such that } a_m = a_n \\ 0 & \text{if such } m < n \text{ doesn't exist, then we take } m = n \rightarrow a_{n+1} = 0. \end{cases}$

### Problem Statement:

Generate the first  $n + 1$  elements  $a_0, a_1, \dots, a_n$  of the Van Eck's Sequence.

#### Starter Code

##### Input Format

$n$

(a single integer)

##### Output Format

$a_0 \ a_1 \ \dots \ a_n$

(space seperated integers)

##### Constraints

$1 \leq n \leq 100000$

##### Sample Input

500

##### Sample Output

0 0 1 0 2 0 2 2 1 6 0 5 0 2 6 5 4 0 5 3 0 3 2 9 0 4 9 3 6 14 0 6 3 5 15 0 5 3 5 2 17 0 6 11 0 3 8 0 3 3 1 42 0  
5 15 20 0 4 32 0 3 11 18 0 4 7 0 3 7 3 2 31 0 6 31 3 6 3 2 8 33 0 9 56 0 3 8 7 19 0 5 37 0 3 8 8 1 46 0 6 23  
0 3 9 21 0 4 42 56 25 0 5 21 8 18 52 0 6 18 4 13 0 5 11 62 0 4 7 40 0 4 4 1 36 0 5 13 16 0 4 8 27 0 4 4 1  
13 10 0 6 32 92 0 4 9 51 0 4 4 1 14 131 0 6 14 4 7 39 0 6 6 1 12 0 5 39 8 36 44 0 6 10 34 0 4 19 97 0 4 4 1  
19 6 12 21 82 0 9 43 0 3 98 0 3 3 1 15 152 0 6 17 170 0 4 24 0 3 12 24 4 6 11 98 21 29 0 10 45 0 3 13 84 0  
4 14 70 0 4 4 1 34 58 0 6 23 144 0 4 9 51 94 0 5 78 0 3 26 0 3 3 1 21 38 0 6 21 4 19 76 0 6 6 1 12 56 166  
0 7 111 0 3 21 16 145 0 5 33 206 0 4 23 46 194 0 5 9 47 0 4 9 4 2 223 0 6 33 19 39 132 0 6 6 1 40 185 0 6  
5 23 28 0 5 4 22 0 4 3 46 36 151 0 6 15 126 0 4 10 110 0 4 4 1 29 118 0 6 14 112 0 4 9 51 102 0 5 33 50 0  
4 9 9 1 20 307 0 7 88 0 3 42 262 0 4 14 27 233 0 5 23 60 0 4 9 22 60 5 8 210 0 8 3 22 8 3 3 1 34 156 0 10  
63 0 3 8 11 183 0 5 22 17 199 0 5 5 1 19 109 0 6 73 0 3 19 7 58 183 20 64 0 8 26 174 0 4 52 319 0 4 4 1  
25 331 0 6 25 4 7 23 69 0 7 4 6 9 71 0 6 4 6 2 158 0 6 4 6 2 6 2 2 1 30 0 10 73 54 0 4 13 247 0 4 4 1 13 6  
18 367 0 8 59 0 3 70 257 0 4 14 123 0 4 4

##### More Test cases

[Input](#) and [Output](#) files

**Fun Video.** [Don't Know \(the Van Eck Sequence\) – Numberphile](#)



### 8.3. Look-And-Say Sequence

As the name suggests, the look-and-say sequence is generated by the *reading* of the digits of the previous sequence. For example, starting with the sequence **1**.

- **1** is read off as “one 1” or **11**.
- **11** is read off as “two 1s” or **21**.
- **21** is read off as “one 2, one 1” or **1211**.
- **1211** is read off as “one 1, one 2, two 1s” or **111221**.
- **111221** is read off as “three 1s, two 2s, one 1” or **312211** and so on.

#### Problem Statement:

Generate the first  $n$  iterations of the look-and-say sequence.

#### Starter Code

##### Input Format

$n$

(a single integer)

##### Output Format

First  $n$  iterations of the look-and-say sequence

(each iteration on a newline)

##### Constraints

$1 \leq n \leq 40$

##### Sample Input

15

##### Sample Output

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
11131221133112132113212221
3113112221232112111312211312113211
1321132132111213122112311311222113111221131221
11131221131211131231121113112221121321132132211331222113112211
311311222113111231131112132112311321322112111312211312111322212311322113212221
```

##### More Test cases

[Input](#) and [Output](#) files

**Fun Video.** [Look-and-Say Numbers \(feat John Conway\) – Numberphile](#)

8.4. Thue-Morse Sequence

Thue-Morse Sequence aka Fair Share Sequence is an infinite binary sequence obtained by starting with 0 and successively appending the Boolean complement of the sequence obtained thus far (called prefixes of the sequence).

For example, starting with the sequence 0,

- Append complement of 0, we get 01
- Append complement of 01, we get 0110
- Append complement of 0110, we get 01101001 and so on.

Also, by using Thue-Morse sequence elements in the turtle simulator, we get a mysterious curve<sup>4</sup> by following the below rule.

- If an element is 0, then the turtle rotates right by 180°.
- If an element is 1, then the turtle moves forward by one unit and then rotates right by 60°.

Can you figure out the pattern of this curve?

Problem Statement:

Generate the first  $n$  elements of the Thue-Morse sequence and draw the corresponding curve using turtleSim. Scale the curve in such a way that it roughly takes same width and height for all  $n$ .

Starter Code
Input Format <div><math>n</math></div> (a single integer)
Output Format <div>First <math>n</math> elements of the Thue-Morse sequence and the curve.</div>
Constraints <div><math>1 \leq n \leq 100000</math> (<math>n</math> need not be a power of 2)</div>
Sample Input <div>111</div>
Sample Output <div>01101001100101101001011001100110011010010110011010010110100110011010010110100110011010011001011</div>
More Test cases <div><a href="#">Input</a> and <a href="#">Output</a> files</div>

The output Koch Curve convergents

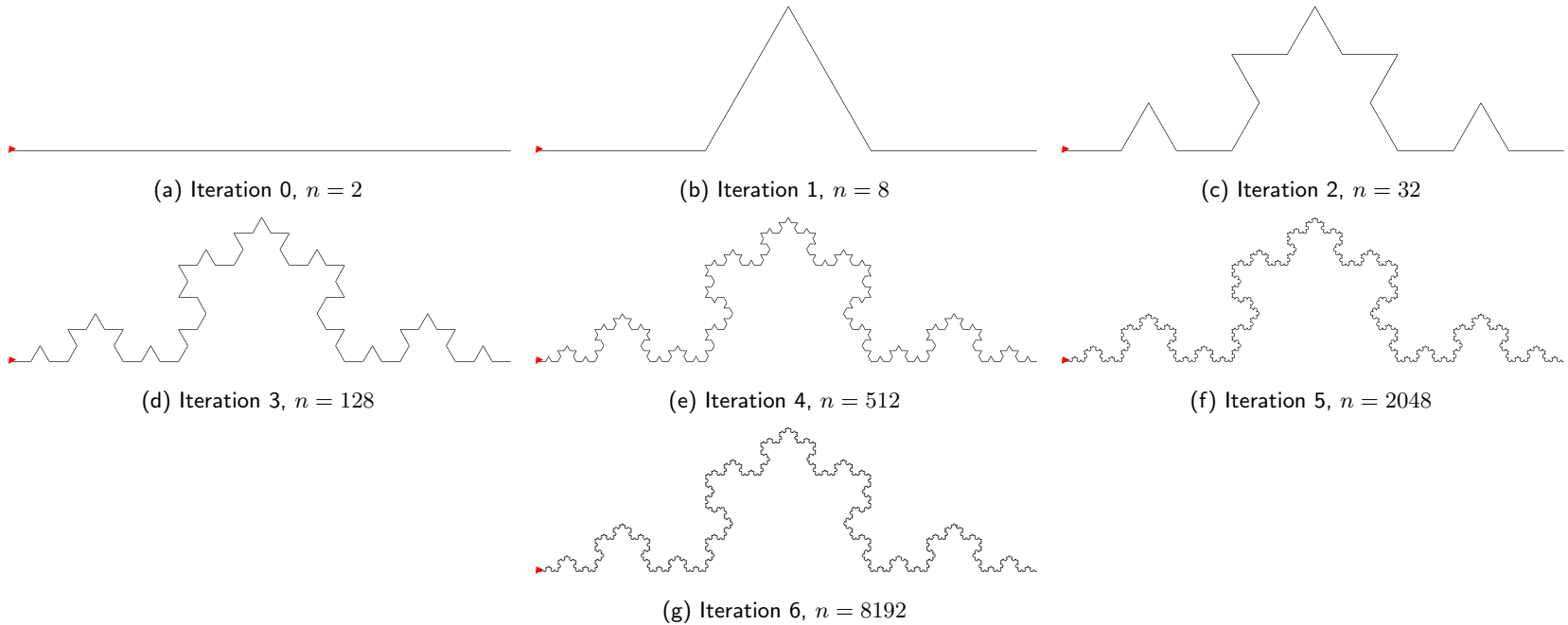


Figure 16: Koch Curve Iterations and the outputs for odd powers of 2

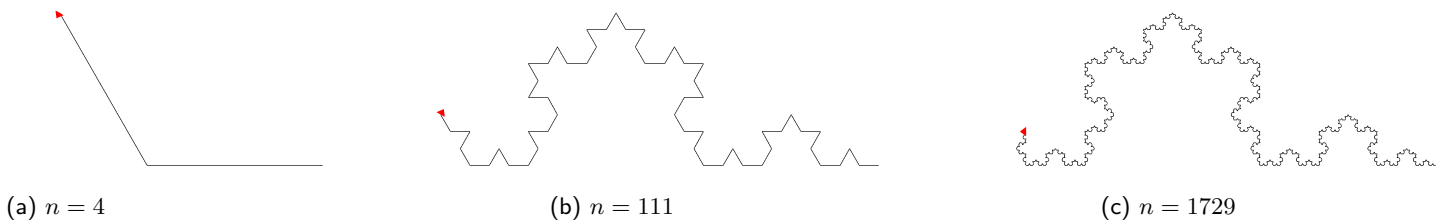


Figure 17: The outputs for numbers which are not a odd power of 2

Fun Video. [The Fairest Sharing Sequence Ever – Stand-up Maths](#)  
[Fractal charm: Space filling curves – 3Blue1Brown](#)

<sup>4</sup>called Koch curve, it is a fractal curve that has infinite length but contained in a finite area. Can you see why?

8.5. Recaman's Sequence

The Recaman's sequence is defined as below:

- $r_0 = 0$
- $r_n = \begin{cases} r_{n-1} - n & \text{if } r_{n-1} - n > 0 \text{ and } \forall i < n, r_i \neq r_{n-1} - n, \text{ i.e. } r_{n-1} - n \text{ is positive and has not yet occurred in the sequence} \\ r_{n-1} + n & \text{otherwise} \end{cases}$

Also, by using Recaman's sequence elements in the turtle simulator, we can get beautiful curves as shown in 18 by following the below rules:

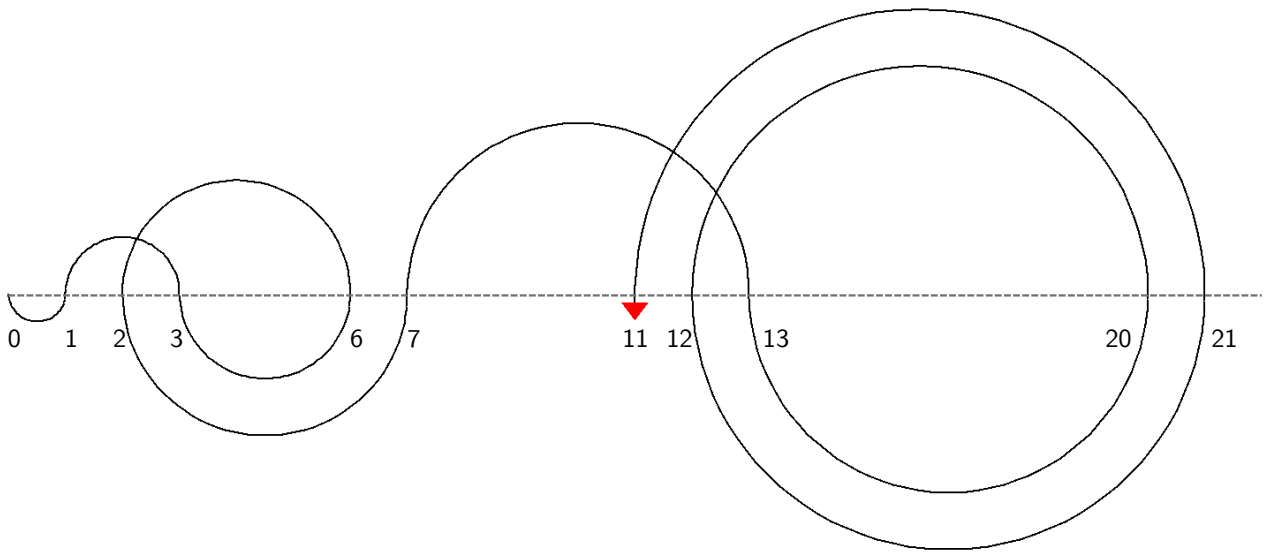


Figure 18: Recaman's Sequence Drawing Procedure

- Create a canvas named "Recamans Sequence" with width=1920, and height=1080.
- Connect all consecutive terms using semicircles.
- The semicircles should be parallel to  $x$ -axis with end points as consecutive terms
- The semicircles should alternate above and below the  $x$ -axis; i.e., it should be below the axis when connecting  $r_0, r_1$ , above the axis when connecting  $r_1, r_2$ , again below for  $r_2, r_3$ , and so on.
- The figure should be dynamic; i.e., the  $x$ -axis should be such that for any  $n$  the figure takes up at least half the canvas and it also remains within the canvas.
- Don't draw the numbers and the axis. They are just to visualise the construction.

Problem Statement:

Generate the first  $n + 1$  elements  $r_0, r_1, \dots, r_n$  of the Recaman's Sequence and draw the corresponding curve using turtleSim.

Starter Code
Input Format <div><math>n</math></div> <div>(a single integer)</div>
Output Format <div>First <math>n + 1</math> elements of the Recaman's sequence and the curve.</div>
Constraints <div><math>1 \leq n \leq 1000</math></div>
Sample Input <div>60</div>
Sample Output <div>0 1 3 6 2 7 13 20 12 21 11 22 10 23 9 24 8 25 43 62 42 63 41 18 42 17 43 16 44 15 45 14 46 79 113 78 114 77 39 78 38 79 37 80 36 81 35 82 34 83 33 84 32 85 31 86 30 87 29 88 28</div>
More Test cases <div><a href="#">Input</a> and <a href="#">Output</a> files</div>

The output curve

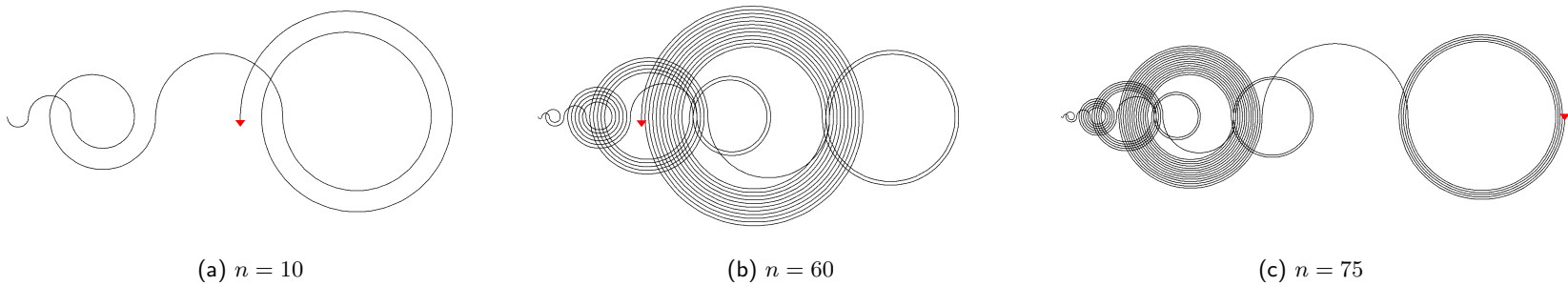


Figure 19: Output Ford Circles for few  $n$

Fun Video. [The Slightly Spooky Recamán Sequence – Numberphile](#)

8.6. Farey Sequence

Farey sequence has all rational numbers in range  $[0/1 \text{ to } 1/1]$  sorted *in increasing order* such that the denominators are less than or equal to  $n$  and all numbers are in *reduced forms* i.e.,  $2/4$  does not belong to this sequence as it can be reduced to  $1/2$ . For example,  $n = 4$ , the possible rational numbers in increasing order are  $0/1, 1/4, 1/3, 1/2, 2/3, 3/4, 1/1$ .

Stern-Brocot Tree

To generate the Farey Sequence, we have to first look at the Stern-Brocot Tree shown in 20.

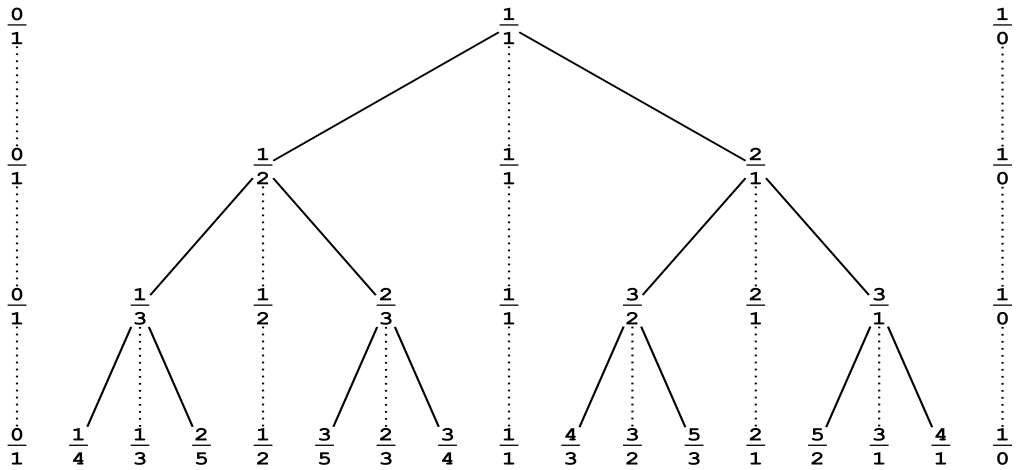


Figure 20: The Stern-Brocot Tree for Level 1 – 4 (Image by Aaron Rotenberg licensed under CC BY-SA 3.0)

In this tree, a child is given by the **mediant** of their parents; i.e, for child of parents  $\frac{a}{c}$  and  $\frac{b}{d}$  is  $\frac{a+b}{c+d}$ .

Some examples for parent, child are as follows –  $\left(\frac{0}{1}, \frac{1}{1} \rightarrow \frac{1}{2}\right), \left(\frac{1}{1}, \frac{1}{0} \rightarrow \frac{2}{1}\right), \left(\frac{0}{1}, \frac{1}{2} \rightarrow \frac{1}{3}\right), \left(\frac{1}{2}, \frac{1}{1} \rightarrow \frac{2}{3}\right), \left(\frac{1}{1}, \frac{2}{1} \rightarrow \frac{3}{2}\right), \left(\frac{2}{1}, \frac{1}{0} \rightarrow \frac{3}{1}\right),$

Notice that the farey sequence for corresponding  $n$  is the subset of vertices of this tree calculated upto level  $n$ .

Also, for every fraction  $\frac{p}{q}$  in the farey sequence draw a circle with centre at  $\left(\frac{p}{q}, \frac{1}{2q^2}\right)$  and radius  $\left(\frac{1}{2q^2}\right)$ . You may need to do some scaling to get a proper figure.

Problem Statement:

Generate the Farey Sequence for corresponding  $n$  using ideas from the Stern-Brocot Tree or otherwise and draw the circles.

Hint. Recursion!

<b>Starter Code</b>	
<b>Input Format</b> $n$	(a single integer)
<b>Output Format</b> Corresponding numbers in farey sequence in $p/q$ format with the circles.	
<b>Constraints</b> $1 \leq n \leq 30$	(an integer)
<b>Sample Input</b> 7	
<b>Sample Output</b> 0/1 1/7 1/6 1/5 1/4 2/7 1/3 2/5 3/7 1/2 4/7 3/5 2/3 5/7 3/4 4/5 5/6 6/7 1/1	
<b>More Test cases</b> <a href="#">Input</a> and <a href="#">Output</a> files	

The output circles (Ford Circles)

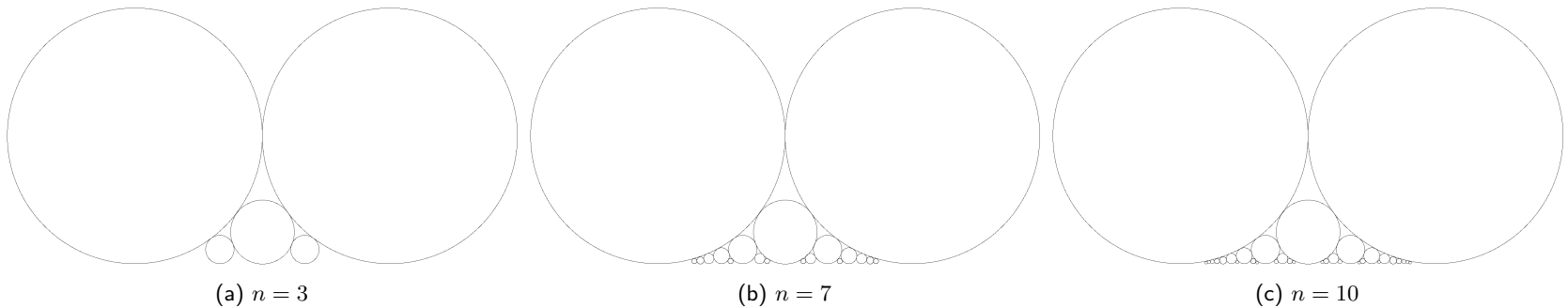


Figure 21: Output Ford Circles for few  $n$

**Interesting Observation.** If the outputs take a long time then how can you make it faster?. Also, try calculating terms mathematically to get the fastest way!

**Fun Video.** [Infinite Fractions – Numberphile](#)  
[Funny Fractions and Ford Circles – Numberphile](#)

# §9. Array Leeway (2-D Arrays)

Topics. 2-D arrays, function & arrays and previous sections.

## 9.1. Case Converter

**Problem Statement:**  
Convert a given text into different cases as mentioned below

**aLtErNaTiNg CaPs** Start with a lower case letter and then keep switching between upper case and lower case letters alternately.

**Capitalize Word** Capitalize the first letter of each word and convert all other letters of that word to lower case.

**lower case** Convert every alphabet to lower case.

**Sentence case** Capitalize the first letter of each sentence and convert all other letters of that sentence to lower case. A sentence only ends with a full stop ('.') .

**tOGGLE cASE** Uncapitalize the first letter of each word and convert all other letters of that word to upper case.

**UPPER CASE** Convert every alphabet to upper case.

**Note.** In all above cases, ignore non-alphabetic characters.

<b>Starter Code</b>	
<b>Input Format</b> sentence_length x sentence	(x is either a/c/l/s/t/u denoting the case to convert to or e for all cases) (entire sentence in a line, the sentence_length includes spaces)
<b>Output Format</b> The sentence converted into x case	(already taken care of in <a href="#">Starter Code</a> )
<b>Constraints</b> 1 ≤ sentence_length ≤ 10000	
<b>Sample Input</b> 479 e The Earth is a very small stage in a vast cosmic arena. Think of the endless cruelties visited by the inhabitants of one corner of this pixel on the scarcely distinguishable inhabitants of some other corner, how frequent their misunderstandings, how eager they are to kill one another, how fervent their hatreds. Think of the rivers of blood spilled by all those generals and emperors so that, in glory and triumph, they could become the momentary masters of a fraction of a dot.	
<b>Sample Output</b> tHe EaRtH iS a VeRy sMaIl sTaGe In A vAsT cOsMiC aReNa. ThInK oF tHe eNDIEs cRuEITiEs ViSiTeD bY tHe InHaBiTaNtS oF oNe CoRnEr Of ThIs PiXeL oN tHe ScArCeLy DiStInGuIsHaBiE iNhAbItAnTs Of SoMe OtHeR cOrNeR, hOw FrEqUeNt ThEiR mIsUnDeRsTaNdInGs, HoW eAgEr ThEy ArE tO kIlL oNe AnOtHeR, hOw FeRvEnT tHeIr HaTrEdS. tHiNk Of ThE rIvErS oF bLoOd SpIlLeD bY aLl ThOsE gEnErAiS aNd EmPeRoRs So ThAt, In GlORy aNd TrLuMpH, tHeY cOuLd BeCoMe ThE mOmEnTaRy MaStErS oF a FrAcTiOn Of A dOt.  The Earth Is A Very Small Stage In A Vast Cosmic Arena. Think Of The Endless Cruelties Visited By The Inhabitants Of One Corner Of This Pixel On The Scarcely Distinguishable Inhabitants Of Some Other Corner, How Frequent Their Misunderstandings, How Eager They Are To Kill One Another, How Fervent Their Hatreds. Think Of The Rivers Of Blood Spilled By All Those Generals And Emperors So That, In Glory And Triumph, They Could Become The Momentary Masters Of A Fraction Of A Dot.  the earth is a very small stage in a vast cosmic arena. think of the endless cruelties visited by the inhabitants of one corner of this pixel on the scarcely distinguishable inhabitants of some other corner, how frequent their misunderstandings, how eager they are to kill one another, how fervent their hatreds. think of the rivers of blood spilled by all those generals and emperors so that, in glory and triumph, they could become the momentary masters of a fraction of a dot.  The earth is a very small stage in a vast cosmic arena. Think of the endless cruelties visited by the inhabitants of one corner of this pixel on the scarcely distinguishable inhabitants of some other corner, how frequent their misunderstandings, how eager they are to kill one another, how fervent their hatreds. Think of the rivers of blood spilled by all those generals and emperors so that, in glory and triumph, they could become the momentary masters of a fraction of a dot.  tHE eARTH iS a VeRY sMaIl sTaGe In a vAsT cOsMiC aReNa. tHiNk oF tHe eNDLEsS cRuELTIeS vISITeD bY tHe iNHABITANTS oF oNe cORNER oF tHiS pIXEL oN tHe sCARCELY DISTINGUISHABLE iNHABITANTS oF sOME oTHER cORNER, hOW fREQUeNT tHEIR mISUNDERSTANDINGS, hOW eAGER tHEY aRE tO kILL oNe aNOTHER, hOW fERVENT tHEIR hATREDS. tHiNk oF tHe rIvERS oF bLOOD sPILLED bY aLL tHOSE gENERALS aNd eMPERORS sO tHAT, iN gLORY aNd tRIUMPH, tHEY cOULD bECOME tHE mOMENTARY mASTERS oF a fRACTION oF a dOT.  THE EARTH IS A VERY SMALL STAGE IN A VAST COSMIC ARENA. THINK OF THE ENDLESS CRUELTIES VISITED BY THE INHABITANTS OF ONE CORNER OF THIS PIXEL ON THE SCARCELY DISTINGUISHABLE INHABITANTS OF SOME OTHER CORNER, HOW FREQUeNT THEIR MISUNDERSTANDINGS, HOW EAGER THEY ARE TO KILL ONE ANOTHER, HOW FERVENT THEIR HATREDS. THINK OF THE RIVERS OF bLOOD sPILLED bY aLL tHOSE gENERALS aNd eMPERORS sO tHAT, iN gLORY aNd tRIUMPH, tHEY cOULD bECOME tHE mOMENTARY mASTERS oF a fRACTION oF a dOT.	

## 9.2. Spiral Grid

**Problem Statement:**

Generate a grid containing numbers from 1 to  $n^2$  such that 1 is at center and then the numbers spiral outwards from 1 in counterclockwise direction. Also, make sure each element of grid is equally spaced as shown in 22.

**Note.** If  $n$  is even then choose the left-bottom element from the four possible centers.

<b>Starter Code</b>
<div><div><b>Input Format</b></div><div><math>t</math> <math>n_1\ n_2\ \dots\ n_t</math></div></div> <div><div>(number of test cases, an integer)</div><div>(<math>t</math> space seperated integers for each testcase)</div></div>
<div><div><b>Output Format</b></div><div>Required spiral grid of <math>n_i^2</math> numbers with appropriate spacing</div></div>
<div><div><b>Constraints</b></div><div><math>1 \leq n_i \leq 100</math></div></div>
<div><div><b>Sample Input</b></div><div>5 1 2 3 6 15</div></div>
<div><div><b>Sample Output</b></div><div>1  4 3 1 2  5 4 3 6 1 2 7 8 9  36 35 34 33 32 31 17 16 15 14 13 30 18 5 4 3 12 29 19 6 1 2 11 28 20 7 8 9 10 27 21 22 23 24 25 26  197 196 195 194 193 192 191 190 189 188 187 186 185 184 183 198 145 144 143 142 141 140 139 138 137 136 135 134 133 182 199 146 101 100 99 98 97 96 95 94 93 92 91 132 181 200 147 102 65 64 63 62 61 60 59 58 57 90 131 180 201 148 103 66 37 36 35 34 33 32 31 56 89 130 179 202 149 104 67 38 17 16 15 14 13 30 55 88 129 178 203 150 105 68 39 18 5 4 3 12 29 54 87 128 177 204 151 106 69 40 19 6 1 2 11 28 53 86 127 176 205 152 107 70 41 20 7 8 9 10 27 52 85 126 175 206 153 108 71 42 21 22 23 24 25 26 51 84 125 174 207 154 109 72 43 44 45 46 47 48 49 50 83 124 173 208 155 110 73 74 75 76 77 78 79 80 81 82 123 172 209 156 111 112 113 114 115 116 117 118 119 120 121 122 171 210 157 158 159 160 161 162 163 164 165 166 167 168 169 170 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225</div></div>
<div>Figure 22: Sample Output</div>
<div><div><b>More Test cases</b></div><div><a href="#">Input</a> and <a href="#">Output</a> files</div></div>

### 9.3. Minesweeper

In the game of Minesweeper, there is an  $m \times n$  board which has exactly  $k$  mines hidden. The aim is to “clear” the board by clicking on cells with no mine and avoiding clicking on any mine. By clicking on a cell with no mine, the player gets the number of neighbouring mines of that cell by the below rule

- If the cell  $c$  is not at the boundary (23a) then it is the number of mines in a  $3 \times 3$  square with that centre  $c$ .
- If the cell  $c$  is at the boundary (23b, 23b) even then  $c$  cell is considered as the centre of  $3 \times 3$  square; but, only some of the cells of the constructed square will lie inside the board.

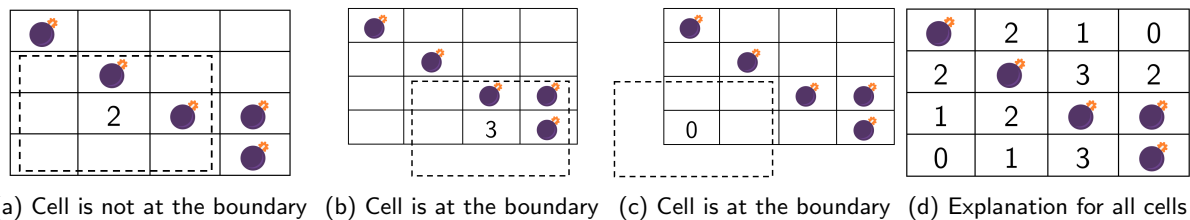


Figure 23: Minesweeper – Explanation

#### Problem Statement:

Calculate the neighbour count for all cells except at the mines where you have to output the character 'M'.

**Starter Code**

**Input Format**  
 $m\ n$  (space separated integer pair corresponding to number of rows and columns)  
 $k\ x_1\ y_1\ \dots\ x_k\ y_k$  ( $2k + 1$  space separated integers corresponding to number of mines and the  $x, y$  co-ordinates of all mines (1-indexed))

**Output Format**  
 $m \times n$  matrix  $A$ , where  $a_{ij} = \begin{cases} \text{'M'} & \text{if there is a mine at } (i, j) \\ \text{number of neighbouring mines of the cell } (i, j) & \text{otherwise} \end{cases}$

**Constraints**  
 $1 \leq m, n \leq 50, 1 \leq k \leq m \times n$   
 $1 \leq x \leq m, 1 \leq y \leq n$   
 $0 \leq a_{ij} \leq 8$  or  $a_{ij} = \text{'M'}$ .

**Sample Input**  
4 4  
5 1 1 2 2 3 3 3 4 4 4

**Sample Output**  
M 2 1 0  
2 M 3 2  
1 2 M M  
0 1 3 M

**More Test cases**  
[Input](#) and [Output](#) files

**Note.** Try implementing the complete minesweeper game :)

### 9.4. Gray Code

A gray code is a rearrangement of binary numbers such that any 2 consecutive numbers differ only in 1 bit.

A simple way to generate n-bit gray code is given below

- Start with an array of 2 numbers  $A = \{0, 1\}$
- Repeat the below steps  $n - 1$  times
  - Reverse the array  $A$  to get array  $A'$  and then append  $A'$  to  $A$ .
  - Append 0 to the left of the first half elements of  $A$  and  
Append 1 to the left of the second half elements of  $A$ .

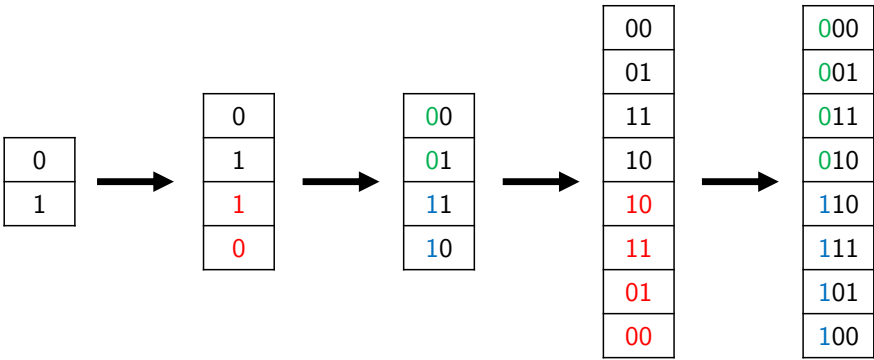


Figure 24: Gray Code – Generation

**Problem Statement:**

For a given  $n$ , generate its corresponding Gray Code (i.e. first  $2^n$  elements).

<b>Starter Code</b>	
<b>Input Format</b> $n$	(integer)
<b>Output Format</b> $2^n$ numbers denoting Gray Code	
<b>Constraints</b> $1 \leq n \leq 10$	
<b>Sample Input</b> 3	
<b>Sample Output</b> 000 001 011 010 110 111 101 100	
<b>More Test cases</b> <a href="#">Input</a> and <a href="#">Output</a> files	



## §10. Array Powerplay (More Arrays or Recursion?)

**Topics.** *Recursion & arrays and previous sections.*

### 10.1. Determinant of a Matrix

For a matrix  $A \in \mathbb{Z}^{n \times n}$  (zero-based indexing) and for a  $i \in \{0, 1, \dots, n-1\}$ , the determinant of  $A$  ( $\det(A)$ ) is

$$\det(A) = \sum_{j=0}^{n-1} (-1)^{i+j} a_{ij} M_{ij} \quad M_{ij} \text{ is the det of the matrix obtained by removing the } i^{\text{th}} \text{ row and } j^{\text{th}} \text{ column of } A. \quad (28)$$

**Problem Statement:**

Find the determinant of given matrix  $A$  using the above formula (called as Laplace Expansion).

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_i$  (size of matrix  $A$ )  
 $a_{0,0} \ a_{0,1} \ \dots \ a_{0,n_i-1}$  (0<sup>th</sup> row of matrix)  
 $a_{1,0} \ a_{1,1} \ \dots \ a_{1,n_i-1}$  (1<sup>th</sup> row of matrix)  
 $\dots$   
 $a_{n_i-1,0} \ a_{n_i-1,1} \ \dots \ a_{n_i-1,n_i-1}$  ( $(n_i - 1)^{\text{th}}$  row of matrix)

##### Output Format

$\det(A)$  (space seperated integers for each test case)

##### Constraints

$1 \leq n_i \leq 10, -1000 \leq a_{i,j} \leq 1000$  (integers)

##### Sample Input

```
5
1
7
2
4 3
1 2
3
5 4 3
6 1 2
7 8 9
4
1 1 1 1
1 3 9 27
1 6 36 216
1 10 100 1000
5
4 -1 -1 -1 -1
-1 2 -1 0 0
-1 -1 2 0 0
-1 0 0 2 -1
-1 0 0 -1 3
```

##### Sample Output

```
7 5 -72 7560 9
```

**Fun Video.** [The Vandermonde Matrix and Polynomial Interpolation – Dr. Will Wood](#)

10.2. Tower of Hanoi

Tower of Hanoi is a mathematical puzzle with three rods (A,B,C) and n disks on left rod (A) with in decreasing order of their radius from top to bottom .

The objective of the puzzle is to move the entire stack of disks to the rightmost rod (C), obeying the following simple rules,

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack; i.e., a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Check any of the linked videos below for more information. **Problem Statement:**

For a given *n*, output the sequence of steps to be taken in the following format:

Disk <disk-number> from <rod-name> to <rod-name>.

Solve the problem **with recursion** and **without recursion** as well :).

Starter Code	
Input Format	
<i>t</i>	(number of test cases, an integer)
<i>n</i> <sub>1</sub> <i>n</i> <sub>2</sub> ... <i>n</i> <sub><i>t</i></sub>	( <i>t</i> numbers)
Output Format	
Corresponding steps till completetion	(each step on a new line for each test case)
Constraints	
1 ≤ <i>n</i> <sub><i>i</i></sub> ≤ 20	(integers)
Sample Input	
4	
1 2 3 4	
Sample Output	
Disk 1 from A to B	
 Disk 1 from A to C Disk 2 from A to B Disk 1 from C to B	
 Disk 1 from A to B Disk 2 from A to C Disk 1 from B to C Disk 3 from A to B Disk 1 from C to A Disk 2 from C to B Disk 1 from A to B	
 Disk 1 from A to C Disk 2 from A to B Disk 1 from C to B Disk 3 from A to C Disk 1 from B to A Disk 2 from B to C Disk 1 from A to C Disk 4 from A to B Disk 1 from C to B Disk 2 from C to A Disk 1 from B to A Disk 3 from C to B Disk 1 from A to C Disk 2 from A to B Disk 1 from C to B	
More Test cases	
<a href="#">Input</a> and <a href="#">Output</a> files	

Fun Video. *Binary, Hanoi and Sierpinski*, [part 1](#) [part 2](#) – 3Blue1Brown  
[Towers of Hanoi: A Complete Recursive Visualization – Reducible](#)

10.3. Quicksort

Quicksort is a divide and conquer algorithm like merge sort discussed in class. It first divides the input array into two smaller sub-arrays: the low elements and the high elements. It then recursively sorts the sub-arrays. Precisely,

- Pick an element, called a pivot, from the array.
- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this *partitioning*, the pivot is in its final position (relative to other elements).
- Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values. The base case of the recursion are arrays of size zero or one, which are in order by definition, so they never need to be sorted.

**Note.** The pivot selection and partitioning steps can be done in several different ways; the algorithm's performance greatly varies with implementation schemes.

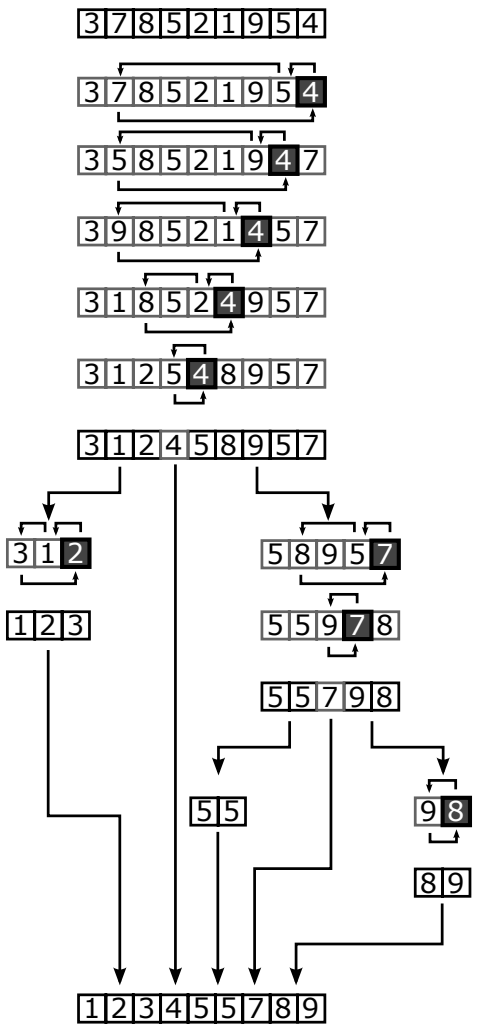


Figure 25: Quicksort Runthrough (Image by Znuipi, Public domain, via Wikimedia Commons)

Problem Statement:

Sort the given array using Quicksort. Use Lomuto partition scheme, i.e. take the last element of array as pivot.

**Note.** You are not provided with the size of array. Learn the way to do it! If you give up then go through procedure in starter code and understand it thoroughly.

Starter Code	
Input Format	
t	(number of test cases, an integer)
a <sub>0</sub> a <sub>1</sub> a <sub>2</sub> ... a <sub>n<sub>i</sub>-1</sub>	(n <sub>i</sub> space seperated integers for each testcase)
Output Format	
Sorted Array	(space seperated elements for each test case)
Constraints	
1 ≤ n <sub>i</sub> ≤ 1000, −100000 ≤ a <sub>i</sub> ≤ 100000	(integers)
Sample Input	
4	
1 7 5 2 3 10 4 6 9 8	
86 56 24 26 55 73 77 100 53 20 52 59 74 43 19 21 74 51 44 79 76 15 54 62 6 43 42 5 28 84	
17 9 10 6 6 12 5 16 18 1 14 11 6 12 14 12 13 10 12 3 2 16 16 14 11 12 7	
59 18 -85 99 87 -90 -17 -83 -28 -19 -39 46 -27 -20 53 48 -11 -42 5 85 -49 78 86 -42 -33 -56 -41 21 -62 95 -59 -63 50 57 78 -8 14 -35 -5 7 4 -45 -17 -10 -23	
Sample Output	
1 2 3 4 5 6 7 8 9 10	
5 6 15 19 20 21 24 26 28 42 43 43 44 51 52 53 54 55 56 59 62 73 74 74 76 77 79 84 86 100	
1 2 3 5 6 6 6 7 9 10 10 11 11 12 12 12 12 12 13 14 14 14 16 16 16 17 18	
-90 -85 -83 -63 -62 -59 -56 -49 -45 -42 -42 -41 -39 -35 -33 -28 -27 -23 -20 -19 -17 -17 -11 -10 -8 -5 4 5 7 14 18 21 46 48 50 53 57 59 78 78 85 86 87 95 99	

Fun Video. What's the fastest way to alphabetize your bookshelf? - Chand John – TED-Ed

## §11. Programming Expositions

**Topics.** All previous sections.

### 11.1. Newton Interpolation

For a given sequence of numbers  $\{a_0, \dots, a_{n-1}\}$ , we define  $\Delta^k$  inductively as follows

- $\Delta^0 = \{a_0, \dots, a_{n-1}\}$
- If  $\Delta^i = \{b_0, b_1, \dots, b_{n-i-2}, b_{n-i-1}\}$  then  $\Delta^{i+1} = \{b_1 - b_0, \dots, b_{n-i-1} - b_{n-i-2}\}$ ; i.e., difference of successive terms gives the next sequence. Also, we treat  $\Delta^k$  as an array with  $i^{\text{th}}$  index as  $\Delta^k[i]$ .

Notice, that number of terms reduces by 1 after each iteration. Hence  $\Delta^{n-1}$  has only 1 term and we stop.

Now, using these  $\Delta^i$ 's, we can construct a polynomial  $f$  such that  $f(i) = a_i$  for  $i = \{0, 1, \dots, n-1\}$ . This process is called *interpolation* and the formula for  $f$  is given below.

$$f(x) = \sum_{k=0}^{n-1} \binom{x}{k} \Delta^k[0] = \sum_{k=0}^{n-1} \frac{(x)_k}{k!} \Delta^k[0] \quad \text{where } (x)_0 = 1 \text{ and } (x)_k = x(x-1) \cdots (x-(k-1)) \quad (29)$$

An example from [wikipedia](#),

$x$	$f = \Delta^0$	$\Delta^1$	$\Delta^2$	$f(x) = \Delta^0 \cdot \frac{(x)_0}{0!} + \Delta^1 \cdot \frac{(x)_1}{1!} + \Delta^2 \cdot \frac{(x)_2}{2!}$	
0	<u>2</u>				
1	2	<u>0</u>			
2	4	2	<u>2</u>		

$$= 2 \cdot 1 + 0 \cdot \frac{x}{1} + 2 \cdot \frac{(x)(x-1)}{2} \quad (30)$$

$$= 2 + (x)(x-1)$$

#### Problem Statement:

For a given sequence  $\{a_0, \dots, a_{n-1}\}$ , find its interpolated polynomial and predict the next term  $a_n = f(n)$ .

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
 $n_i \quad a_0 \quad a_1 \quad \dots \quad a_{n_i-1}$  ( $n_i + 1$  space separated integers for each testcase)

##### Output Format

$f(n_i)$ , followed by expansion of  $f(x) \sum_{k=0}^{n-1} \frac{x_k}{k!} \Delta^k[0]$ , ignoring  $\Delta^i[0] = 0$  terms and  $\Delta^i[0] = \pm 1$  coefficients.

##### Constraints

$1 \leq n_i \leq 20, -1000 \leq a_i \leq 1000$

##### Sample Input

```
4
3 2 2 4
4 1 2 3 4
7 3 1 4 1 5 9 2
9 1 2 4 8 16 31 57 99 163
```

##### Sample Output

```
8 2(x)_0/0! + 2(x)_2/2!
5 (x)_0/0! + (x)_1/1!
45 3(x)_0/0! - 2(x)_1/1! + 5(x)_2/2! - 11(x)_3/3! + 24(x)_4/4! - 44(x)_5/5! + 60(x)_6/6!
256 (x)_0/0! + (x)_1/1! + (x)_2/2! + (x)_3/3! + (x)_4/4!
```

**Fun Video.** [Why don't they teach Newton's calculus of 'What comes next?' – Mathologer](#)

## 11.2. ISBN

You may have wondered about the 10 (or 13) digits numbers on the back of every book. They are ISBN, which stands for International Standard Book Number and is used for uniquely identifying books and other publications (including e-publications). Go find the ISBN of your favourite book! :)

Let us consider ISBN 10 (10 digit numbers), an old format that got replaced by ISBN 13. The first 9 digits contain information about the geographical region, publisher and edition of the title. The last digit is a check digit used for validating the number. Let the number be  $x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}$ , then the check digit  $x_{10}$  is chosen such that the checksum  $= 10x_1 + 9x_2 + 8x_3 + 7x_4 + 6x_5 + 5x_6 + 4x_7 + 3x_8 + 2x_9 + 1x_{10}$  is a multiple of 11. This condition is succinctly represented as below:

$$\left( \sum_{i=1}^{10} (11-i)x_i \right) \% 11 = 0 \quad (31)$$

### Generation of check digit (example)

If the first nine digits are 812913572 then  $8 \times 10 + 1 \times 9 + 2 \times 8 + 9 \times 7 + 1 \times 6 + 3 \times 5 + 5 \times 4 + 7 \times 3 + 2 \times 2 = 234$ . So if  $x_{10} = 8$ , then the checksum is divisible by 11. Hence, the ISBN is 8129135728.

**Note.** It is possible that the calculated check digit is 10 as we can get any remainder from 0 to 10 when divided by 11. But when the remainder is 10, as is not a single digit, appending 10 to ISBN will make its length 11. To avoid such cases, the letter 'X' is used to denote check digit = 10.

### Problem Statement:

Recover and output the missing digit from a given valid ISBN 10 code with a digit erased.

The missing digit can be any  $x_i$  ( $1 \leq i \leq 10$ ).

#### Starter Code

##### Input Format

$t$  (number of test cases, an integer)  
10 characters each either representing a digit (0-9) or a missing number ('?'). (for each testcase)  
The last character (check digit) can also be 'X'.

##### Output Format

A single digit, that is to be placed at '?' position to make the given ISBN valid. (space separated)  
If the missing integer is 10 then, the output should be 'X'

##### Constraints

It is always possible that a unique ISBN exists. (Why?)

##### Sample Input

9  
81291?5728  
30303935?7  
366205414?  
366?054140  
05?0764845  
?590764845  
?43935806X  
933290152?  
9332?0152X

##### Sample Output

3 7 0 2 9 0 0 X 9

Fun Video. [11.11.11 – Numberphile](#)

11.3. Vigenère Cipher

Vigenère Cipher is an cryptographic technique used for encryption and decryption of alphabetic texts. The process is done letter-by-letter.

Encryption

- Generate a message (also called as a plaintext) and an empty ciphertext.
- Select a key; i.e., a string of alphabets. Keep repeating the key until it is as long as message.
- Iterate through the message and key simulataneously, to get current message alphabet and key alphabet.
- Now from the table 26, insert into ciphertext the alphabet corresponding to message alphabet as row and key alphabet as column.

Decryption

- In this case, key and ciphertext are known.
- Iterate through the key and ciphertext simulataneously, to get current key letter and ciphertext letter.
- Now from the table 26, find the ciphertext letter in the column corresponding to the current key letter. The row of ciphertext letter gives plaintext letter.

**Note.** The encryption and decryption hinges on the fact that the key is kept secret and known only to people encrypting and decrypting the messages. With the knowledge of key, decryption is “easy” but without key it is “hard”.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 26: The Vigenere square (Image by Matt Crypto, Public domain, via Wikimedia Commons)

Problem Statement:

Implement encryption and decryption function as stated above:

Starter Code	
-----	
Input Format	
$t$	(number of test cases, an integer)
$k_i\ p_i$	(size of key and plaintext for each testcase)
$K_0 \dots K_{k_i}$	(key)
$P_0 \dots P_{p_i}$	(plaintext)
-----	
Output Format	
Ciphertext	(Verify decryption manually for decrypting the generated ciphertext and comparing with plaintext)
-----	
Constraints	
$1 \leq k_i, p_i \leq 1000$ , $K_i, P_i$ are CAPITAL LETTERS.	
-----	
Sample Input	
6	
5 12	
LEMON	
ATTACKATDAWN	
16 39	
LEOPOLDKRONECKER	
GODMADETHEINTEGERSALLELSEISTHEWORKOFMAN	

17 136  
STUARTMILNERBARRY  
IDONOTIMAGINETHATANYWARSINCECLASSICALTIMESIFEVERHASBEENFOUGHTINWHICHONESIDEREADCONSISTENTLYTHEMAINMILITARYAND  
NAVALINTELLIGENCEOFTHEOTHER

15 201  
BERTRANDRUSSELL  
MATHEMATICSPOSSESSESNOTONLYTRUTHBUTASUPREMEBEAUTYCOLDANDAUSTERELIKETHATOFSCULPTUREWITHOUTTHEGORGEOUSTRAP  
PINGSOFPAININGORMUSICYETSUBLIMELYPUREANDCAPABLEOFASTERNPERFECTIONSUCHASONLYTHEGREATESTARTCANSHOW

15 382  
BERTRANDRUSSELL  
IOUGHTTOCALLMYSELFANAGNOSTICBUTFORALLPRACTICALPURPOSESIAMANATHEISTIDONOTTHINKTHEEXISTENCEOFTHECHRISTIANGODAN  
YMOREPROBABLETHANTHEEXISTENCEOFTHEGODSOFOLYMPUSORVALHALLATOTAKEANOTHERILLUSTRATIONNOBODYCANPROVETHATTHE  
EREISNOTBETWEENTHEEARTHANDMARSACHINATEAPOTREVOLVINGINANELLIPTICALORBITBUTNOBODYTHINKSTHISSUFFICIENTLYLIKELYTO  
BETAKENINTOACCOUNTINPRACTICEITHINKTHECHRISTIANGODJUSTASUNLIKELY

9 389  
CARLSAGAN  
THEEARTHISAVERYSMALLSTAGEINAVASTCOSMICARENATHINKOFTHEENDLESSCRUELTIESVISITEDBYTHEINHABITANTSOFOFONECORNEROFTHIS  
PIXELONTHESCARCELYDISTINGUISHABLEINHABITANTSOFSOMEOTHERCORNERHOWFREQUENTTHEIRMISSUNDERSTANDINGSHOWEAGERTHEY  
ARETOKILLONEANOTHERHOWFERVENTTHEIRHATREDSTHINKOFTHERIVERSOFBLOODSPILLEDDBYALLTHOSEGENERALSANDEMPERORSSOTHAT  
TINGLORYANDTRIUMPHTHEYCOULDBECOMETHEMOMENTARYMASTERSOFAFRACTIONOFADOT

Sample Output  
LXFOPVEFRNHR

RSRBOOHDYSVRVOKVCWOAZPOCVWFXJOAFCOCUALQ

AWINFMUULTMEFTYRRSGSWRKEQYPITMAJJGUTFTZFQATSIMFRYRQTXYNWHGOSGMEXHZTFGGYSZWQZPNHTPNJZQLXHTCRFPPEZOMZCG  
LTLYRGPVLIECJNKVJDBAEVQWQGLVPTYVP

NEKAVMNWZWKHSDDFWJXJNBWFHDQXCFULSNKAFXGLWEIMPBYKRTOYGRHVSYDEFVVEZKRWYULGJDNVPGMLRRZZNZGYEEIIXHIGRRLMLJ  
EAAJRXLFFCDZHLARRZSPLLZCLHKMMTPTXFPILRRDEXUSTLMMIFYRSGHIIHHWVPDXZHESHFYUKGRWJULVZIENWVMLSVENBRJAFW

JSLZYTGRTUDDQJDFPWTEATQFMLAGMFUJFKRLYSIUULMNLMTLKGOFHJCSEEYLULVBJSVGFHGLXSTOOKAVEKLJNWFGPZGXYXTHELJNASRR  
ZEEERDOEHGLGTEMWFXYTETUHVRAKXPYDIFYKHRJFXKGJZWZQGNJOEYRFZSPWLUSKTBENQFNZWVTWYJMIAGLFHFGFZOZGRGGRBYVN  
ZSXESFVVBJSBWSYLOIPYULVXRRGKRHVEECDBGYBEAGHRJGLVPGPPMBEGVQRHWDPTAUMTTCOEEZNTMXYZCSURKHVQBMLZMDDVJWB  
TIRQKFQDMVPMCKHSEGBDYFAREZBGTHLNGLEJJSGETDIZMYIANKBWULCTTXZTEGBGAOKLEDFOFZDVL

VHVPSRZHVUAMPJYYMNNLJESGKIACVRDLCUSZKCRCWNGTUKNBZXTNERPDCPKSIRHGLKTWSBIFKTVOTYZHRKNYLTIZAASFQGNKCBTNVCG  
FZHVUPZIWLVUNGJEJNSRIEYADZDLITGHKSYLTLKIAJASTLATTFQFJZEEUTUGRTZJNKRUQWWCWQAEAVTYPARSIFWNUPJSZAAFIERKHUWRCGV  
CLHKYNTKZCIRLBPERYGTNEEJONQWRBEAVTYPARNAGTEUDLHONXQFKSWROVRTSFQTLUOQUPZWDEJBLCLCEZOYETGNVCSLYAAFEDAWR  
URFUOKSSTONTNOIJSNJTEKUDAZTNELEOLWVBKCBOKESWMUMRPTRCQMGSGGRJZXALRNETZZFOLAQQT

Fun Video. [Cracking the Cipher Challenge — Simon Singh — GOTO 2016](#)  
[The Science of Secrecy – Simon Singh](#)

A Challenge

Can you decrypt this ciphertext? Key is not provided :)

KHGPYVJLCJAPEPYYOYUGCWJGNVOIUOPBGDOTZCMXGDCVGCZIPOVDIQXGZCZYPLUHDGYGCQPHWCBBVHUUSHPWQENKYXOTTUMOUBWJK  
OPMVKMLHGFGYHGFGYCLEUACLEUAWQENKHVZTVPGSVOYJOEVPGBGKBKCVYUEUOYIYVABCDKKCQDCANJOEVJACJVJVYNPYHYTOCOIQB  
NJSRPNJSUPMUYOLNJSNPSQEVOCPUVOCUSUIUFDJPMVRKZWJSEHHGBAOYUNQUYYTYVRCVAVKVNIIICBFHLGIQBNGVNPHIWGABCDCTUPT  
WZNJKRWYPCVVZCVNSCMOVOUVXQOYQBEOYUDTHNGNKAADKWOFBGNGMYEVLXVVRTVOIRCZOPBQVZCXFPMCFGKBKWCUXKCJVONNPABC  
FGPNQYMOCOSPAIOIQDHHSTTQJKVDUUSVOCPUKUAJONSHGFGYWJKPNYJONSHGFGYWJKPNYGFGYMKXELBGGCZHKXGHFYKAZNJOUHGGM  
QBFFXVRYGZJPMJKPKMQEVVZVRGJUURFYUYOTIOVXQAIWBLPGOIEVONNPAVGZTLWKYVWZDKWOFMVOCSCPQVOYOLNPHFKPKBGQGAMVYD  
LUNKYFYTGJHNCKJELYMLCURQBFFFGZNQZRLXJSODBGXKOUFDJLWJKPJYCXXFIWIQBBCFGAIUDQWBKW

11.4. Linear Feedback Shift Register

How does a computer generate truly random numbers? Computers are deterministic which means the actions it takes are predetermined. So it can't generate truly random numbers unless they observe some unpredictable data like noise. But we can still generate “seemingly” random numbers called **pseudorandom numbers**. One such approach is using Linear Feedback Shift Registers (LFSRs).

An LFSR is defined by

- $n$  state variables  $x_1, x_2, x_3, \dots, x_n$  (collectively called as the state of LFSR (“register”)) with their initial values (called taps)  $t_1, t_2, t_3, \dots, t_n$  ( $t_i$  is 0 or 1).
- A feedback polynomial  $c_1x^0 + c_2x^1 + c_3x^2 + \dots + c_nx^{n-1} + x^n$  ( $c_i$  is 0 or 1) which updates the state of LFSR as follows
  - $\text{next}(x_1, x_2, x_3, \dots, x_{n-1}) = (x_2, x_3, x_4, \dots, x_n)$  – this is called “shifting” next value of  $x_1$  becomes  $x_2$ , next value of  $x_2$  becomes  $x_3$ , and so on.
  - $\text{next}(x_n) = c_1x_1 \oplus c_2x_2 \oplus \dots \oplus c_{n-1}x_{n-1} \oplus c_nx_n$  where  $\oplus$  is the binary **xor** operator – this is the “linear feedback”.
- The output bit is  $x_1$

For example, consider a 3-bit LFSR as shown in 27a. Here,  $(t_1, t_2, t_3) = (1, 1, 0)$  and  $(c_1, c_2, c_3) = (1, 0, 1)$ . Next, the sequence generation is shown in 27b. Here, the initial state  $(1, 1, 0)$  becomes  $(1, 0, 1 \oplus 0) = (1, 0, 0)$  and with similar updates, eventually the sequence repeats when the state becomes  $(1, 1, 1)$  as next state will be  $(1, 1, 1 \oplus 1) = (1, 1, 0)$ .

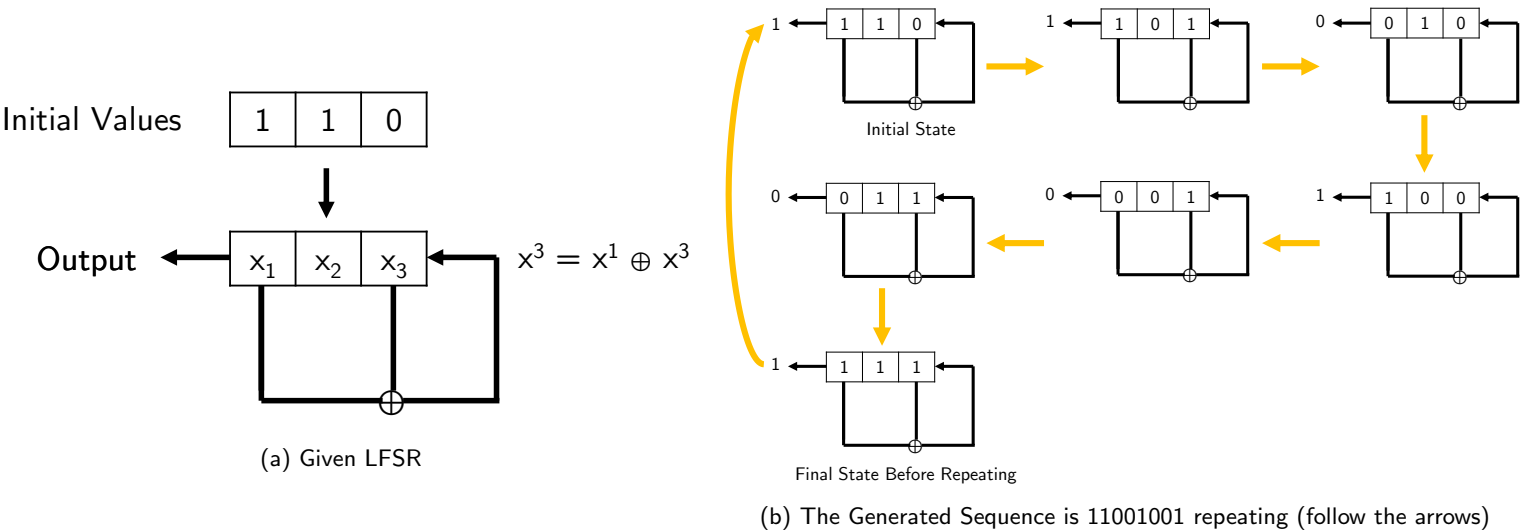


Figure 27: Linear Feedback Shift Register – Working

Problem Statement:

A property of  $n$  bit LFSR is that the output sequence it generates will start repeating in at most  $2^{n-1}$  iterations called its period<sup>5</sup>. Your task is to simulate an LFSR with a given initial state and feedback polynomial until it repeats and find its period<sup>6</sup> in the process.

Starter Code	
Input Format	
$t$	(number of test cases, an integer)
$n_i \quad t_1 \ t_2 \ \dots t_{n_i} \quad c_1 \ c_2 \ \dots c_{n_i}$	( $2n_i + 1$ space seperated integers for each testcase)
Output Format	
the output sequence generated by the given LFSR followed by the period of this output sequence	
(each iteration on a newline)	
Constraints	
$1 \leq n_i \leq 15$	
$t_i$ is either 0 or 1 and $c_1 = 1$ <sup>7</sup> , other $c_i$ are either 0 or 1	
(The LFSR will repeat from the beginning)	
Sample Input	
1 1 1	
2 1 0 1 0	
2 1 1 1 0	
2 1 1 1 1	
3 1 1 0 1 0 1	
5 1 0 1 0 0 1 0 0 1 0	
7 1 1 0 0 0 0 0 1 0 0 0 0 0 1	
Sample Output	
1 1	
1 0 2	
1 1	
1 1 0 3	
1 1 0 1 0 0 1 7	
1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 31	
1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 1 1	
1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 127	
More Test cases	
Input and Output files	

Fun Video. [Random Numbers with LFSR \(Linear Feedback Shift Register\) – Computerphile](#)

<sup>5</sup>Interestingly, there also exists a feedback polynomial which achieves this maximum period for every  $n$ .  
<sup>6</sup>Is there a way to get the period of the sequence using just the feedback polynomial and without actually calculating sequence? The basis of this problem lie in the fascinating area of mathematics known as Abstract Algebra!  
<sup>7</sup>This makes sure that the sequence will repeat from the beginning and will not have any non-periodic part. For example, 110101010... ('10' repeating) is not possible if  $c_1 = 1$ .



§12. Fractal Fun

Topics. Everthing but not actually everything if you think hard enough :)

12.1. L-Systems

Lindenmayer system, shortly L-system is a recursive system to generate self-similar patterns. Simply put, it contains variables, constants, an axiom and rules.

In fact, we have already seen its example [here](#). So, let's take that as a reference. We can generate the Thue-Morse Sequence using the below L-System

variables 0,1  
constants none  
axiom 0 (start with 0)  
rules  $0 \rightarrow 01, 1 \rightarrow 10$  (replace 0 by 01 in next step and 1 by 10)

This produces the following sequences

Iterate 0 0  
Iterate 1 01  
Iterate 2 0110  
Iterate 3 01101001  
Iterate 4 0110100110010110 and so on

12.1.1 Dragon Curve

variables F,G  
constants +-  
axiom F  
rules  $F \rightarrow F+G, G \rightarrow F-G$

The generated sequence is  $F+G+F-G+F+G-F-G+F+G+F-G-F+G-F-G \dots$ . Consider F, G as moving forward and + (-) as turning left (right) by  $90^\circ$ .

Problem Statement:

Draw the corresponding curve using turtleSim with appropriate scaling such that it roughly takes same width and height for all iterates.

Starter Code

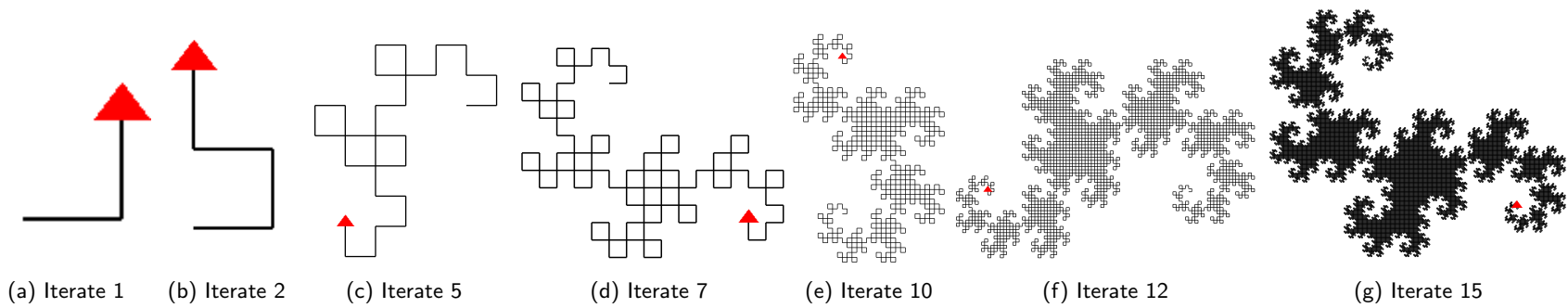


Figure 28: Dragon Curve iterates

12.1.2 Sierpiński Arrowhead Curve

variables A,B  
constants +-  
axiom A  
rules  $A \rightarrow B-A-B, B \rightarrow A+B+A$

Try generating this. Here, A, B denote moving forward and + (-) denote turning left (right) by  $60^\circ$ .

Problem Statement:

Again, draw the corresponding curve using turtleSim with appropriate scaling such that it roughly takes same width and height for all iterates.

Starter Code

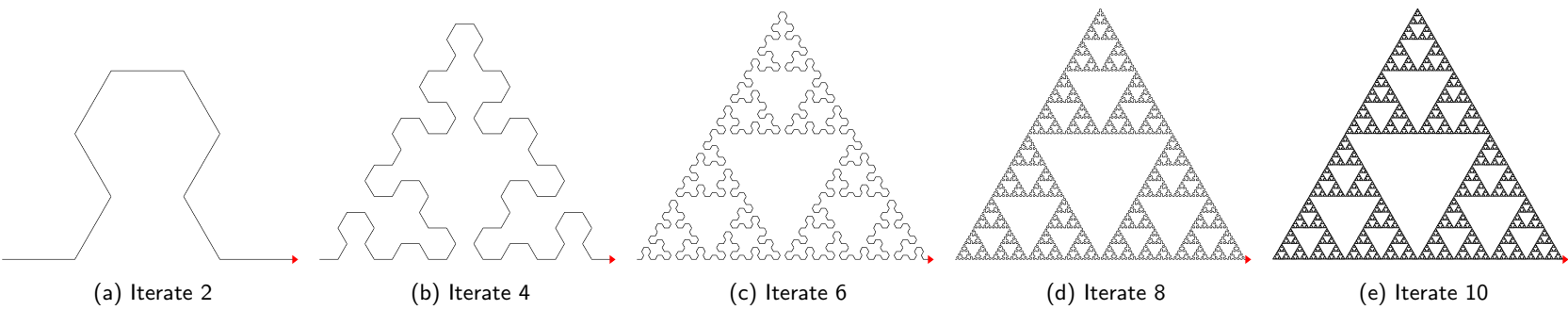


Figure 29: Sierpiński Arrowhead Curve for even iterates (why only even?)

Fun Video. [Unfolding The Dragon — Fractal Curve – Think Twice](#)  
[Fractals are typically not self-similar – 3Blue1Brown](#)

12.2. Chaos Game (Iterated Function Systems)

Intuitively, in these systems, we iterate a specific function repeatedly. For simplicity, let us only consider affine transformations<sup>8</sup>. We take a random initial point  $P = \begin{bmatrix} x \\ y \end{bmatrix}$  and repeatedly apply different affine transformations to get  $P_{\text{next}} = f_i(P)$  with some probability  $p_i$ . After large number of iterations, a pattern emerges!

$$f(x,y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{32}$$

**Note.** You can get probabilities by smartly generating random numbers. `randuv(x, y)` (Simplecpp library) generates random numbers (double) between  $x$  and  $y$ . If you feel adventurous then implement your own random number generator using **LFSRs** :).

12.2.1 Sierpiński Triangle

Consider,  $A, B, C$  as some co-ordinates of an equilateral triangle. Now, after taking a random initial point  $P$  we go half the distance towards  $A$  or  $B$  or  $C$  with equal probability and repeat this with the new point over and over. This operation can be represented using affine transformations as below

With probability 1/3, apply  $f_1(x,y) = \begin{bmatrix} 0.05 & 0.00 \\ 0.00 & 0.05 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} A_x \\ A_y \end{bmatrix}$  (33)

With probability 1/3, apply  $f_2(x,y) = \begin{bmatrix} 0.05 & 0.00 \\ 0.00 & 0.05 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} B_x \\ B_y \end{bmatrix}$  (34)

With probability 1/3, apply  $f_3(x,y) = \begin{bmatrix} 0.05 & 0.00 \\ 0.00 & 0.05 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} C_x \\ C_y \end{bmatrix}$  (35)

In limit, we get the Sierpiński Triangle.

Problem Statement:

Simulate this system and observe the generated pattern using `turtleSim` with appropriate scaling such that it takes same width and height for all iterates.

Starter Code

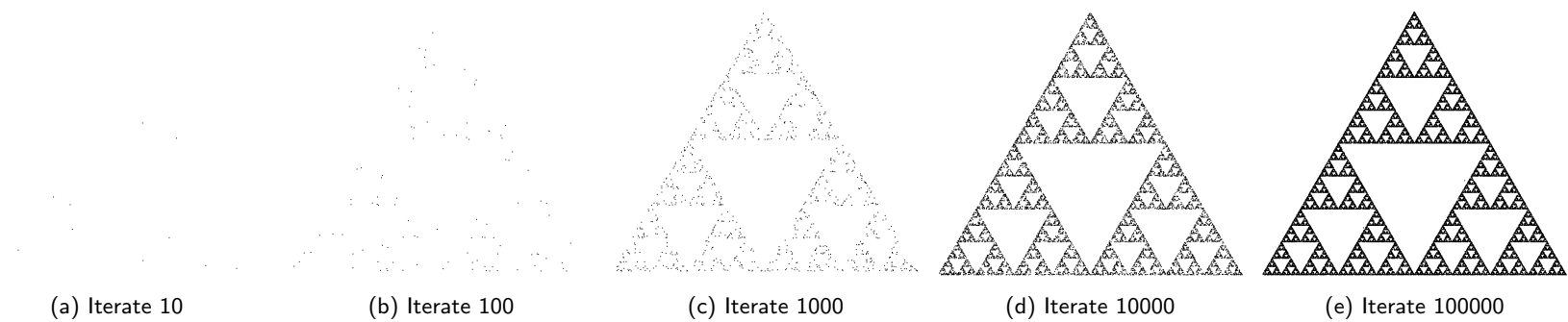


Figure 30: Sierpiński Triangle for iterates growing with power of 10

12.2.2 Barnsley's Fern

Again, by taking different  $f_i$ , we get different fractal. An explain

With probability 0.01, apply  $f_1(x,y) = \begin{bmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$  (36)

With probability 0.85, apply  $f_2(x,y) = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$  (37)

With probability 0.07, apply  $f_3(x,y) = \begin{bmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$  (38)

With probability 0.07, apply  $f_4(x,y) = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.44 \end{bmatrix}$  (39)

In limit, we get the Barnsley's Fern.

Problem Statement:

Simulate this system and observe the generated pattern using `turtleSim` with appropriate scaling such that it takes same width and height for all iterates.

Starter Code

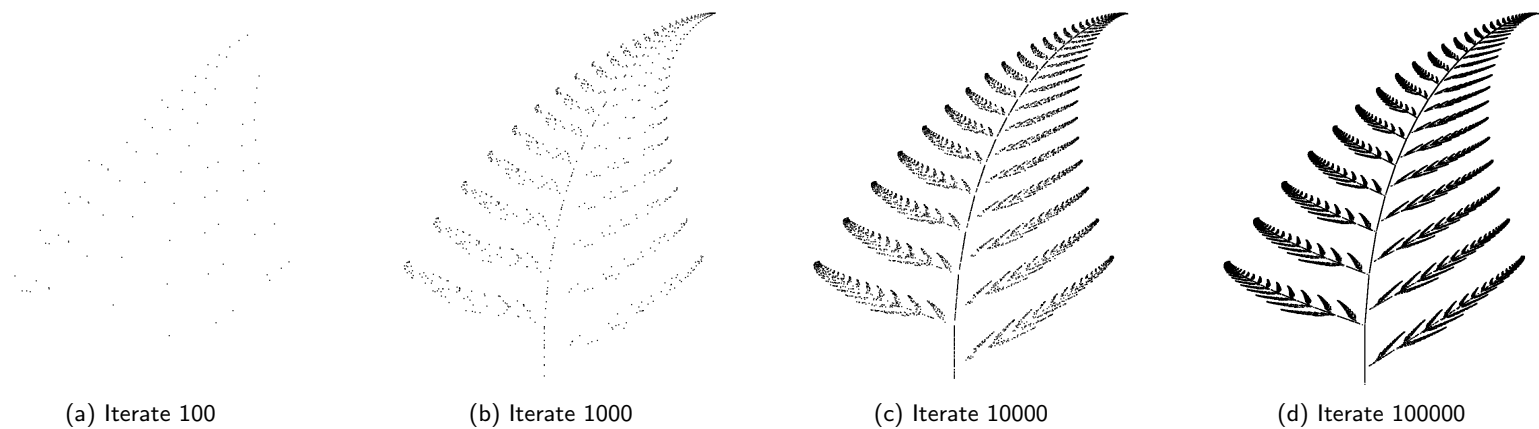


Figure 31: Barnsley's Fern for iterates growing with power of 10

**Fun Video.** [Chaos Game – Numberphile](#)  
[Chaos Game — Fractals emerging from chaos — Computer simulation – Think Twice](#)

<sup>8</sup>In general, affine transformations are of the form  $Ax + b$  where  $A$  is a matrix and  $x, b$  are vectors.