CS 101 Computer Programming and Utilization

Practice Problems

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

Spring Semester 2021-22

Last update: 2022-06-11 01:56:01+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. 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 thanks 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. Thanks to the CS101 professors, TAs, my 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, Regular Star Polygon, Hilbert Curve, Thue-Morse Sequence, Recaman's Sequence, Farey Sequence (some are yet to be added).

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

- 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.
- 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`
- 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

§1. Prodigal 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.

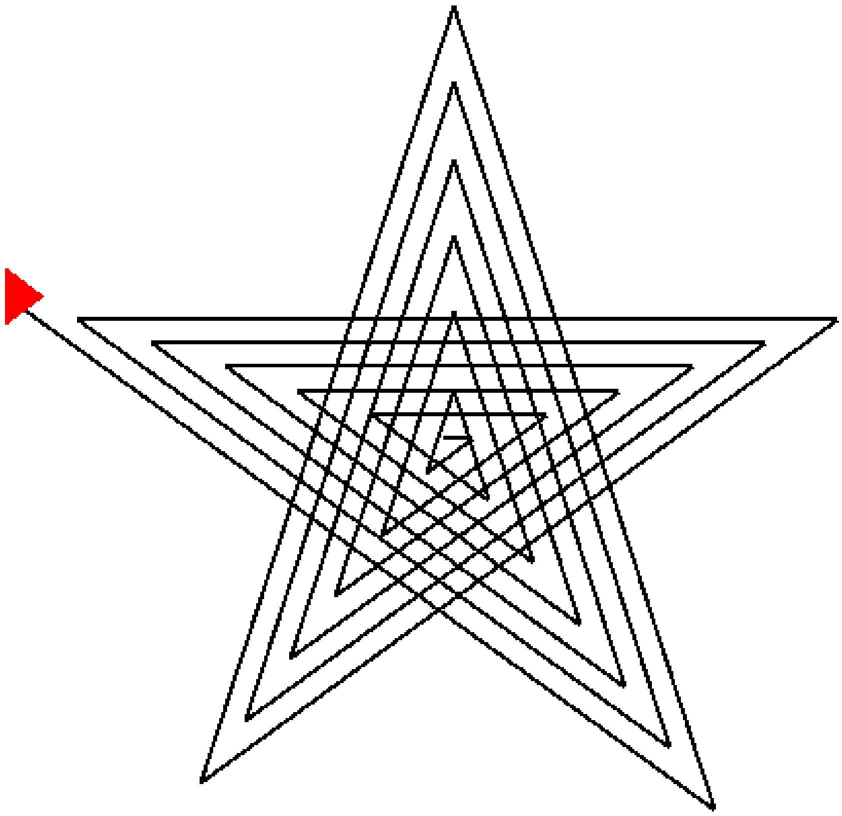
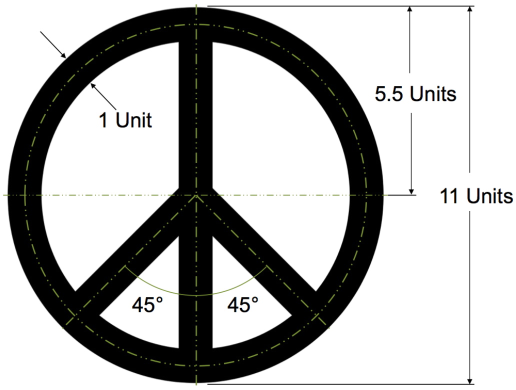


Figure 1: A Star Spiral of 30 sides

1.2. Peace

Problem Statement:
Draw the outline of the Proportionl Peace Sign according to measurements as shown in 2a.



(a) Measurements by Jerry S. Sadin, based on Peace Sign 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.

Input Format

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

Output Format

Butterfly pattern (each test case on a newline)

Constraints

$1 \leq n_i \leq 10$

Sample Input

5
1 2 3 4 5

Sample Output

```
*   *
* * *
*   *

*       *
* *   * *
* * * * *
* *   * *
*       *

*           *
* *       * *
* * *   * * *
* * * * * * *
* * *   * * *
* *       * *
*           *

*               *
* *           * *
* * *       * * *
* * * *   * * * *
* * * * * * * * *
* * * *   * * * *
* * *       * * *
* *           * *
*               *

*                   *
* *               * *
* * *           * * *
* * * *       * * * *
* * * * *   * * * * *
* * * * * * * * * * *
* * * * *   * * * * *
* * * *       * * * *
* * *           * * *
* *               * *
*                   *
```

Starter Code

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^{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.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated integers for each testcase)

Output Format

Alphabetical Floyd's Triangle (left-aligned, each test case on a newline)

Constraints

$1 \leq n_i \leq 20$

Sample Input

```
5
1 2 3 5 17
```

Sample Output

```
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
```

Starter Code

1.5. Bernoulli's Triangle

You might have heard about Pascal's Triangle. The k^{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^{th} element (partial sums).

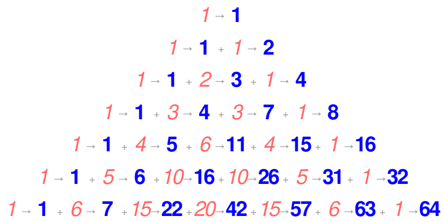


Figure 3: Bernoulli's triangle (**blue bold** text) from Pascal's triangle (*pink italics*) (Drawn by CMG Lee, Image Source)

Problem Statement:
Print the left-aligned Bernoulli's Triangle for all given n. See Starter code (below) for more details.

Input Format
t (number of test cases, an integer)
$n_1\ n_2\ \ldots\ n_t$ (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
```

Starter Code

Fun Video. *Pascal's Triangle – Numberphile*
What You Don't Know About Pascal's Triangle – Tipping Point Math

§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.

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

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

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

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated 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

Starter Code

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

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

Let's define π_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 π_n for all test cases accurate till 10 decimal places. See Starter code (below) for more details.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated integers for each testcase)

Output Format

π_{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
```

Starter Code

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:

$$^{n}a = \underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{n} \quad \text{repeated exponentiation} \tag{3}$$

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

Input Format

t (number of test cases, an integer)

$a_1\ n_1 \quad a_2\ n_2 \quad \ldots \quad a_t\ n_t$ (t space seperated pairs for each testcase)

Output Format

^{n}a (each test case on a newline, accurate till 10 decimal places)

Constraints

$0.05 \leq a_i \leq 3$ (a double)

$1 \leq n_i \leq 1000$ (an integer)

Sample Input

```
10
1 1   1 2   2 1   2 2   2 3   3 2   3 3   1.41421356237 20   0.06598803584 1000   1.44466786101 1000
```

Sample Output

```
1.0000000000
1.0000000000
2.0000000000
4.0000000000
16.0000000000
27.0000000000
7625597484987.0000000000
1.9995856229
0.3968311347
2.7128728643
```

Starter Code

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

$$r = \sqrt{1 + 2\sqrt{1 + 3\sqrt{1 + 4\sqrt{1 + \cdots}}}} = \lim_{n\to\infty} \sqrt{1 + 2\sqrt{1 + 3\sqrt{\cdots\sqrt{1 + n}}}} \tag{4}$$

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.

Input Format

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

Output Format

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

Constraints

$2 \le n_i \le 100$

Sample Input

```
8
2 3 5 10 20 30 50 100
```

Sample Output

```
1.7320508076
2.2360679775
2.7550532613
2.9899203606
2.9999878806
2.9999999868
3.0000000000
3.0000000000
```

Starter Code

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

2.5. Simple Continued Fractions

A (finite) simple continued fraction of a rational number r is defined using $n+1$ coefficients $= [a_0; a_1, a_2, \ldots, a_{n-1}, a_n]$. They can be expressed in Gauss' Kettenbruch notation as follows

$$r = a_0 + \underset{i=1}{\overset{n}{\mathrm{K}}} \frac{1}{a_n} \triangleq a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}} \tag{5}$$

Problem Statement:
Express r as a quotient p/q where p, q are integers and $q \neq 0$. See Starter code (below) for more details.

Input Format

t (number of test cases, an integer)

$n_i \quad a_{n_i}\ a_{n_{i-1}}\ \ldots\ a_1\ a_0$ ($n_i + 2$ space seperated integers for each testcase)

Output Format

p_{n_i}/q_{n_i} (each test case on a newline, where $r_{n_i} = p_{n_i}/q_{n_i}$ (in irreducible form))

Constraints

$0 \leq n_i \leq 50$

a_0 is an integer whereas $a_1, a_2, \ldots, a_{n_i-1}, a_{n_i}$ are positive integers

$a_0, a_1, a_2, \ldots, a_{n_i-1}, a_{n_i}$ are such that $-2,147,483,648 \leq p_{n_i}, q_{n_i} \leq 2,147,483,647$ (C++'s `int` range)

Sample Input

```
11
0 0
1 1 0
1 1 1
1 7 3
8 1 1 1 1 1 1 1 1 1
10 1 1 1 1 1 1 1 1 1 2 -2
3 1 15 7 3
9 13 3 4 1 2 1 2 1 1 0
12 14 1 3 1 2 1 1 1 292 1 15 7 3
22 1 1 14 1 1 12 1 1 10 1 1 8 1 1 6 1 1 4 1 1 2 1 2
45 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
```

Sample Output

```
0/1
1/1
2/1
22/7
55/34
-233/144
355/113
3035/5258
80143857/25510582
848456353/312129649
1134903170/1836311903
```

Starter Code

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.

$$\sqrt{\frac{\pi e}{2}} = \cfrac{1}{1+\cfrac{1}{1+\cfrac{2}{1+\cfrac{3}{1+\cfrac{4}{1+\ddots}}}}} + \left\{1 + \frac{1}{1\cdot 3} + \frac{1}{1\cdot 3\cdot 5} + \frac{1}{1\cdot 3\cdot 5\cdot 7} + \frac{1}{1\cdot 3\cdot 5\cdot 7\cdot 9} + \cdots\right\} \tag{6}$$

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

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

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.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (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
```

Starter Code

Fun Video. *7 factorials you probably didn't know – blackpenredpen*
The Man Who Knew Infinity – Tipping Point Math

2.7. Viète's π Formula

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

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

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

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

Problem Statement:
Calculate π_n for all test cases accurate till 15 decimal places. See Starter code (below) for more details.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated integers for each testcase)

Output Format

π_{n_i} (each test case on a newline, accurate till 15 decimal places)

Constraints

$1 \leq n_i \leq 50$

Sample Input

8
1 2 3 5 10 20 30 50

Sample Output

2.828427124746190
3.061467458920718
3.121445152258052
3.140331156954753
3.141591421511200
3.141592653588618
3.141592653589793
3.141592653589793

Starter Code

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 $x_1, x_2, \ldots, x_n$, it is defined as

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

Its special cases are

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

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

Input Format
t (number of test cases, an integer)
$n_i \quad x_1 \; x_2 \; \ldots x_{n_i-1} \; x_{n_i}$ ($n_i + 1$ space seperated numbers for each testcase)

Output Format
$M_p(x_1, \ldots, x_n)$ for $p = \{-\infty, -1, 0, 1, 2, 3, \infty\}$ (each test case on a newline, accurate till 5 decimal places))

Constraints
$1 \le n_i \le 50$ (an integer)
$0 < x_i \le 100$ (a double)
Also assume that the calculations are always within the range of double

Sample Input
```
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
```

Sample Output
```
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
```

More Test cases
Input and Output files

Starter Code

2.9. Shoelace Formula

Shoelace Formula determines the area of a simple polygon whose vertices are given by Cartesian coordinates.

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

which can be simplfied 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.

Input Format
t (number of test cases, an integer)
$n_i \quad x_1\ y_1 \quad x_2\ y_2 \quad \cdots \quad x_n\ y_n$ ($2n_i + 1$ space seperated 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

Starter Code

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 4.

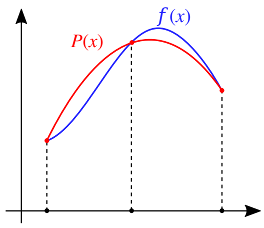


Figure 4: Approximating $f(x)$ by a parabola $P(x)$. (Image Source)

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

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

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

where each of the $n+1$ ordinates is given by $x_i = a + i\Delta x$ for $i = 0, 1, \ldots, n$ 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:

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

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

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated integers for each testcase)

Output Format

π_{n_i} (each test case on a newline, accurate till 15 decimal places)

Constraints

$0 < n_i < 500$ and n_i is odd

Sample Input

```
10
3 5 7 11 15 31 57 99 163 441
```

Sample Output

```
3.140773809523810
3.141684884891457
3.141601987350571
3.141593090129691
3.141592711563659
3.141592654188570
3.141592653603947
3.141592653590286
3.141592653589817
3.141592653589793
```

Starter Code

§3. Traditional Conditionals

Topics. `if else` *statement, loop contol 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 $< 90°$

Right One angle $= 90°$

Obtuse One angle $> 90°$

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.

Input Format
t (number of test cases, an integer)
$a_i\ b_i\ c_i$ (three space seperated 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
```

Starter Code

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.,* $0 \le \angle_{ij} \le 180$.

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., $0 \le$ Hours ≤ 23, $0 \le$ Minutes ≤ 59, $0 \le$ Seconds ≤ 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

Starter Code

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.

Input Format

t (number of test cases, an integer)

$n_i \quad a_1\ a_2\ \ldots\ a_{n_i-1}\ a_{n_i}$ ($n_i + 1$ space seperated 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

Starter Code

Note. *Verify your program on even more testcases from here.*

3.4. 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:

$$\left(\sum_{i=1}^{10}(11-i)x_i\right)\%11 = 0 \tag{14}$$

3.4.1 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 as 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 $(1 \leq i \leq 10)$.

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 seperated)
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
```

Starter Code

Fun Video. *11.11.11 – Numberphile*

3.5. 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)

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
$1 \leq$ Date ≤ 99, $1 \leq$ Month ≤ 99, $1 \leq$ Year ≤ 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

Starter Code

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

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} \quad n \in \mathbb{Z} \quad \text{(Yes! They can be extended to negative numbers)}
\end{aligned}
\tag{15}
$$

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:

$$0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, \ldots \text{(A082115)}$$

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 $n_1, n_2, \ldots, n_t$

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

Output Format
$\pi(n_i)$ (each test case space seperated)

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

Starter Code

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.

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
"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

Starter Code

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.

$$K_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{8} + \frac{1}{10} + \cdots + \frac{1}{n} = \sum_{i=1}^{n} c_i \frac{1}{i} \text{ where } c_i = \begin{cases} 0 & \text{if } i\text{'s decimal expansion contains a } 9 \\ 1 & \text{else} \end{cases} \quad (16)$$

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.

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

Output Format
K_{n_i} (each test case on a newline, accurate till 10 decimal places)

Constraints
$1 \le n_i \le 10^6$

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

Starter Code

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:

$$(a_n \dots a_2 a_1 a_0)_{(-2)} = a_n(-2)^n + \cdots + a_2(-2)^2 + a_1(-2)^1 + a_0(-2)^0 \quad \text{where } a_i \text{ is either 0 or 1} \tag{17}$$

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 it as shown below,

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

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.

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
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

Starter Code

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 $(n_1, n_2, \ldots, n_t)$ to decimal.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated numbers for each testcase)

Output Format

Converted decimal number (space seperated)

Constraints

$0 \leq n_i \leq 10^{15}$, a maximum of 8 digits after binary point ('.') (base 2, a double)

Sample Input

9
1 111 110001 101010111 100101100001 1.00011001 11.001001 110.01 10110.01110101

Sample Output

1.00000000 7.00000000 49.00000000 343.00000000 2401.00000000 1.09765625 3.14062500
6.25000000 22.45703125

Starter Code

(b) **Problem Statement:**
Convert t positive decimal numbers $(n_1, n_2, \ldots, n_t)$ to binary.

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated numbers for each testcase)

Output Format

Converted binary number truncated till 8 decimal places (space seperated)

Constraints

$0 \leq n_i \leq 2500$, a maximum of 8 digits after decimal point ('.') (base 10, a double)

Sample Input

9
1 7 49 343 2401 1.1 3.1415 6.25 22.459

Sample Output

1.00000000 111.00000000 110001.00000000 101010111.00000000 100101100001.00000000
1.00011001 11.00100100 110.01000000 10110.01110101

Starter Code

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:

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases} \tag{18}$$

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 .

$$3 \xrightarrow[(1)]{3\times3+1} 10 \xrightarrow[(2)]{10/2} 5 \xrightarrow[(3)]{3\times5+1} 16 \xrightarrow[(4)]{16/2} 8 \xrightarrow[(5)]{8/2} 4 \xrightarrow[(6)]{4/2} 2 \xrightarrow[(7)]{2/2} 1$$

Problem Statement:
Your task is to return the number of operations required to reach 1[1] for arbitrary number of inputs.

Input Format
$n_1\ n_2\ \ldots n_i\ \ldots -1$ (space separated arbitrary number of testcases, stop when input is negative)

Output Format
number of operations required to reach 1 with initial value of $n = n_i$ (space seperated for each test case)

Constraints
$1 \leq n_i \leq 10^6$

Function(s) to Implement
`void f(long long &n)` – updates value of n
`int count_operations(long long n)` – returns the number of operations required to reach 1

Sample Input
1 3 7 9 27 255 871 4255 77031 665215 837799 -1

Sample Output
0 7 16 19 111 47 178 201 350 441 524

Starter Code

Fun Video. *Collatz Conjecture: The Simplest Math Problem No One Can Solve – Veritasium*

[1] 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 abundancy index.
The abundancy index of a number is the ratio of sum of divisors of that number and the number itself.

$$\text{abundancy index} = \frac{\sigma(n)}{n} \quad \text{where } \sigma(n) \text{ is the sum of divisors of } n \tag{19}$$

For example, 6 and 28 form a friendly pair[2] 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 abundancy (if it exists) as a quotient p/q where p, q are integers and $q \neq 0$.

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

Output Format
Output the common abundancy 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 abundancy $= p_{a_i}/q_{a_i}$ 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 abundancy 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
```

Starter Code

Fun Video. *A Video about the Number 10 – Numberphile*

[2]in fact, they are called perfect numbers as their abundancy = 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?).

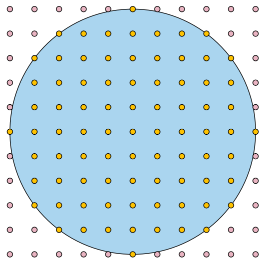


Figure 5: A circle with $r = 5$ units bounding 81 integer points. $N(r) = 81 \sim \pi r^2 \approx 78.54$

Consider the subproblem of finding $M(i)$ – the number of $(x, y) \in \mathbb{Z}^2$ such that $x^2 + y^2 = i$ where $i \in \{;0, 1, \ldots, n\}$. Clearly $N(r) = \sum_{i=0}^{r^2} M(i) \to N(\sqrt{n}) = \sum_{i=0}^{n} M(i)$. Now,

$$M(i) = 4 \sum_{j|n} \chi(j) \quad \text{where} \quad \chi(n) = \begin{cases} 1 & \text{if } n\%4 = 1 \\ -1 & \text{if } n\%4 = 3 \\ 0 & \text{else} \end{cases} \tag{20}$$

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

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots\ n_t$ (t space seperated integers for each testcase)

Output Format

$N(\sqrt{n_i})$ (each test case space seperated)

Constraints

$1 < n_i \leq 10^7$

Function(s) to Implement

`int X(int n)` – returns $\chi(n)$

`int count_lattice_points(int n)` – returns $M(n)$

Sample Input

15
0 1 2 3 5 10 20 30 50 100 1000 10000 100000 1000000 10000000

Sample Output

1 5 9 9 21 37 69 97 161 317 3149 31417 314197 3141549 31416025

Starter Code

Interesting Observation. *Does the last few outputs look familiar? How can this happen? :o*
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 apporach 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

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right) \qquad \text{For all primes } p \leq n \tag{21}$$

So, if $n = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}$, where $p_1, p_2, \ldots, p_r$ 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

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

Output Format
$\varphi(n_i)$ (each test case on a newline)

Constraints
$1 < n_i \leq 10^9$

Function(s) to Implement
`int totient(int n)` – returns $\varphi(n)$

Sample Input
```
13
1 4 8 20 44 69 97 120 2520 55440 277200 720720 88888888
```

Sample Output
```
1
2
4
8
20
44
96
32
576
11520
57600
138240
12690687
```

Starter Code

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°).

Construction via vertex connection Connect every m^{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.

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 correspoding regular star polygon

Sample Input
See 6.

Sample Output
See 6.

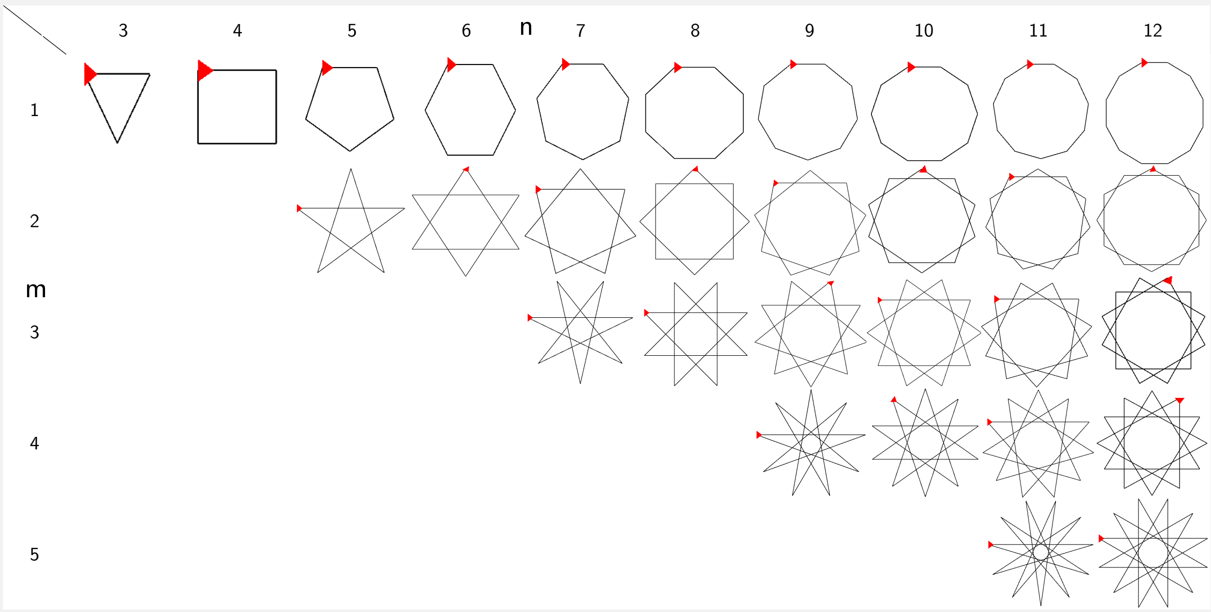


Figure 6: Some inputs m, n and their correspoding star polygons in a tabular fashion.

Starter Code

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

$$\begin{aligned} \mathrm{A}(0, n) &= n + 1 \\ \mathrm{A}(m, 0) &= \mathrm{A}(m-1, 1) \\ \mathrm{A}(m, n) &= \mathrm{A}(m-1, \mathrm{A}(m, n-1)) \end{aligned} \tag{22}$$

Problem Statement:
Calculate $\mathrm{A}(m, n)$ (given m, n) for all test cases.

Input Format

t (number of test cases, an integer)

$m_1\ n_1 \quad m_2\ n_2 \quad \ldots \quad m_t\ n_t$ (t space seperated integer pairs for each testcase)

Output Format

$\mathrm{A}(m_i, n_i)$ (each on a newline)

Constraints

m_i, n_i are postive integers such that $\mathrm{A}(m_i, n_i)$ 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
```

Starter Code

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 $x^0, x^1, x^2, \ldots, x^n$ 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 23, is an optimal algorithm for polynomial evaluation needing only n multiplications and n additions.

$$f(x) = a_0 + x\Big(a_1 + x\Big(a_2 + x\big(a_3 + \cdots + x(a_{n-1} + x\,a_n)\cdots\big)\Big)\Big) \tag{23}$$

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

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 seperated 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
```

Starter Code

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 $a_{n_i}\ a_{n_{i-1}}\ \ldots\ a_1\ a_0$. 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.[3]

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

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

$$x \pmod k \to x^2 \pmod k \to x^4 \pmod k \to x^8 \pmod k \to \cdots \to x^{2^{\lfloor \log y \rfloor}} \pmod k \tag{24}$$

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,

$$x^{25} = x^{11001_2} = x^{10000_2} \cdot x^{1000_2} \cdot x^{1_2} = x^{16} \cdot x^8 \cdot x^1$$

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?

Input Format
t (number of test cases, an integer)
$x_1\ y_1 \quad x_2\ y_2 \quad \ldots \quad x_t\ y_t$ (t space seperated integer pairs for each testcase)

Output Format
$x_i^{y_i} \pmod k$ (each test case on a newline)

Constraints
$1 < x_i,\ y_i \leq 10^9$

Function(s) to Implement
`int mod_exp(int x, int y, int k)` – returns $x^y \pmod k$

Sample Input
5
3 4   2 8   123 123   129612095 411099530   241615980 487174929

Sample Output
81
256
921450052
276067146
838400234

Starter Code

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?

Input Format
t (number of test cases, an integer)
$x_1\ y_1\ z_1 \quad x_2\ y_2\ z_2 \quad \ldots \quad x_t\ y_t z_t$ (t space seperated triples for each testcase)

Output Format
$x_i^{y_i^{z_i}} \pmod k$ (each test case on a newline)

Constraints
$1 < x_i,\ y_i,\ z_i \leq 10^9$

Function(s) to Implement
`int mod_super_exp(int x, int y, int z, int k)` – returns $x^{y^z} \pmod k$

Sample Input
5
3 7 1   15 2 2   3 4 5   427077162 725488735 969284582   690776228 346821890 923815306

Sample Output
2187
50625
763327764
464425025
534369328

Starter Code

Note. *Verify your program on more testcases from here.*

Fun Video. *Square & Multiply Algorithm - Computerphile*

[3]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, their are 5 partitions of 4 given by $\{4, 3+1, 2+2, 2+1+1, 1+1+1+1\}$.
Let use denote the number of partitions of n by $\mathrm{P}(n)$.
Now, we move to a seemingly unrelated theorem.

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

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{k=-\infty}^{\infty}(-1)^k x^{k(3k-1)/2} = 1 + \sum_{k=1}^{\infty}(-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right) \tag{25}$$

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 $1, 2, 5, 7, 12, \ldots$ on the right hand side are called (generalized) pentagonal numbers (A001318). They are given by the formula $p_k = k(3k-1)/2$ for $k = 1, -1, 2, -2, 3, -3, \ldots$

Equation 25 implies a recurrence relation for calculating $\mathrm{P}(n)$ given by

$$\mathrm{P}(n) = \mathrm{P}(n-1) + \mathrm{P}(n-2) - \mathrm{P}(n-5) - \mathrm{P}(n-7) + \cdots = \sum_{k \neq 0}(-1)^{k-1}\,\mathrm{P}(n-p_k) \tag{26}$$

Problem Statement:
Calculate $\mathrm{P}(n)$ for all test cases using25 or otherwise :).

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

Output Format
$\mathrm{P}(n_i)$ (each test case on a newline)

Constraints
$1 \leq n_i \leq 40$

Function(s) to Implement
`int P(int n)` – returns $\mathrm{P}(n)$

Sample Input
9
1 2 3 4 5 10 20 30 40

Sample Output
1
2
3
5
7
42
627
5604
37338

Starter Code

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

$$n_b = a_0 \cdot b^0 + a_1 \cdot b^1 + \cdots \quad \text{where } a_i\text{'s} \in \{0, 1, \ldots, b-1\} \tag{27}$$

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 $0, 1, 2, \ldots, b-1$ for all exponents of b.

This is the Hereditary Representation! Representing a natural number n_b in base b using only $0, 1, 2, \ldots, b-1$ 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,

$$\begin{aligned} 666_2 &= 2^1 + 2^3 + 2^4 + 2^7 + 2^9 \\ &= 2^1 + 2^{2^0+2^1} + 2^{2^2} + 2^{2^0+2^1+2^2} + 2^{2^0+2^3} \\ &= 2^1 + 2^{2^0+2^1} + 2^{2^{2^1}} + 2^{2^0+2^1+2^{2^1}} + 2^{2^0+2^{2^0+2^1}} \end{aligned} \tag{28}$$

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 (≥ 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 $(A, B, C, \ldots, Z)$ to denote $(10, 11, 12, \ldots, 35)$ respectively.

Input Format
t (number of test cases, an integer)
$n_1\ b_1 \quad n_2\ b_2 \quad \ldots \quad n_t\ b_t$ (t space seperated pairs (number, base) for each testcase)

Output Format
Hereditary Representation of n_i in base b_i (each on a newline)

Constraints
$1 < n_i \leq 2 \cdot 10^{18}$
$1 < b_i \leq 35$

Function(s) to Implement
`void Hereditary (long long num, int base)` – prints the required representation

Sample Input
9
2 2   10 2   100 2   666 3   3435 3   3816547290 4   3816547290 9   3816547290 35   1162849439785405935 10

Sample Output
```
2^{1}

2^{1} + 2^{2^{0} + 2^{1}}

2^{2^{1}} + 2^{2^{0} + 2^{2^{1}}} + 2^{2^{1} + 2^{2^{1}}}

2*3^{2} + 2*3^{3^{0} + 3^{1}} + 2*3^{2*3^{0} + 3^{1}}

2*3^{1} + 3^{3^{1}} + 2*3^{2*3^{0} + 3^{1}} + 3^{2*3^{1}} + 3^{3^{0} + 2*3^{1}}

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}}

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}}

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}}
+ 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}} +
A^{7*A^{0} + A^{1}} + A^{8*A^{0} + A^{1}}
```

More Test cases
Input and Output files

Starter Code

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.

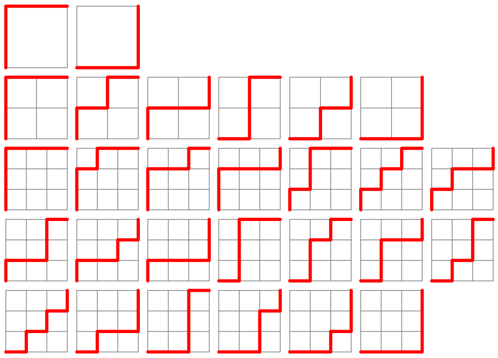


Figure 7: Example walks for case $m = n = 1$ $(\#2)$, $m = n = 2$ $(\#6)$, $m = n = 3$ $(\#20)$ (Image Source)

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

Input Format
t (number of test cases, an integer)
$m_1\ n_1 \quad m_2\ n_2 \quad \ldots \quad m_t\ n_t$ (t space seperated 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

Starter Code

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*.

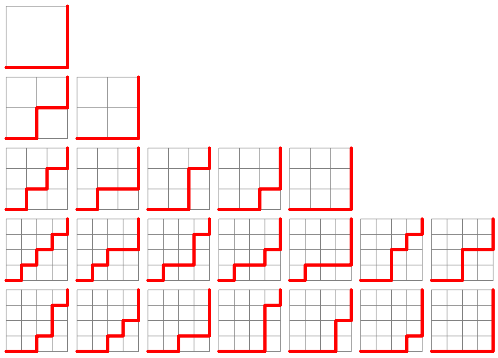


Figure 8: Example walks for case $n = 1$ $(\#1)$, $n = 2$ $(\#2)$, $n = 3$ $(\#5)$, $n = 4$ $(\#14)$ (Image Source)

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

Input Format

t (number of test cases, an integer)

$n_1\ n_2\ \ldots n_t$ (t space seperated integers for each testcase)

Output Format

Number of Dyck Paths for n_i (each test case on a newline)

Constraints

$1 \leq n_i \leq 15$

Function(s) to Implement

`int dyck_paths(int n)` – returns the number of possible staircase walks for n.

Sample Input

```
15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Sample Output

```
1
2
5
14
42
132
429
1430
4862
16796
58786
208012
742900
2674440
9694845
```

Starter Code

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.

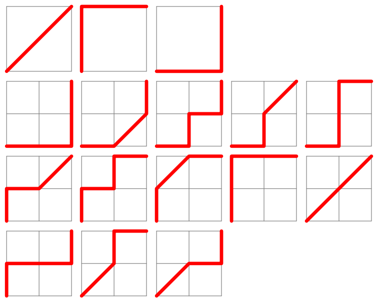


Figure 9: Example walks for case $m=n=1$ $(\#2)$, $m=n=2$ $(\#6)$, $m=n=3$ $(\#20)$ (Image Source)

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

Input Format
t (number of test cases, an integer)
$m_1\ n_1 \quad m_2\ n_2 \quad \ldots \quad m_t\ n_t$ (t space seperated integer pairs for each testcase)

Output Format
Number of Delannoy Numbers for m_i, n_i (each test case on a newline)

Constraints
$1 \leq m_i, n_i \leq 13$

Function(s) to Implement
`int delannoy_number(int m, int n)` – returns the number of Delannoy Numbers for m, n.

Sample Input
11
1 1 2 2 3 3 5 5 10 10 13 13 2 5 3 3 6 3 7 10 13 8

Sample Output
3
13
63
1683
8097453
1409933619
61
63
377
433905
8405905

Starter Code

7.4. Schröder Number

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

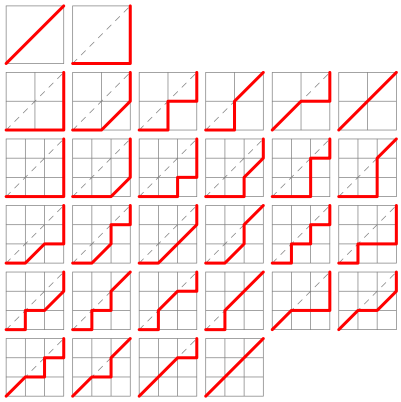


Figure 10: Example walks for case $n = 1$ (#2), $n = 2$ (#6), $n = 3$ (#22) (Image Source)

Problem Statement:
Find the *Schroder Number* for a given n (for all test cases).

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

Output Format
Number of Schroder Numbers for n_i (each test case on a newline)

Constraints
$1 \leq n_i \leq 14$

Function(s) to Implement
`int schroder_number(int n)` – returns the number of possible delannoy walks for n.

Sample Input
14
1 2 3 4 5 6 7 8 9 10 11 12 13 14

Sample Output
2
6
22
90
394
1806
8558
41586
206098
1037718
5293446
27297738
142078746
745387038

Starter Code

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 fowards *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).

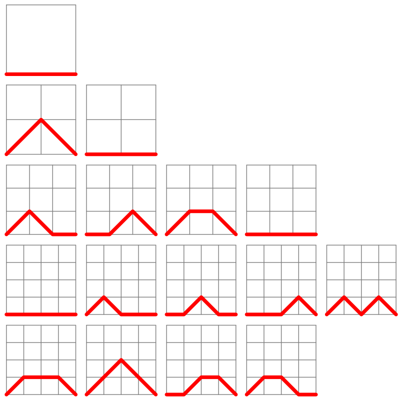


Figure 11: Example walks for case $n = 1$ $(\#1)$, $n = 2$ $(\#2)$, $n = 3$ $(\#4)$, $n = 4$ $(\#9)$ (Image Source)

Problem Statement:
Find the *Motzkin Number* for a given n (for all test cases).

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

Output Format
Number of Motzkin Numbers for n_i (each test case on a newline)

Constraints
$1 \leq n_i \leq 20$

Function(s) to Implement
`int motzkin_number(int n)` – returns the number of possible walks for n.

Sample Input
10
1 2 3 4 5 8 11 14 17 20

Sample Output
1
2
4
9
21
323
5798
113634
2356779
50852019

Starter Code

7.6. Hilbert Curve

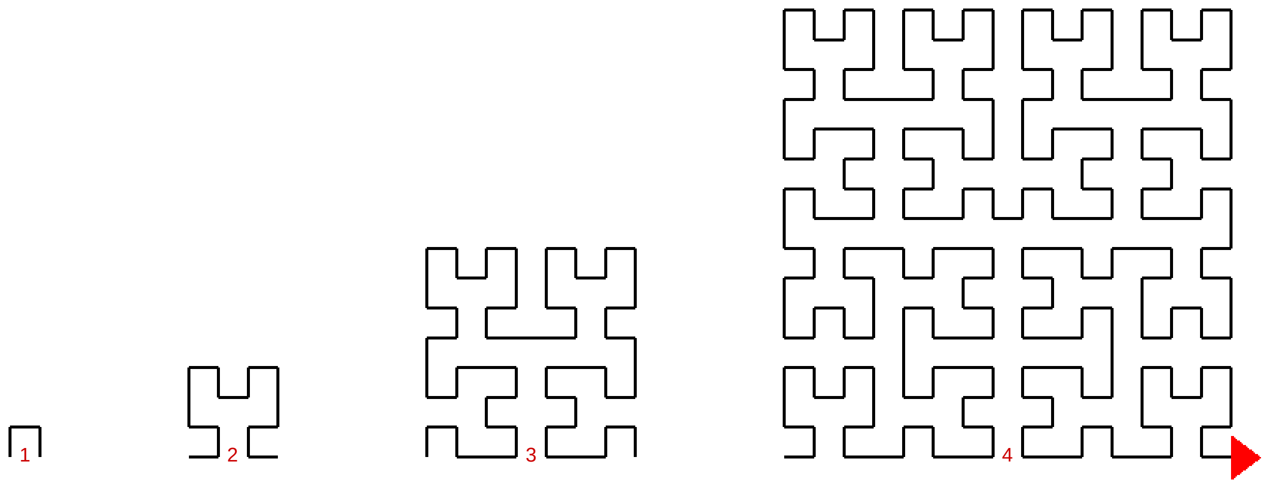


Figure 12: Hilbert Curve (Image Source)

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!

Feel free to discuss your thoughts.

Fun Video. *Hilbert's Curve: Is infinite math useful?*
Recursive PowerPoint Presentations [Gone Fractal!]

§8. Sequence Imminence

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

Note. *This is the last problem set where we use Simplecpp. We will move onto C++ from the next problem set.*

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^{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^{th} terrorist is killed until only one survives. Then the last terrorist is killed.

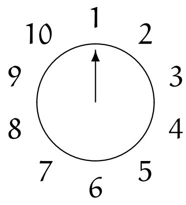


Figure 13: Example arrangement of 10 terrorists

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.

Input Format
t (number of test cases, an integer)
$n_1\ k_1 \quad n_2\ k_2 \quad \ldots \quad n_t\ k_t$ (t space seperated 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
```

Starter Code

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 $a_0, a_1, \ldots, a_n$ of the Van Eck's Sequence.

Input Format
n (a single integer)

Output Format
$a_0\ a_1\ \ldots\ a_n$ (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

Starter Code

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.

Input Format
n (a single integer)

Output Format
First n iterations of the look-and-say sequence (each iteration on a newline)

Constraints
$1 \leq n_i \leq 40$

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

Starter Code

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 0**1**
- Append complement of **01**, we get 01**10**
- Append complement of **0110**, we get 0110**1001** and so on.

Also, by using Thue-Morse sequence elements in the turtle simulator, we get a mysterious curve[4] 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.

Input Format
n (a single integer)

Output Format
First n elements of the Thue-Morse sequence and the curve.

Constraints
$1 \leq n \leq 100000$ (n need not be a power of 2)

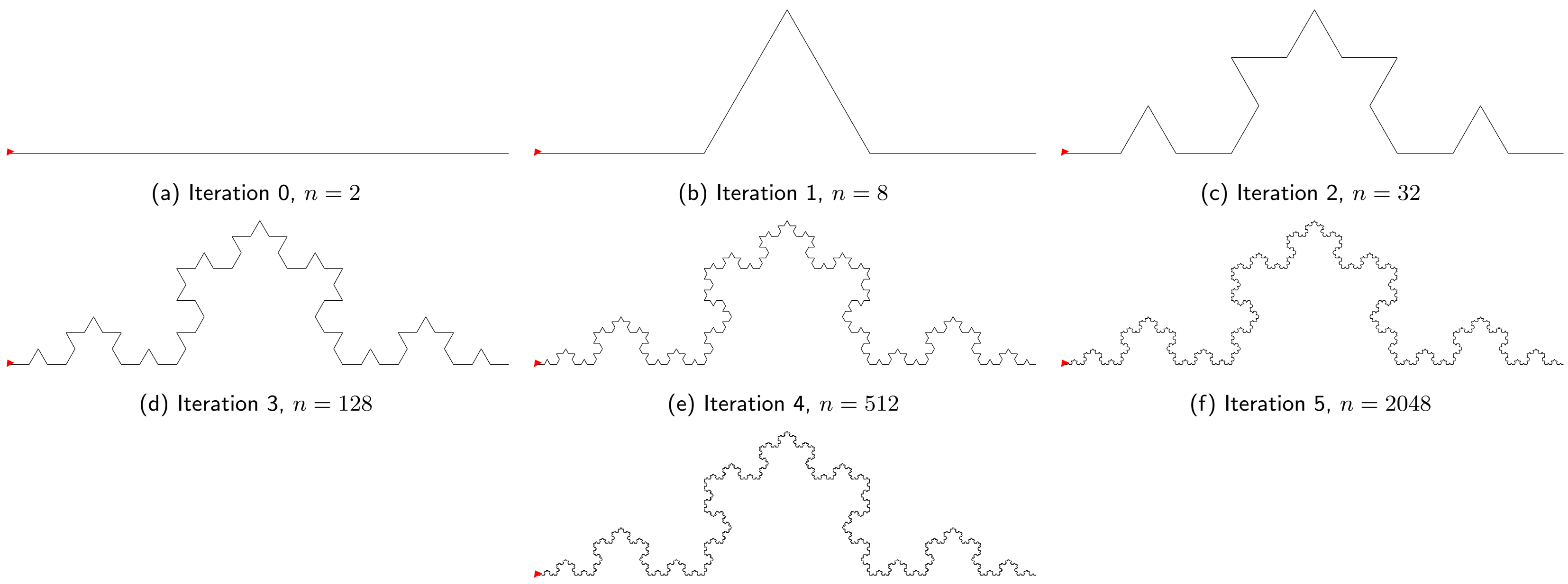
Sample Input
111

Sample Output
011010011001011010010110011010011001011001101001011010011001011010010110011010010110100110010110011010011001011

More Test cases
Input and Output files

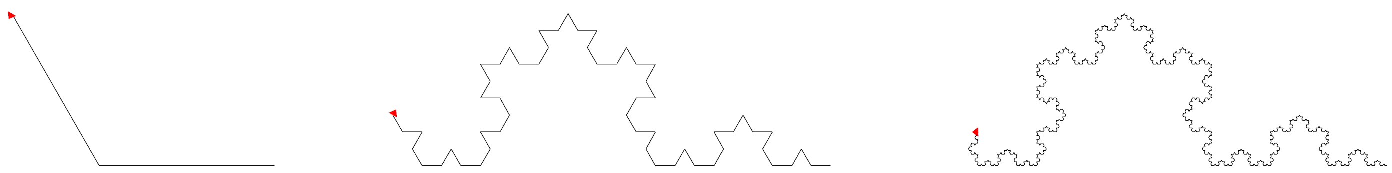
Starter Code

The output Koch Curve convergents



(a) Iteration 0, $n = 2$ (b) Iteration 1, $n = 8$ (c) Iteration 2, $n = 32$
(d) Iteration 3, $n = 128$ (e) Iteration 4, $n = 512$ (f) Iteration 5, $n = 2048$
(g) Iteration 6, $n = 8192$

Figure 14: Koch Curve Iterations and the outputs for odd powers of 2



(a) $n = 4$ (b) $n = 111$ (c) $n = 1729$

Figure 15: The outputs for numbers which are not a odd power of 2

Fun Video. *The Fairest Sharing Sequence Ever – Stand-up Maths*
Fractals are typically not self-similar – 3Blue1Brown

[4]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 16 by following the below rules:

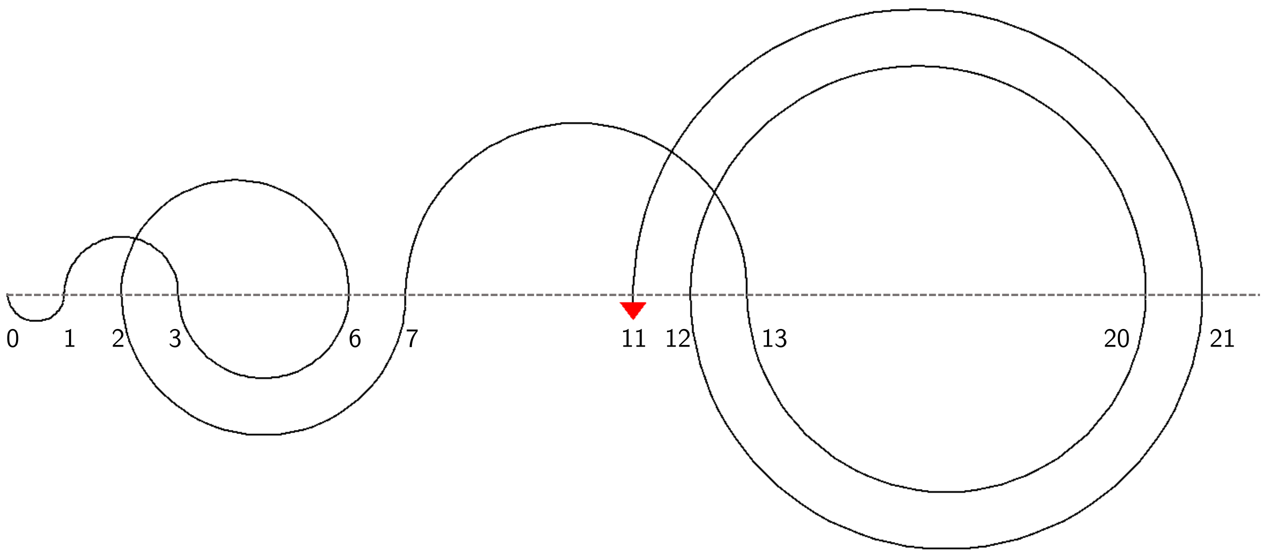


Figure 16: Recaman's Sequence Drawing Procedure

- 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 $r_0, r_1, \ldots, r_n$ of the Recaman's Sequence and draw the corresponding curve using `turtleSim`.

Input Format
n (a single integer)

Output Format
First $n+1$ elements of the Recaman's sequence and the curve.

Constraints
$1 \leq n \leq 1000$

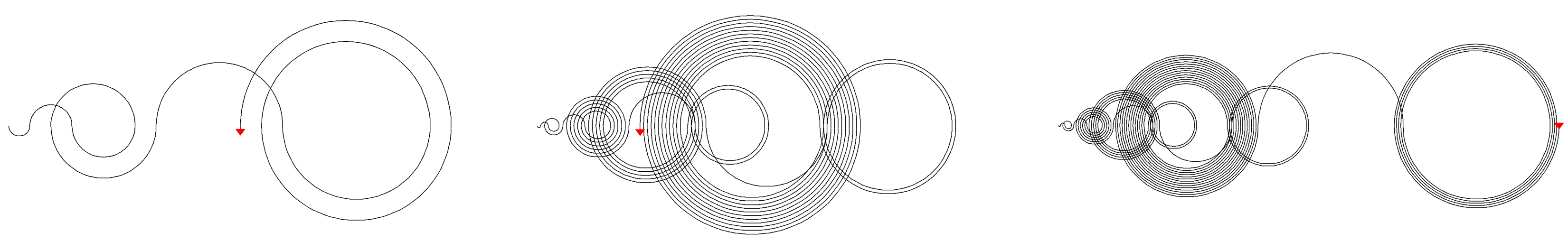
Sample Input
60

Sample Output
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

More Test cases
Input and Output files

Starter Code

The output curve



(a) $n = 10$ (b) $n = 60$ (c) $n = 75$

Figure 17: 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 $[0/1$ 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 18.

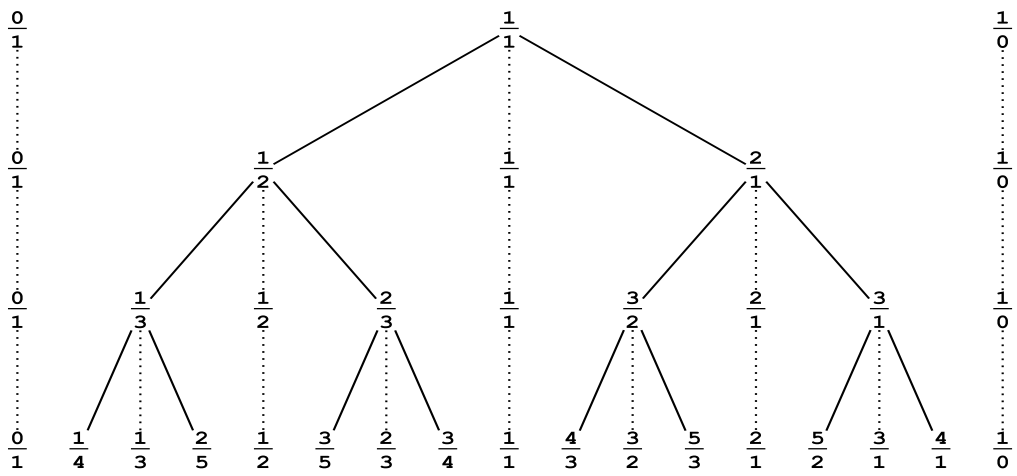


Figure 18: The Stern-Brocot Tree for Level $1-4$

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 – $\left(\frac{0}{1}, \frac{1}{1} \to \frac{1}{2}\right)$, $\left(\frac{1}{1}, \frac{1}{0} \to \frac{2}{1}\right)$, $\left(\frac{0}{1}, \frac{1}{2} \to \frac{1}{3}\right)$, $\left(\frac{1}{2}, \frac{1}{1} \to \frac{2}{3}\right)$, $\left(\frac{1}{1}, \frac{2}{1} \to \frac{3}{2}\right)$, $\left(\frac{2}{1}, \frac{1}{0} \to \frac{3}{1}\right)$,

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

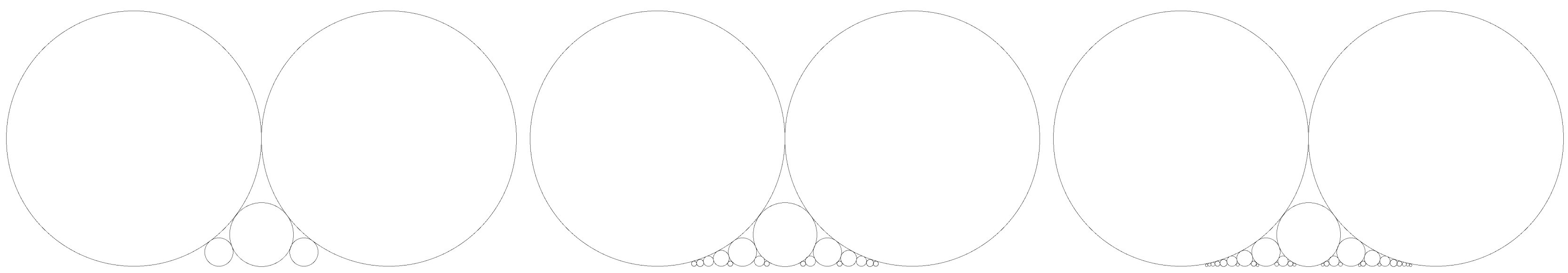
Also, for every fraction $\frac{p}{q}$ in the farey sequence draw a circle with centre at $\left(\frac{p}{q}, \frac{1}{2q^2}\right)$ and radius $\left(\frac{1}{2q^2}\right)$. You may need to do some scaling to get a proper figure.

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

Hint. *Recursion!*

Input Format n	(a single integer)
Output Format Corresponding numbers in farey sequence in p/q format with the circles.	
Constraints $1 \leq n \leq 30$	(an integer)
Sample Input 7	
Sample Output 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	
More Test cases Input and Output files	
Starter Code	

The output circles (Ford Circles)



(a) $n = 3$ (b) $n = 7$ (c) $n = 10$

Figure 19: 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

Topics. *2-D arrays; Function, Recursion with arrays and previous sections.*

§10. Programming Expositions

Topics. *All previous sections.*

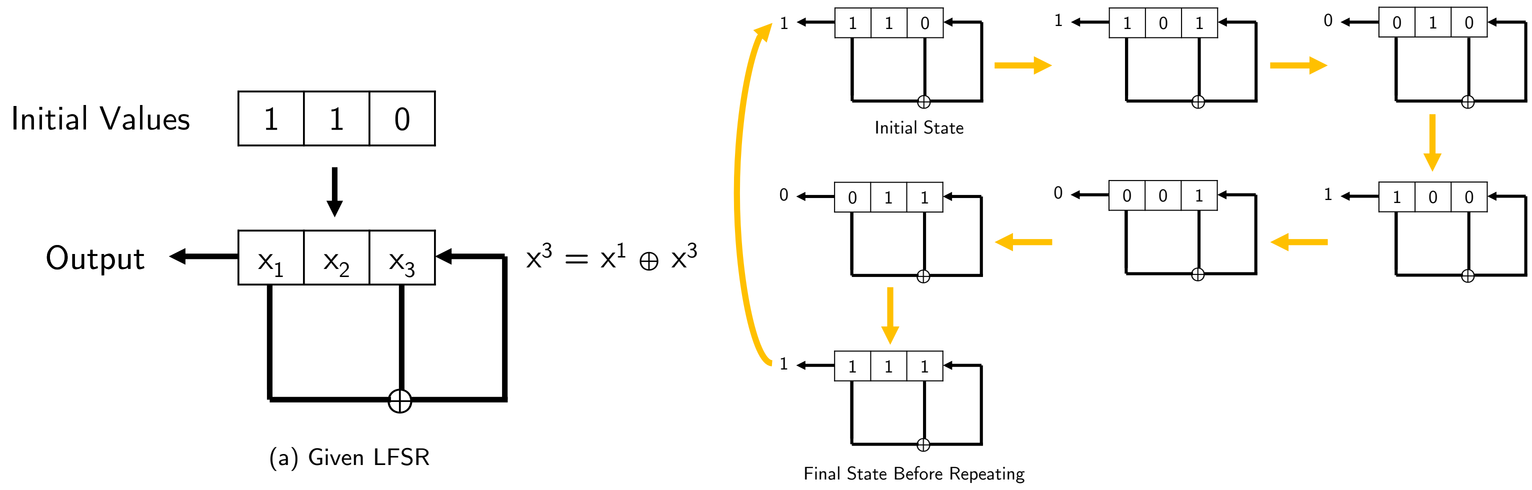
10.1. Linear Feedback Shift Register

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 $x_1, x_2, x_3, \ldots, x_n$ (collectively called as the state of LFSR ("register")) with their initial values (called taps) $t_1, t_2, t_3, \ldots, t_n$ (t_i is 0 or 1).
- A feedback polynomial $c_1 x^0 + c_2 x^1 + c_3 x^2 + \cdots + c_n x^{n-1} + x^n$ (c_i is 0 or 1) which updates the state of LFSR as follows
  - $\text{next}(x_1, x_2, x_3, \ldots, x_{n-1}) = (x_2, x_3, x_4, \ldots, x_n)$ – this is called "shifting" next value of x_1 becomes x_2, next value of x_2 becomes x_3, and so on.
  - $\text{next}(x_n) = c_1 x_1 \oplus c_2 x_2 \oplus \cdots \oplus c_{n-1} x_{n-1} \oplus c_n x_n$ 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 20a. 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 20b. 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 $(1, 1, 1 \oplus 1) = (1, 1, 0)$.



(a) Given LFSR

(b) The Generated Sequence is 11001001 repeating (follow the arrows)

Figure 20: 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[5].
Your task is to simulate an LFSR with a given initial state and feedback polynomial until it repeats and find its period[6] in the process.

Input Format
t (number of test cases, an integer)
$n_i \quad t_1\ t_2\ \cdots t_{n_i} \quad c_1\ c_2\ \cdots c_{n_i}$ ($2n_i + 1$ 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 $c_1 = 1$[7], 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 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

Starter Code

Fun Video. *Random Numbers with LFSR (Linear Feedback Shift Register) – Computerphile*

[5]Interestingly, there also exists a feedback polynomial which achieves this maximum period for every n.
[6]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!
[7]This makes sure that the sequence will repeat from the beginning and will not have any non-periodic part. For example, $110101010\ldots$ ('10' repeating) is not possible if $c_1 = 1$.