More Exercises: Arrays

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](#)

You can check your solutions in [Judge](#)

1. Encrypt, Sort, and Print Array

Write a program that reads a **sequence of strings** from the console. Encrypt every string by summing:

- The code of **each vowel multiplied by the string length**
- The code of **each consonant divided by the string length**

Sort the **number** sequence in ascending order and print it in the console.

On the first line, you will always receive the number of strings you have to read.

Examples

Input	Output	Comments
4 Peter Maria Katya Todor	1032 1071 1168 1532	Peter = 1071 Maria = 1532 Katya = 1032 Todor = 1168
3 Sofia London Washington	1396 1601 3202	Sofia = 1601 London = 1396 Washington = 3202

2. Pascal Triangle

The triangle may be constructed in the following manner: In row 0 (the topmost row), there is a unique nonzero entry 1. Each entry of each subsequent row is constructed by adding the number above and to the left with the number above and to the right, treating blank entries as 0. For example, the initial number in the first (or any other) row is 1 (the sum of 0 and 1), whereas the numbers 1 and 3 in the third row are added to produce the number 4 in the fourth row.

If you want more info about it: https://en.wikipedia.org/wiki/Pascal's_triangle

Print each row element separated with whitespace.

Examples

Input	Output
4	1 1 1 1 2 1 1 3 3 1
13	1 1 1 1 2 1 1 3 3 1 1 4 6 4 1

	1 5 10 10 5 1 1 6 15 20 15 6 1 1 7 21 35 35 21 7 1 1 8 28 56 70 56 28 8 1 1 9 36 84 126 126 84 36 9 1 1 10 45 120 210 252 210 120 45 10 1 1 11 55 165 330 462 462 330 165 55 11 1 1 12 66 220 495 792 924 792 495 220 66 12 1

Hints

- The input number **n** will be **`1 <= n <= 60`**.
- Think about the proper **type** for the elements of the array.
- Don't be scared to use **more and more arrays**.

3. Recursive Fibonacci

The Fibonacci sequence is a quite famous sequence of numbers. Each member of the sequence is calculated from the sum of the two previous elements. The **first two** elements are 1, 1. Therefore the sequence goes like 1, 1, 2, 3, 5, 8, 13, 21, 34…
The following sequence can be generated with an array, but that's easy, so your task is to implement recursively.

So if the function **`GetFibonacci(n)`** returns the n^{th} Fibonacci number we can express it using **`GetFibonacci(n) = GetFibonacci(n-1) + GetFibonacci(n-2)`**.

However, this will never end and in a few seconds, a StackOverflow Exception is thrown. For the recursion to stop, it has to have a "**bottom**". The bottom of the recursion is **`GetFibonacci(2)`** should return 1 and **`GetFibonacci(1)`** should return 1.

Input Format

- On the only line in the input, the user should enter the wanted Fibonacci number.

Output Format

- The output should be the n^{th} Fibonacci number counting from 1.

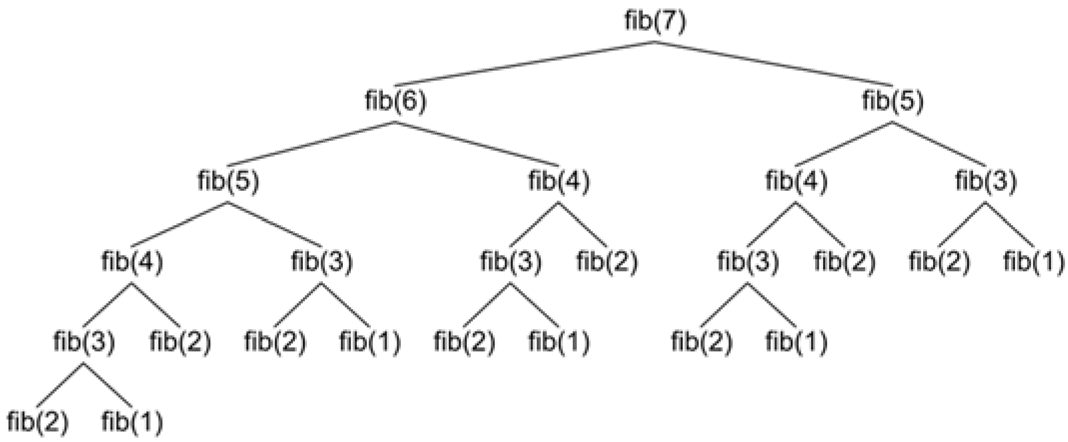
Constraints

- 1 ≤ N ≤ 50

Examples

Input	Output
5	5
10	55
21	10946

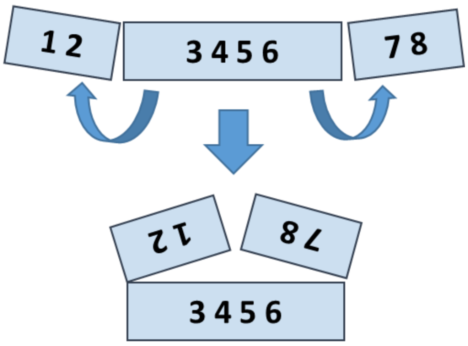
For the N^{th} Fibonacci number, we calculate the $N\text{-}1^{th}$ and the $N\text{-}2^{th}$ number, but for the calculation of the $N\text{-}1^{th}$ number we calculate the $N\text{-}1\text{-}1^{th}$($N\text{-}2^{th}$) and the $N\text{-}1\text{-}2^{th}$ number, so we have a lot of repeated calculations.



If you want to figure out how to skip those unnecessary calculations, you can search for a technique called memoization.

4. Fold and Sum

Read an array of **4*k** integers, fold it like shown below, and print the sum of the upper and lower two rows (each holding 2*k integers):



Examples

Input	Output	Comments
5 **2 3** 6	7 9	5 6 + 2 3 = 7 9
1 2 **3 4 5 6** 7 8	5 5 13 13	2 1 8 7 + 3 4 5 6 = 5 5 13 13
4 3 -1 **2 5 0 1 9 8** 6 7 -2	1 8 4 -1 16 14	-1 3 4 -2 7 6 + 2 5 0 1 9 8 = 1 8 4 -1 16 14

Hints

- Create the **first row** after folding: the first **k** numbers reversed, followed by the last **k** numbers reversed.
- Create the **second row** after folding: the middle **2*k** numbers.
- **Sum** the first and the second rows.

5. Longest Increasing Subsequence (LIS)

Read a **list of integers** and find the **longest increasing subsequence** (LIS). If several such exist, print the **leftmost**.

Examples

Input	Output
`1`	`1`
`7 3 5 8 -1 0 6 7`	`3 5 6 7`
`1 2 5 3 5 2 4 1`	`1 2 3 5`
`0 10 20 30 30 40 1 50 2 3 4 5 6`	`0 1 2 3 4 5 6`
`11 12 13 3 14 4 15 5 6 7 8 7 16 9 8`	`3 4 5 6 7 8 16`
`3 14 5 12 15 7 8 9 11 10 1`	`3 5 7 8 9 11`

Hints

- Assume we have **n** numbers in an array **`nums[0…n-1]`**.
- Let **`len[p]`** hold the length of the longest increasing subsequence (LIS) ending at position **p**.
- In a `for` loop, we shall calculate **`len[p]`** for **p** = **`0`** … **`n-1`** as follows:
    - Let **`left`** be the leftmost position on the left of **p** (**`left`** < **p**), such that **`len[left]`** is the largest possible.
    - Then, **`len[p]`** = **1** + **`len[left]`**. If **`left`** does not exist, **`len[p]`** = **1**.
    - Also, save **`prev[p]`** = **`left`** (we hold in **`prev[]`** the previous position, used to obtain the best length for position **p**).
- Once the values for **`len[0…n-1]`** are calculated, restore the LIS starting from position **p** such that **`len[p]`** is maximal and go back and back through **p** = **`prev[p]`**.
- The table below illustrates these computations:

`index`	0	1	2	3	4	5	6	7	8	9	10
`nums[]`	**3**	**14**	**5**	**12**	**15**	**7**	**8**	**9**	**11**	**10**	**1**
`len[]`	1	2	2	3	4	3	4	5	6	6	1
`prev[]`	-1	0	0	2	3	2	5	6	7	7	-1
`LIS`	{3}	{3,14}	{3,5}	{3,5,12}	{3,5,12,15}	{3,5,7}	{3,5,7,8}	{3,5,7,8,9}	{3,5,7,8,9,11}	{3,5,7,8,9,10}	{1}