2025 Rookie Code Rumble Problems

UNSW CPMSoc

11 June 2025

Contents

Glowing Crossroads

Program time limit: 1 second

Program memory limit: 512 MB

The final rune is entered and the ancient city walls solemnly melt away, the magic that held them together now resolved. Ahead of the entrance is yet another ancient defense, a dark chasm that blocks the path to 3 passageways in the distance.

Elspeth remembers that she can summon a levitating, glowing set of crossroads consisting of a horizontal and vertical road of odd length n, meeting at the center. For $n = 5$ it looks like this.

```
..#..
..#..
#####
..#..
..#..
```

Perhaps this can connect her to the distant passageways?

Input

- The first and only line contains a single odd integer n, the size of the crossroads.

Constraints

For all test cases:

- $1 \leq n \leq 999$.
- n is always odd.

Output

Output the glowing crossroad of length n. The crossroads should consist of '#' symbols, and all empty space should consist of '.'

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```python
# Taking inputs, already done! :D
N = int(input())

# Printing output
print("..#..") # Example top row for n = 5
```

In C or C++, you could use the following code.

```c
// Taking inputs, already done! :D
int N; scanf("%d", &N);

// Printing output
printf("..#..\n"); // Example top row for n = 5
```

Sample Input 1

```
5
```

Sample Output 1

```
..#..
..#..
#####
..#..
..#..
```

Explanation 1

The glowing crossroads from the question, consisting of a horizontal and vertical road of length 5 meeting in the center.

Sample Input 2

```
3
```

Sample Output 2

```
.#.
###
.#.
```

Explanation 2

A smaller set of glowing crossroads of size 3.

Scoring

Your program will be run on the sample cases and all secret cases one after another, and if it produces the correct output for **all** test cases, it solves this task. Recall that your final score on the task is the score of your highest scoring submission.

Deck Dealing

Program time limit: 1 second

Program memory limit: 512 MB

You are given a deck of cards numbered from 1 to N. Initially, the cards are arranged in order: $[1, 2, 3, \ldots, N]$.

Your task is to deal all the cards by performing the following operation exactly N times:

- Choose either the top or the bottom card from the remaining deck,
- Remove that card and add it to the end of your sequence.

How many different sequences can you produce by dealing the cards in this manner?

Constraints

For all test cases:

- $1 \leq N \leq 20$

Input

The first and only line of input contains a single integer N, the number of cards in the deck.

Output

Output a single integer representing the number of different sequences that can be generated.

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```python
# Taking inputs, already done! :D
N = int(input())

ans = 0
# Write your code here


# Printing output
print(ans)
```

In C or C++, you could use the following code.

```cpp
// Taking inputs, already done! :D
int N; scanf("%d", &N);

int ans;
// Write your code here


// Printing output
printf("%d\n", answer);
```

Sample Input 1

```
2
```

Sample Output 1

```
2
```

Explanation 1

For $N = 2$, the initial deck is $[1, 2]$. The possible sequences are: - $[1, 2]$, if you choose the top card first - $[2, 1]$, if you choose the bottom card first

So there are 2 possible sequences.

Sample Input 2

```
3
```

Sample Output 2

```
4
```

Explanation 2

For $N = 3$, the initial deck is $[1, 2, 3]$. The possible sequences are:

- $[1, 2, 3]$
- $[1, 3, 2]$
- $[3, 1, 2]$
- $[3, 2, 1]$

So there are 4 possible sequences.

Scoring

Your program will be run on the 2 sample cases and 18 secret cases one after another, and if it produces the correct output for **all** test cases, it solves this task. Recall that your final score on the task is the score of your highest scoring submission.

Silly Dot Product

Program time limit: 1 second

Program memory limit: 512 MB

Simon is given two arrays A and B, each consisting of N non-negative integers. He can change any integer into any non-negative integer.

What is the least number of changes Simon needs to make in order to make the dot product of A and B equal to zero?

The dot product of two arrays is calculated as the sum of $A_i \times B_i$ for all $1 \leq i \leq N$.

Input

- The first line of input contains an integer N, representing the number of integers in each array.
- The second line contains N space separated integers, representing A.
- The third line contains N space separated integers, representing B.

Constraints

For all test cases:

- $1 \leq N \leq 200,000$,
- $0 \leq A_i, B_i \leq 9$ for $1 \leq i \leq N$.

Additionally:

- For Subtask 1 (50% of points): $1 \leq A_i, B_i$ for $1 \leq i \leq N$ and $N \leq 100$.
- For Subtask 2 (50% of points): there are no additional constraints.

Output

Output a single integer, which represents the least number of changes Simon needs to make.

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```python
# Taking inputs, already done! :D
N = int(input())
A = list(map(int, input().split()))
B = list(map(int, input().split()))

ans = 0
# Write your code here


# Printing output
print(ans)
```

In C or C++, you could use the following code.

```c
// Taking inputs, already done! :D
int N; scanf("%d", &N);
int A[N]; for (int i = 0; i < N; i++) scanf("%d", &A[i]);
int B[N]; for (int i = 0; i < N; i++) scanf("%d", &B[i]);
```

```
int ans;
// Write your code here


// Printing output
printf("%d\n", answer);
```

Sample Input 1

```
6
1 2 3 5 5 6
1 1 1 9 5 4
```

Sample Output 1

```
6
```

Explanation 1

This sample would be an example of subtask 1. One way to make the dot product zero is to change every number in A into 0, and since there are 6 numbers in A, this will require 6 changes. It can be shown that it is not possible to make the dot product zero in fewer changes.

Sample Input 2

```
2
0 0
0 1
```

Sample Output 2

```
0
```

Explanation 2

Since the dot product is already 0, no changes are necessary in this instance.

Scoring

For each subtask (worth 50% and 50% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Quadratic

Program time limit: 1 second

Program memory limit: 512 MB

Elspeth looks up and notices some runes carved into the city wall.

$$(2x - 1)(x + 4)$$

"At last!" she exclaimed. "I knew year 9 maths would come in handy some day!". Her prior knowledge tells her that the expression expands out to

$$2x^2 + 7x - 4$$

but after outputting each coefficient, more factorisations take its place.

Could this be key to unlocking the ancient city? Help Elspeth expand the quadratics.

Input

- The first line contains 4 integers $x_1, c_1, x_2 and c_2$, where
  - x_1 is the coefficient of the first x variable,
  - c_1 is the value of the first constant,
  - x_2 is the coefficient of the second x variable, and
  - c_2 is the value of the second constant.

Constraints

For all test cases:

- $-10,000 \leq x_1, c_1, x_2, c_2 \leq 10,000$.

Output

Ouput each coefficient of the expanded quadratic onto one line separated by spaces (see the samples for reference).

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```
# Taking inputs, already done! :D
x1, c1, x2, c2 = map(int, input().split())

coeff1 = 0
coeff2 = 0
coeff3 = 0
# Write your code here


# Printing output
print(coeff1, coeff2, coeff3)
```

In C or C++, you could use the following code.

```c
// Taking inputs, already done! :D
int N; scanf("%d", &N);
int A[N]; for (int i = 0; i < N; i++) scanf("%d", &A[i]);
int B[N]; for (int i = 0; i < N; i++) scanf("%d", &B[i]);

int coeff1;
int coeff2;
int coeff3;
// Write your code here


// Printing output
printf("%d%d%d\n", coeff1, coeff2, coeff3);
```

Sample Input 1

```
2 -1 1 4
```

Sample Output 1

```
2 7 -4
```

Explanation 1

This input and output corresponds to the example from the question. Expanding $(2x-1)(x+4)$ results in $2x^2+7x-4$.

Sample Input 2

```
1 5 0 2
```

Sample Output 2

```
0 2 10
```

Explanation 2

Some coefficients may be 0. In this case expanding $(x+5)(0x+2)$ results in $2x+10$.

Scoring

Your program will be run on the sample cases and all secret cases one after another, and if it produces the correct output for **all** test cases, it solves this task. Recall that your final score on the task is the score of your highest scoring submission.

Palindromes

Program time limit: 1 second

Program memory limit: 512 MB

Eve loves palindromes, and asks you to help her with a simple problem:

Between 1 and a given integer N, inclusive, how many palindromes are there?

A number is a palindrome if it is the same when the digits are written in reverse order. Note that numbers cannot start with the digit 0.

Input

- The first and only line of input contains an integer N, representing the inclusive upper bound

Constraints

For all test cases:

- $1 \leq N \leq 200,000$.

Additionally:

- For Subtask 1 (50% of points): $N \leq 100$.
- For Subtask 2 (50% of points): there are no additional constraints.

Output

Output a single integer, which represents the number of palindromes between 1 and N.

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```
# Taking inputs, already done! :D
N = int(input())

ans = 0
# Write your code here


# Printing output
print(ans)
```

In C, you can use the following code.

```
// Taking inputs, already done! :D
int N; scanf("%d", &N);

int ans;
// Write your code here


// Printing output
printf("%d\n", answer);
```

Sample Input 1

```
5
```

Sample Output 1

```
5
```

Explanation 1

Any one digit number is automatically a palindrome, so the palindomes that are less than or equal to 5 are $1, 2, 3, 4, 5$.

Sample Input 2

```
13
```

Sample Output 2

```
10
```

Explanation 2

The palindomes less than or equal to 13 are $1, 2, 3, 4, 5, 6, 7, 8, 9, 11$.

Scoring

For each subtask (worth 50% and 50% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Non^4

Program time limit: 1 second

Program memory limit: 512 MB

A non-empty array of numbers must have at least one term.

For an array A containing $N \geq 2$ terms to be non-decreasing, for all $1 \leq i \leq N-1$, $A_i \leq A_{i+1}$.

For an array B containing $N \geq 2$ terms to be non-increasing, for all $1 \leq i \leq N-1$, $A_i \geq A_{i+1}$.

All arrays containing exactly one element are considered as non-decreasing as well as non-increasing.

You are given an array S, consisting of positive integers. How many ways are there to split this array into two parts such that the left part is a non-empty, non-decreasing array A, and the right part is a non-empty, non-increasing array B?

If it is not possible to split S as described, output 0. Otherwise, output the number of different ways S can be split into A and B.

Input

- The first line of input contains an integer N, representing the number of terms in S.
- The second line of input contains N integers which represent array S.

Constraints

For all test cases:

- $2 \leq N \leq 200,000$.
- $1 \leq S_i \leq 100$ for all $1 \leq i \leq N$.

Additionally:

- For Subtask 1 (40% of points): $N \leq 100$ and all values of S are distinct.
- For Subtask 2 (30% of points): $N \leq 100$.
- For Subtask 3 (30% of points): there are no additional constraints.

Output

Output a single integer, which represents the number of ways S can be broken into A and B as described.

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```
# Taking inputs, already done! :D
N = int(input())
S = list(map(int, input().split()))

ans = 0
# Write your code here


# Printing output
print(ans)
```

In C, you can use the following code.

```c
// Taking inputs, already done! :D
int N; scanf("%d", &N);
int S[N]; for (int i = 0; i < N; i++) scanf("%d", &S[i]);

int ans;
// Write your code here


// Printing output
printf("%d\n", answer);
```

Sample Input 1

```
5
1 7 9 3 2
```

Sample Output 1

```
2
```

Explanation 1

This sample would be an example of subtask 1. The two ways to split S are:

- $A = [1, 7, 9], B = [3, 2]$
- $A = [1, 7], B = [9, 3, 2]$

Sample Input 2

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

Sample Output 2

```
0
```

Explanation 2

This sample would be an example of subtask 1. There are no valid ways to split S into A and B.

Sample Input 3

```
3
1 2 2
```

Sample Output 3

```
2
```

Explanation 3

This sample would be an example of subtask 2. The two ways to split S are:

- $A = [1, 2], B = [2]$
- $A = [1], B = [2, 2]$

Sample Input 4

```
2
1 5
```

Sample Output 4

```
1
```

Explanation 4

This sample would be an example of subtask 2. The only way to split S is:

- $A = [1], B = [5]$

Scoring

For each subtask (worth 40%, 30% and 30% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Escape

Program time limit: 1 second

Program memory limit: 512 MB

You are trapped in a rectangular box that extends from $(0, 0)$ to (w, h). You are currently standing at position (a, b) and must escape by touching all four sides of the box, then ending at position (c, d).

Your task is to find the shortest possible path that touches all four sides of the rectangle and ends at the target position. You can move in any direction (horizontally, vertically, or diagonally at any angle).

Note: Touching a corner counts as touching both adjacent sides simultaneously. You can touch the sides at any point along them, not necessarily at integer coordinates.

Input

- The first line of input contains two integers w and h, where (w, h) defines the top-right corner of the box.
- The second line contains two integers a and b, representing your starting position.
- The third line contains two integers c and d, representing your target ending position.

Constraints

For all test cases:

- $1 \leq w, h \leq 10000$.
- $0 \leq a \leq w$ and $0 \leq b \leq h$.
- $0 \leq c \leq w$ and $0 \leq d \leq h$.
- All coordinates are integers.
- The square of the minimum distance is guaranteed to be an integer.

Additionally:

- For Subtask 1 (30% of points): $w = h$, $a = b$, $c = d$.
- For Subtask 2 (30% of points): $a = c$, $b = d$.
- For Subtask 3 (40% of points): there are no additional constraints.

Output

Output a single integer representing the square of the minimum distance you must travel to touch all four sides and reach the target position.

Templates

You should read from standard input and write to standard output.

In Python, you could use the following code.

```
# Taking inputs, already done! :D
w,h = map(int, input().split())
a,b = map(int, input().split())
c,d = map(int, input().split())

ans = 0
# Write your code here


# Printing output
print(ans)
```

In C, you can use the following code.

```c
// Taking inputs, already done! :D
int w, h; scanf("%d%d", &w, &h);
int a, b; scanf("%d%d", &a, &b);
int c, d; scanf("%d%d", &c, &d);

int ans;
// Write your code here


// Printing output
printf("%d\n", answer);
```

Sample Input 1

```
4 4
1 1
2 2
```

Sample Output 1

```
98
```

Explanation 1

For a 4×4 box, starting at $(1, 1)$ and ending at $(2, 2)$, one optimal path is:

- Start at $(1, 1)$
- Move directly to $(0, 0)$ to touch the left and bottom sides simultaneously
- Move directly to $(4, 4)$ to touch the right and top sides simultaneously
- Move directly to $(2, 2)$ to reach the target

The distances are:

- $(1, 1)$ to $(0, 0)$: $\sqrt{(0-1)^2 + (0-1)^2} = \sqrt{1+1} = \sqrt{2}$
- $(0, 0)$ to $(4, 4)$: $\sqrt{(4-0)^2 + (4-0)^2} = \sqrt{16+16} = \sqrt{32} = 4\sqrt{2}$
- $(4, 4)$ to $(2, 2)$: $\sqrt{(2-4)^2 + (2-4)^2} = \sqrt{4+4} = \sqrt{8} = 2\sqrt{2}$

Total distance: $\sqrt{2} + 4\sqrt{2} + 2\sqrt{2} = 7\sqrt{2}$

The total distance squared is 98.

Sample Input 2

```
3 2
1 1
1 0
```

Sample Output 2

```
45
```

Explanation 2

For a 3×2 box, starting at $(1, 1)$ and ending at $(1, 0)$, one optimal path is:

- Start at $(1, 1)$
- Move directly to $(3, 2)$ to touch the right and top sides simultaneously

- Move directly to $(1, 1)$ (back to the origin)
- Move directly to $(0, 0.5)$ to touch the left side (at a non-integer point)
- Move directly to $(1, 0)$ to touch the bottom side and reach the target

The distances are:

- $(1, 1)$ to $(3, 2)$: $\sqrt{(3-1)^2 + (2-1)^2} = \sqrt{4+1} = \sqrt{5}$
- $(3, 2)$ to $(1, 1)$: $\sqrt{(1-3)^2 + (1-2)^2} = \sqrt{4+1} = \sqrt{5}$
- $(1, 1)$ to $(0, 0.5)$: $\sqrt{(0-1)^2 + (0.5-1)^2} = \sqrt{1+0.25} = \sqrt{1.25}$
- $(0, 0.5)$ to $(1, 0)$: $\sqrt{(1-0)^2 + (0-0.5)^2} = \sqrt{1+0.25} = \sqrt{1.25}$

Total distance: $\sqrt{5} + \sqrt{5} + \sqrt{1.25} + \sqrt{1.25} = 2\sqrt{5} + 2\sqrt{1.25} = 3\sqrt{5}$

The total distance squared is 45.

Sample Input 3

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

Sample Output 3

```
74
```

Scoring

For each subtask (worth 30%, 30%, and 40% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Oh no pencils!

Program time limit: 1 seconds

Program memory limit: 512 MB

Oh no, Frank dropped n pencils!

Pencils are segments on a coordinate plane, where each segment is given by two pairs of x, y coordinates, which are their endpoints. Luckily, Frank has a helper robot, which picks up all the pencils in a rectangle of area. Frank isn't that lazy though! So he decides to pick up just one pen.

What is the smallest rectangle of area that the helper bot needs to sweep after Frank picks up one pencil?

Input

- The first line contains one integer n, where n is the number of pencils dropped.
- Then follows n lines , each in the format of: x_1, y_1, x_2, y_2. Specifying the (x, y) coordinates of the two endpoints.

Output

Output a single integer representing the smallest area of the rectangle that is parallel to the x and y axes, the helper robot needs to sweep.

Constraints

For all test cases:

- $1 \leq n \leq 100\ 000$.
- $0 \leq x, y \leq 1\ 000\ 000\ 000$.

Additionally:

- For Subtask 1 (50% of points): $1 \leq n \leq 100$.
- For Subtask 2 (50% of points): No additional constraints.

Sample Input 1

```
1
0 1 1 1
```

Sample Output 1

```
0
```

Explanation 1

Since there is only one pencil, Frank can just pick it up and don't require the helper robot!

Sample Input 2

```
3
1 0 1 5
0 2 5 2
2 0 2 5
```

Sample Output 2

```
5
```

Explanation 2

In this case, we can remove the second line with the coordinates 0 2 5 2, reducing the rectangle area to 5.

Sample Input 3

```
4
1 0 1 5
0 2 5 2
2 0 2 5
0 3 5 3
```

Sample Output 3

```
25
```

Explanation 3

In this case, no matter what line we remove, the area stays to be 25.

Scoring

For each subtask (worth 50% and 50% of points, as per the Constraints section), your program will be run on multiple secret test cases one after another, and if it produces the correct output for **all** test cases, it solves that subtask. Your program will receive the points for each subtask it solves. Recall that your final score on the task is the score of your highest scoring submission.

Unique List

Unique List

Time limit: 1 second
Memory limit: 512 MB

You are given a list of N **distinct non-negative integers**, $A = \{A_1, A_2, \ldots, A_N\}$. Noah and Frank are playing a game using this list.

In each turn, the current player chooses the **largest number** in the list and **replaces it** with a **smaller non-negative integer** that is **not already present** in the list. The list must **remain with all unique elements** after the operation.

The first player who **cannot make a move** on their turn **loses** the game.

If both players play optimally and **Noah starts first**, determine who will win.

Figure 1: unique list

Input

- The first line contains one integer N — the length of the list.
- The second line contains N distinct integers $A_1, A_2, \ldots, A_N$ — representing the elements of the list.

You should read from standard input.

Output

Output the winner's name: either `"Noah"` or `"Frank"` without quotation marks.

You should write to standard output.

Constraints

- $1 \leq N \leq 10^5$
- $0 \leq A_i \leq 10^9$
- $A_i < A_{i+1}$ All elements of the list are **distinct**.

Subtasks:

- **Subtask 1 (10 points):** $A_i \leq 3$
- **Subtask 2 (20 points):** $A_i \leq 4$
- **Subtask 3 (30 points):** $A_i \leq 7$
- **Subtask 4 (10 points):** $N = 2$
- **Subtask 5 (30 points):** No additional constraints

Sample Input 1

```
1
2
```

Sample Output 1

```
Noah
```

Explanation 1

Here, Noah can change the only number to 0 on his turn.

Sample Input 2

```
2
2 3
```

Sample Output 2

```
Frank
```

Explanation 2

Here, no matter no matter which number Noah changes 3 to, Frank can always pick 2 and make the remaining list to `0 1`. Thus, Noah always loses in this case.