AVAILABLE LESSONS:

|  |  |
| --- | --- |
| *[Lesson 1](https://codility.com/programmers/lessons/1-iterations/)*  [Iterations](https://codility.com/programmers/lessons/1-iterations/)  *[Lesson 2](https://codility.com/programmers/lessons/2-arrays/)*  [Arrays](https://codility.com/programmers/lessons/2-arrays/)  *[Lesson 3](https://codility.com/programmers/lessons/3-time_complexity/)*  [Time Complexity](https://codility.com/programmers/lessons/3-time_complexity/)  *[Lesson 4](https://codility.com/programmers/lessons/4-counting_elements/)*  [Counting Elements](https://codility.com/programmers/lessons/4-counting_elements/)  *[Lesson 5](https://codility.com/programmers/lessons/5-prefix_sums/)*  [Prefix Sums](https://codility.com/programmers/lessons/5-prefix_sums/)  *[Lesson 6](https://codility.com/programmers/lessons/6-sorting/)*  [Sorting](https://codility.com/programmers/lessons/6-sorting/)  *[Lesson 7](https://codility.com/programmers/lessons/7-stacks_and_queues/)*  [Stacks and Queues](https://codility.com/programmers/lessons/7-stacks_and_queues/)  *[Lesson 8](https://codility.com/programmers/lessons/8-leader/)*  [Leader](https://codility.com/programmers/lessons/8-leader/)  *[Lesson 9](https://codility.com/programmers/lessons/9-maximum_slice_problem/)*  [Maximum slice problem](https://codility.com/programmers/lessons/9-maximum_slice_problem/)  *[Lesson 10](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/)*  [Prime and composite numbers](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/) | *[Lesson 11](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/)*  [Sieve of Eratosthenes](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/)  *[Lesson 12](https://codility.com/programmers/lessons/12-euclidean_algorithm/)*  [Euclidean algorithm](https://codility.com/programmers/lessons/12-euclidean_algorithm/)  *[Lesson 13](https://codility.com/programmers/lessons/13-fibonacci_numbers/)*  [Fibonacci numbers](https://codility.com/programmers/lessons/13-fibonacci_numbers/)  *[Lesson 14](https://codility.com/programmers/lessons/14-binary_search_algorithm/)*  [Binary search algorithm](https://codility.com/programmers/lessons/14-binary_search_algorithm/)  *[Lesson 15](https://codility.com/programmers/lessons/15-caterpillar_method/)*  [Caterpillar method](https://codility.com/programmers/lessons/15-caterpillar_method/)  *[Lesson 16](https://codility.com/programmers/lessons/16-greedy_algorithms/)*  [Greedy algorithms](https://codility.com/programmers/lessons/16-greedy_algorithms/)  *[Lesson 17](https://codility.com/programmers/lessons/17-dynamic_programming/)*  [Dynamic programming](https://codility.com/programmers/lessons/17-dynamic_programming/)  *[Lesson 90](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/)*  [Tasks from Indeed Prime 2015 challenge](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/)  *[Lesson 91](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/)*  [Tasks from Indeed Prime 2016 challenge](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/)  *[Lesson 92](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/)*  [Tasks from Indeed Prime 2016 College Coders challenge](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/)  *[Lesson 99](https://codility.com/programmers/lessons/99-future_training/)*  [Future training](https://codility.com/programmers/lessons/99-future_training/) |

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Codility Solutions

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// https://www.martinkysel.com/codility-tape-equilibrium-solution/

// https://www.martinkysel.com/codility-solutions/

// https://codesays.com/solutions-to-training-by-codility/

*Lesson 1*

## Iterations

[BinaryGap](https://codility.com/programmers/lessons/1-iterations/binary_gap/)

Find longest sequence of zeros in binary representation of an integer.

A *binary gap* within a positive integer N is any maximal sequence of consecutive zeros that is surrounded by ones at both ends in the binary representation of N.

For example, number 9 has binary representation 1001 and contains a binary gap of length 2.

The number 529 has binary representation 1000010001 and contains two binary gaps: one of length 4 and one of length 3.

The number 20 has binary representation 10100 and contains one binary gap of length 1.

The number 15 has binary representation 1111 and has no binary gaps.

Write a function:

class Solution { public int solution(int N); }

that, given a positive integer N, returns the length of **its longest** binary gap.

The function should return 0 if N doesn't contain a binary gap.

For example, given N = 1041 the function should return 5, because N has binary representation 10000010001 and so its longest binary gap is of length 5.

Given N = 32 the function should return 0, because N has binary representation '100000' and thus no binary gaps.

Write an efficient algorithm for the following assumptions:

Assume that:

* N is an integer within the range [1..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(log(N));
* expected worst-case space complexity is O(1).

*Lesson 2*

## Arrays

#### [OddOccurrencesInArray](https://codility.com/programmers/lessons/2-arrays/odd_occurrences_in_array/)

Find value that occurs in odd number of elements.

A non-empty zero-indexed array A consisting of N integers is given. The array contains an odd number of elements, and each element of the array can be paired with another element that has the same value, except for one element that is left unpaired.

For example, in array A such that:

A[0] = 9 A[1] = 3 A[2] = 9 A[3] = 3 A[4] = 9 A[5] = 7 A[6] = 9

* the elements at indexes 0 and 2 have value 9,
* the elements at indexes 1 and 3 have value 3,
* the elements at indexes 4 and 6 have value 9,
* the element at index 5 has value 7 and is unpaired.

Write a function:

class Solution { public int solution(int[] A); }

that, given an array A consisting of N integers fulfilling the above conditions, returns the value of the unpaired element.

For example, given array A such that:

A[0] = 9 A[1] = 3 A[2] = 9 A[3] = 3 A[4] = 9 A[5] = 7 A[6] = 9

the function should return 7, as explained in the example above.

Assume that:

* N is an odd integer within the range [1..1,000,000];
* each element of array A is an integer within the range [1..1,000,000,000];
* all but one of the values in A occur an even number of times.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

#### [CyclicRotation](https://codility.com/programmers/lessons/2-arrays/cyclic_rotation/)

[**START**](https://codility.com/programmers/lessons/2-arrays/cyclic_rotation/start/)

Rotate an array to the right by a given number of steps.

A zero-indexed array A consisting of N integers is given. Rotation of the array means that each element is shifted right by one index, and the last element of the array is also moved to the first place.

For example, the rotation of array A = [3, 8, 9, 7, 6] is [6, 3, 8, 9, 7]. The goal is to rotate array A K times; that is, each element of A will be shifted to the right by K indexes.

Write a function:

class Solution { public int[] solution(int[] A, int K); }

that, given a zero-indexed array A consisting of N integers and an integer K, returns the array A rotated K times.

For example, given array A = [3, 8, 9, 7, 6] and K = 3, the function should return [9, 7, 6, 3, 8].

Assume that:

* N and K are integers within the range [0..100];
* each element of array A is an integer within the range [−1,000..1,000].

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

*Lesson 3*

## Time Complexity

#### [PermMissingElem](https://codility.com/programmers/lessons/3-time_complexity/perm_missing_elem/)

**[START](https://codility.com/programmers/lessons/3-time_complexity/perm_missing_elem/start/)**

Find the missing element in a given permutation.

A zero-indexed array A consisting of N different integers is given. The array contains integers in the range [1..(N + 1)], which means that exactly one element is missing.

Your goal is to find that missing element.

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A, returns the value of the missing element.

For example, given array A such that:

A[0] = 2 A[1] = 3 A[2] = 1 A[3] = 5

the function should return 4, as it is the missing element.

Assume that:

* N is an integer within the range [0..100,000];
* the elements of A are all distinct;
* each element of array A is an integer within the range [1..(N + 1)].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

#### [FrogJmp](https://codility.com/programmers/lessons/3-time_complexity/frog_jmp/)

[**START**](https://codility.com/programmers/lessons/3-time_complexity/frog_jmp/start/)

Count minimal number of jumps from position X to Y.

A small frog wants to get to the other side of the road. The frog is currently located at position X and wants to get to a position greater than or equal to Y. The small frog always jumps a fixed distance, D.

Count the minimal number of jumps that the small frog must perform to reach its target.

Write a function:

class Solution { public int solution(int X, int Y, int D); }

that, given three integers X, Y and D, returns the minimal number of jumps from position X to a position equal to or greater than Y.

For example, given:

X = 10 Y = 85 D = 30

the function should return 3, because the frog will be positioned as follows:

* after the first jump, at position 10 + 30 = 40
* after the second jump, at position 10 + 30 + 30 = 70
* after the third jump, at position 10 + 30 + 30 + 30 = 100

Assume that:

* X, Y and D are integers within the range [1..1,000,000,000];
* X ≤ Y.

Complexity:

* expected worst-case time complexity is O(1);
* expected worst-case space complexity is O(1).

#### [TapeEquilibrium](https://codility.com/programmers/lessons/3-time_complexity/tape_equilibrium/)

[**START**](https://codility.com/programmers/lessons/3-time_complexity/tape_equilibrium/start/)

Minimize the value |(A[0] + ... + A[P-1]) - (A[P] + ... + A[N-1])|.

A non-empty zero-indexed array A consisting of N integers is given. Array A represents numbers on a tape.

Any integer P, such that 0 < P < N, splits this tape into two non-empty parts: A[0], A[1], ..., A[P − 1] and A[P], A[P + 1], ..., A[N − 1].

The *difference* between the two parts is the value of: |(A[0] + A[1] + ... + A[P − 1]) − (A[P] + A[P + 1] + ... + A[N − 1])|

In other words, it is the absolute difference between the sum of the first part and the sum of the second part.

For example, consider array A such that:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 4 A[4] = 3

We can split this tape in four places:

* P = 1, difference = |3 − 10| = 7
* P = 2, difference = |4 − 9| = 5
* P = 3, difference = |6 − 7| = 1
* P = 4, difference = |10 − 3| = 7

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A of N integers, returns the minimal difference that can be achieved.

For example, given:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 4 A[4] = 3

the function should return 1, as explained above.

Assume that:

* N is an integer within the range [2..100,000];
* each element of array A is an integer within the range [−1,000..1,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 4*

## Counting Elements

#### [PermCheck](https://codility.com/programmers/lessons/4-counting_elements/perm_check/)

[**START**](https://codility.com/programmers/lessons/4-counting_elements/perm_check/start/)

Check whether array A is a permutation.

A non-empty zero-indexed array A consisting of N integers is given.

A *permutation* is a sequence containing each element from 1 to N once, and only once.

For example, array A such that:

A[0] = 4 A[1] = 1 A[2] = 3 A[3] = 2

is a permutation, but array A such that:

A[0] = 4 A[1] = 1 A[2] = 3

is not a permutation, because value 2 is missing.

The goal is to check whether array A is a permutation.

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A, returns 1 if array A is a permutation and 0 if it is not.

For example, given array A such that:

A[0] = 4 A[1] = 1 A[2] = 3 A[3] = 2

the function should return 1.

Given array A such that:

A[0] = 4 A[1] = 1 A[2] = 3

the function should return 0.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

#### [FrogRiverOne](https://codility.com/programmers/lessons/4-counting_elements/frog_river_one/)

[**START**](https://codility.com/programmers/lessons/4-counting_elements/frog_river_one/start/)

Find the earliest time when a frog can jump to the other side of a river.

 small frog wants to get to the other side of a river. The frog is initially located on one bank of the river (position 0) and wants to get to the opposite bank (position X+1). Leaves fall from a tree onto the surface of the river.

You are given a zero-indexed array A consisting of N integers representing the falling leaves. A[K] represents the position where one leaf falls at time K, measured in seconds.

The goal is to find the earliest time when the frog can jump to the other side of the river. The frog can cross only when leaves appear at every position across the river from 1 to X (that is, we want to find the earliest moment when all the positions from 1 to X are covered by leaves). You may assume that the speed of the current in the river is negligibly small, i.e. the leaves do not change their positions once they fall in the river.

For example, you are given integer X = 5 and array A such that:

A[0] = 1 A[1] = 3 A[2] = 1 A[3] = 4 A[4] = 2 A[5] = 3 A[6] = 5 A[7] = 4

In second 6, a leaf falls into position 5. This is the earliest time when leaves appear in every position across the river.

Write a function:

class Solution { public int solution(int X, int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers and integer X, returns the earliest time when the frog can jump to the other side of the river.

If the frog is never able to jump to the other side of the river, the function should return −1.

For example, given X = 5 and array A such that:

A[0] = 1 A[1] = 3 A[2] = 1 A[3] = 4 A[4] = 2 A[5] = 3 A[6] = 5 A[7] = 4

the function should return 6, as explained above.

Assume that:

* N and X are integers within the range [1..100,000];
* each element of array A is an integer within the range [1..X].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(X), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

#### [MissingInteger](https://codility.com/programmers/lessons/4-counting_elements/missing_integer/)

[**START**](https://codility.com/programmers/lessons/4-counting_elements/missing_integer/start/)

Find the smallest positive integer that does not occur in a given sequence.

This is a demo task.

Write a function:

class Solution { public int solution(int[] A); }

that, given an array A of N integers, returns the smallest positive integer (greater than 0) that does not occur in A.

For example, given A = [1, 3, 6, 4, 1, 2], the function should return 5.

For another example, given A = [1, 2, 3], the function should return 4.

Given A = [−1, −3], the function should return 1.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [−1,000,000..1,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

#### [MaxCounters](https://codility.com/programmers/lessons/4-counting_elements/max_counters/)

[**START**](https://codility.com/programmers/lessons/4-counting_elements/max_counters/start/)

Calculate the values of counters after applying all alternating operations: increase counter by 1; set value of all counters to current maximum.

You are given N counters, initially set to 0, and you have two possible operations on them:

* *increase(X)* − counter X is increased by 1,
* *max counter* − all counters are set to the maximum value of any counter.

A non-empty zero-indexed array A of M integers is given. This array represents consecutive operations:

* if A[K] = X, such that 1 ≤ X ≤ N, then operation K is increase(X),
* if A[K] = N + 1 then operation K is max counter.

For example, given integer N = 5 and array A such that:

A[0] = 3 A[1] = 4 A[2] = 4 A[3] = 6 A[4] = 1 A[5] = 4 A[6] = 4

the values of the counters after each consecutive operation will be:

(0, 0, 1, 0, 0) (0, 0, 1, 1, 0) (0, 0, 1, 2, 0) (2, 2, 2, 2, 2) (3, 2, 2, 2, 2) (3, 2, 2, 3, 2) (3, 2, 2, 4, 2)

The goal is to calculate the value of every counter after all operations.

Write a function:

class Solution { public int[] solution(int N, int[] A); }

that, given an integer N and a non-empty zero-indexed array A consisting of M integers, returns a sequence of integers representing the values of the counters.

The sequence should be returned as:

* a structure Results (in C), or
* a vector of integers (in C++), or
* a record Results (in Pascal), or
* an array of integers (in any other programming language).

For example, given:

A[0] = 3 A[1] = 4 A[2] = 4 A[3] = 6 A[4] = 1 A[5] = 4 A[6] = 4

the function should return [3, 2, 2, 4, 2], as explained above.

Assume that:

* N and M are integers within the range [1..100,000];
* each element of array A is an integer within the range [1..N + 1].

Complexity:

* expected worst-case time complexity is O(N+M);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 5*

## Prefix Sums

[CountDiv](https://codility.com/programmers/lessons/5-prefix_sums/count_div/)

[**START**](https://codility.com/programmers/lessons/5-prefix_sums/count_div/start/)

Compute number of integers divisible by k in range [a..b].

Write a function:

class Solution { public int solution(int A, int B, int K); }

that, given three integers A, B and K, returns the number of integers within the range [A..B] that are divisible by K, i.e.:

{ i : A ≤ i ≤ B, i mod K = 0 }

For example, for A = 6, B = 11 and K = 2, your function should return 3, because there are three numbers divisible by 2 within the range [6..11], namely 6, 8 and 10.

Assume that:

* A and B are integers within the range [0..2,000,000,000];
* K is an integer within the range [1..2,000,000,000];
* A ≤ B.

Complexity:

* expected worst-case time complexity is O(1);
* expected worst-case space complexity is O(1).

[PassingCars](https://codility.com/programmers/lessons/5-prefix_sums/passing_cars/)

[**START**](https://codility.com/programmers/lessons/5-prefix_sums/passing_cars/start/)

Count the number of passing cars on the road.

A non-empty zero-indexed array A consisting of N integers is given. The consecutive elements of array A represent consecutive cars on a road.

Array A contains only 0s and/or 1s:

* 0 represents a car traveling east,
* 1 represents a car traveling west.

The goal is to count passing cars. We say that a pair of cars (P, Q), where 0 ≤ P < Q < N, is passing when P is traveling to the east and Q is traveling to the west.

For example, consider array A such that:

A[0] = 0 A[1] = 1 A[2] = 0 A[3] = 1 A[4] = 1

We have five pairs of passing cars: (0, 1), (0, 3), (0, 4), (2, 3), (2, 4).

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A of N integers, returns the number of pairs of passing cars.

The function should return −1 if the number of pairs of passing cars exceeds 1,000,000,000.

For example, given:

A[0] = 0 A[1] = 1 A[2] = 0 A[3] = 1 A[4] = 1

the function should return 5, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer that can have one of the following values: 0, 1.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[GenomicRangeQuery](https://codility.com/programmers/lessons/5-prefix_sums/genomic_range_query/)

[**START**](https://codility.com/programmers/lessons/5-prefix_sums/genomic_range_query/start/)

Find the minimal nucleotide from a range of sequence DNA.

A DNA sequence can be represented as a string consisting of the letters A, C, G and T, which correspond to the types of successive nucleotides in the sequence. Each nucleotide has an *impact factor*, which is an integer. Nucleotides of types A, C, G and T have impact factors of 1, 2, 3 and 4, respectively. You are going to answer several queries of the form: What is the minimal impact factor of nucleotides contained in a particular part of the given DNA sequence?

The DNA sequence is given as a non-empty string S = S[0]S[1]...S[N-1] consisting of N characters. There are M queries, which are given in non-empty arrays P and Q, each consisting of M integers. The K-th query (0 ≤ K < M) requires you to find the minimal impact factor of nucleotides contained in the DNA sequence between positions P[K] and Q[K] (inclusive).

For example, consider string S = CAGCCTA and arrays P, Q such that:

P[0] = 2 Q[0] = 4 P[1] = 5 Q[1] = 5 P[2] = 0 Q[2] = 6

The answers to these M = 3 queries are as follows:

* The part of the DNA between positions 2 and 4 contains nucleotides G and C (twice), whose impact factors are 3 and 2 respectively, so the answer is 2.
* The part between positions 5 and 5 contains a single nucleotide T, whose impact factor is 4, so the answer is 4.
* The part between positions 0 and 6 (the whole string) contains all nucleotides, in particular nucleotide A whose impact factor is 1, so the answer is 1.

Write a function:

class Solution { public int[] solution(string S, int[] P, int[] Q); }

that, given a non-empty zero-indexed string S consisting of N characters and two non-empty zero-indexed arrays P and Q consisting of M integers, returns an array consisting of M integers specifying the consecutive answers to all queries.

The sequence should be returned as:

* a Results structure (in C), or
* a vector of integers (in C++), or
* a Results record (in Pascal), or
* an array of integers (in any other programming language).

For example, given the string S = CAGCCTA and arrays P, Q such that:

P[0] = 2 Q[0] = 4 P[1] = 5 Q[1] = 5 P[2] = 0 Q[2] = 6

the function should return the values [2, 4, 1], as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* M is an integer within the range [1..50,000];
* each element of arrays P, Q is an integer within the range [0..N − 1];
* P[K] ≤ Q[K], where 0 ≤ K < M;
* string S consists only of upper-case English letters A, C, G, T.

Complexity:

* expected worst-case time complexity is O(N+M);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[MinAvgTwoSlice](https://codility.com/programmers/lessons/5-prefix_sums/min_avg_two_slice/)

[**START**](https://codility.com/programmers/lessons/5-prefix_sums/min_avg_two_slice/start/)

Find the minimal average of any slice containing at least two elements.

A non-empty zero-indexed array A consisting of N integers is given. A pair of integers (P, Q), such that 0 ≤ P < Q < N, is called a *slice* of array A (notice that the slice contains at least two elements). The *average* of a slice (P, Q) is the sum of A[P] + A[P + 1] + ... + A[Q] divided by the length of the slice. To be precise, the average equals (A[P] + A[P + 1] + ... + A[Q]) / (Q − P + 1).

For example, array A such that:

A[0] = 4 A[1] = 2 A[2] = 2 A[3] = 5 A[4] = 1 A[5] = 5 A[6] = 8

contains the following example slices:

* slice (1, 2), whose average is (2 + 2) / 2 = 2;
* slice (3, 4), whose average is (5 + 1) / 2 = 3;
* slice (1, 4), whose average is (2 + 2 + 5 + 1) / 4 = 2.5.

The goal is to find the starting position of a slice whose average is minimal.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the starting position of the slice with the minimal average. If there is more than one slice with a minimal average, you should return the smallest starting position of such a slice.

For example, given array A such that:

A[0] = 4 A[1] = 2 A[2] = 2 A[3] = 5 A[4] = 1 A[5] = 5 A[6] = 8

the function should return 1, as explained above.

Assume that:

* N is an integer within the range [2..100,000];
* each element of array A is an integer within the range [−10,000..10,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 6*

## Sorting

[Distinct](https://codility.com/programmers/lessons/6-sorting/distinct/)

[**START**](https://codility.com/programmers/lessons/6-sorting/distinct/start/)

Compute number of distinct values in an array.

Write a function

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers, returns the number of distinct values in array A.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer within the range [−1,000,000..1,000,000].

For example, given array A consisting of six elements such that:

A[0] = 2 A[1] = 1 A[2] = 1 A[3] = 2 A[4] = 3 A[5] = 1

the function should return 3, because there are 3 distinct values appearing in array A, namely 1, 2 and 3.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[Triangle](https://codility.com/programmers/lessons/6-sorting/triangle/)

[**START**](https://codility.com/programmers/lessons/6-sorting/triangle/start/)

Determine whether a triangle can be built from a given set of edges.

A zero-indexed array A consisting of N integers is given. A triplet (P, Q, R) is *triangular* if 0 ≤ P < Q < R < N and:

* A[P] + A[Q] > A[R],
* A[Q] + A[R] > A[P],
* A[R] + A[P] > A[Q].

For example, consider array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 20

Triplet (0, 2, 4) is triangular.

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers, returns 1 if there exists a triangular triplet for this array and returns 0 otherwise.

For example, given array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 20

the function should return 1, as explained above. Given array A such that:

A[0] = 10 A[1] = 50 A[2] = 5 A[3] = 1

the function should return 0.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[MaxProductOfThree](https://codility.com/programmers/lessons/6-sorting/max_product_of_three/)

[**START**](https://codility.com/programmers/lessons/6-sorting/max_product_of_three/start/)

Maximize A[P] \* A[Q] \* A[R] for any triplet (P, Q, R).

A non-empty zero-indexed array A consisting of N integers is given. The *product* of triplet (P, Q, R) equates to A[P] \* A[Q] \* A[R] (0 ≤ P < Q < R < N).

For example, array A such that:

A[0] = -3 A[1] = 1 A[2] = 2 A[3] = -2 A[4] = 5 A[5] = 6

contains the following example triplets:

* (0, 1, 2), product is −3 \* 1 \* 2 = −6
* (1, 2, 4), product is 1 \* 2 \* 5 = 10
* (2, 4, 5), product is 2 \* 5 \* 6 = 60

Your goal is to find the maximal product of any triplet.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A, returns the value of the maximal product of any triplet.

For example, given array A such that:

A[0] = -3 A[1] = 1 A[2] = 2 A[3] = -2 A[4] = 5 A[5] = 6

the function should return 60, as the product of triplet (2, 4, 5) is maximal.

Assume that:

* N is an integer within the range [3..100,000];
* each element of array A is an integer within the range [−1,000..1,000].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[NumberOfDiscIntersections](https://codility.com/programmers/lessons/6-sorting/number_of_disc_intersections/)

[**START**](https://codility.com/programmers/lessons/6-sorting/number_of_disc_intersections/start/)

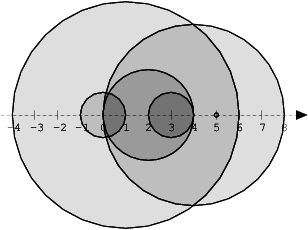
Compute the number of intersections in a sequence of discs.

We draw N discs on a plane. The discs are numbered from 0 to N − 1. A zero-indexed array A of N non-negative integers, specifying the radiuses of the discs, is given. The J-th disc is drawn with its center at (J, 0) and radius A[J].

We say that the J-th disc and K-th disc intersect if J ≠ K and the J-th and K-th discs have at least one common point (assuming that the discs contain their borders).

The figure below shows discs drawn for N = 6 and A as follows:

A[0] = 1 A[1] = 5 A[2] = 2 A[3] = 1 A[4] = 4 A[5] = 0



There are eleven (unordered) pairs of discs that intersect, namely:

* discs 1 and 4 intersect, and both intersect with all the other discs;
* disc 2 also intersects with discs 0 and 3.

Write a function:

class Solution { public int solution(int[] A); }

that, given an array A describing N discs as explained above, returns the number of (unordered) pairs of intersecting discs. The function should return −1 if the number of intersecting pairs exceeds 10,000,000.

Given array A shown above, the function should return 11, as explained above.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer within the range [0..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 7*

## Stacks and Queues

[StoneWall](https://codility.com/programmers/lessons/7-stacks_and_queues/stone_wall/)

[**START**](https://codility.com/programmers/lessons/7-stacks_and_queues/stone_wall/start/)

Cover "Manhattan skyline" using the minimum number of rectangles.

You are going to build a stone wall. The wall should be straight and N meters long, and its thickness should be constant; however, it should have different heights in different places. The height of the wall is specified by a zero-indexed array H of N positive integers. H[I] is the height of the wall from I to I+1 meters to the right of its left end. In particular, H[0] is the height of the wall's left end and H[N−1] is the height of the wall's right end.

The wall should be built of cuboid stone blocks (that is, all sides of such blocks are rectangular). Your task is to compute the minimum number of blocks needed to build the wall.

Write a function:

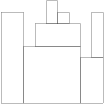
class Solution { public int solution(int[] H); }

that, given a zero-indexed array H of N positive integers specifying the height of the wall, returns the minimum number of blocks needed to build it.

For example, given array H containing N = 9 integers:

H[0] = 8 H[1] = 8 H[2] = 5 H[3] = 7 H[4] = 9 H[5] = 8 H[6] = 7 H[7] = 4 H[8] = 8

the function should return 7. The figure shows one possible arrangement of seven blocks.



Assume that:

* N is an integer within the range [1..100,000];
* each element of array H is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[Brackets](https://codility.com/programmers/lessons/7-stacks_and_queues/brackets/)

[**START**](https://codility.com/programmers/lessons/7-stacks_and_queues/brackets/start/)

Determine whether a given string of parentheses is properly nested.

A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

* S is empty;
* S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string;
* S has the form "VW" where V and W are properly nested strings.

For example, the string "{[()()]}" is properly nested but "([)()]" is not.

Write a function:

class Solution { public int solution(string S); }

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.

Assume that:

* N is an integer within the range [0..200,000];
* string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N) (not counting the storage required for input arguments).

[Nesting](https://codility.com/programmers/lessons/7-stacks_and_queues/nesting/)

[**START**](https://codility.com/programmers/lessons/7-stacks_and_queues/nesting/start/)

Determine whether given string of parentheses is properly nested.

A string S consisting of N characters is called *properly nested* if:

* S is empty;
* S has the form "(U)" where U is a properly nested string;
* S has the form "VW" where V and W are properly nested strings.

For example, string "(()(())())" is properly nested but string "())" isn't.

Write a function:

class Solution { public int solution(string S); }

that, given a string S consisting of N characters, returns 1 if string S is properly nested and 0 otherwise.

For example, given S = "(()(())())", the function should return 1 and given S = "())", the function should return 0, as explained above.

Assume that:

* N is an integer within the range [0..1,000,000];
* string S consists only of the characters "(" and/or ")".

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1) (not counting the storage required for input arguments).

[Fish](https://codility.com/programmers/lessons/7-stacks_and_queues/fish/)

[**START**](https://codility.com/programmers/lessons/7-stacks_and_queues/fish/start/)

N voracious fish are moving along a river. Calculate how many fish are alive.

You are given two non-empty zero-indexed arrays A and B consisting of N integers. Arrays A and B represent N voracious fish in a river, ordered downstream along the flow of the river.

The fish are numbered from 0 to N − 1. If P and Q are two fish and P < Q, then fish P is initially upstream of fish Q. Initially, each fish has a unique position.

Fish number P is represented by A[P] and B[P]. Array A contains the sizes of the fish. All its elements are unique. Array B contains the directions of the fish. It contains only 0s and/or 1s, where:

* 0 represents a fish flowing upstream,
* 1 represents a fish flowing downstream.

If two fish move in opposite directions and there are no other (living) fish between them, they will eventually meet each other. Then only one fish can stay alive − the larger fish eats the smaller one. More precisely, we say that two fish P and Q meet each other when P < Q, B[P] = 1 and B[Q] = 0, and there are no living fish between them. After they meet:

* If A[P] > A[Q] then P eats Q, and P will still be flowing downstream,
* If A[Q] > A[P] then Q eats P, and Q will still be flowing upstream.

We assume that all the fish are flowing at the same speed. That is, fish moving in the same direction never meet. The goal is to calculate the number of fish that will stay alive.

For example, consider arrays A and B such that:

A[0] = 4 B[0] = 0 A[1] = 3 B[1] = 1 A[2] = 2 B[2] = 0 A[3] = 1 B[3] = 0 A[4] = 5 B[4] = 0

Initially all the fish are alive and all except fish number 1 are moving upstream. Fish number 1 meets fish number 2 and eats it, then it meets fish number 3 and eats it too. Finally, it meets fish number 4 and is eaten by it. The remaining two fish, number 0 and 4, never meet and therefore stay alive.

Write a function:

class Solution { public int solution(int[] A, int[] B); }

that, given two non-empty zero-indexed arrays A and B consisting of N integers, returns the number of fish that will stay alive.

For example, given the arrays shown above, the function should return 2, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [0..1,000,000,000];
* each element of array B is an integer that can have one of the following values: 0, 1;
* the elements of A are all distinct.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 8*

## Leader

[EquiLeader](https://codility.com/programmers/lessons/8-leader/equi_leader/)

[**START**](https://codility.com/programmers/lessons/8-leader/equi_leader/start/)

Find the index S such that the leaders of the sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] are the same.

A non-empty zero-indexed array A consisting of N integers is given.

The *leader* of this array is the value that occurs in more than half of the elements of A.

An *equi leader* is an index S such that 0 ≤ S < N − 1 and two sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N − 1] have leaders of the same value.

For example, given array A such that:

A[0] = 4 A[1] = 3 A[2] = 4 A[3] = 4 A[4] = 4 A[5] = 2

we can find two equi leaders:

* 0, because sequences: (4) and (3, 4, 4, 4, 2) have the same leader, whose value is 4.
* 2, because sequences: (4, 3, 4) and (4, 4, 2) have the same leader, whose value is 4.

The goal is to count the number of equi leaders.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the number of equi leaders.

For example, given:

A[0] = 4 A[1] = 3 A[2] = 4 A[3] = 4 A[4] = 4 A[5] = 2

the function should return 2, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [−1,000,000,000..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[Dominator](https://codility.com/programmers/lessons/8-leader/dominator/)

[**START**](https://codility.com/programmers/lessons/8-leader/dominator/start/)

Find an index of an array such that its value occurs at more than half of indices in the array.

A zero-indexed array A consisting of N integers is given. The *dominator* of array A is the value that occurs in more than half of the elements of A.

For example, consider array A such that

A[0] = 3 A[1] = 4 A[2] = 3 A[3] = 2 A[4] = 3 A[5] = -1 A[6] = 3 A[7] = 3

The dominator of A is 3 because it occurs in 5 out of 8 elements of A (namely in those with indices 0, 2, 4, 6 and 7) and 5 is more than a half of 8.

Write a function

int solution(int A[], int N);

that, given a zero-indexed array A consisting of N integers, returns index of any element of array A in which the dominator of A occurs. The function should return −1 if array A does not have a dominator.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

For example, given array A such that

A[0] = 3 A[1] = 4 A[2] = 3 A[3] = 2 A[4] = 3 A[5] = -1 A[6] = 3 A[7] = 3

the function may return 0, 2, 4, 6 or 7, as explained above.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 9*

## Maximum slice problem

PAINLESS

[MaxDoubleSliceSum](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_double_slice_sum/)

[**START**](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_double_slice_sum/start/)

Find the maximal sum of any double slice.

A non-empty zero-indexed array A consisting of N integers is given.

A triplet (X, Y, Z), such that 0 ≤ X < Y < Z < N, is called a *double slice*.

The *sum* of double slice (X, Y, Z) is the total of A[X + 1] + A[X + 2] + ... + A[Y − 1] + A[Y + 1] + A[Y + 2] + ... + A[Z − 1].

For example, array A such that:

A[0] = 3 A[1] = 2 A[2] = 6 A[3] = -1 A[4] = 4 A[5] = 5 A[6] = -1 A[7] = 2

contains the following example double slices:

* double slice (0, 3, 6), sum is 2 + 6 + 4 + 5 = 17,
* double slice (0, 3, 7), sum is 2 + 6 + 4 + 5 − 1 = 16,
* double slice (3, 4, 5), sum is 0.

The goal is to find the maximal sum of any double slice.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the maximal sum of any double slice.

For example, given:

A[0] = 3 A[1] = 2 A[2] = 6 A[3] = -1 A[4] = 4 A[5] = 5 A[6] = -1 A[7] = 2

the function should return 17, because no double slice of array A has a sum of greater than 17.

Assume that:

* N is an integer within the range [3..100,000];
* each element of array A is an integer within the range [−10,000..10,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[MaxProfit](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_profit/)

[**START**](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_profit/start/)

Given a log of stock prices compute the maximum possible earning.

A zero-indexed array A consisting of N integers is given. It contains daily prices of a stock share for a period of N consecutive days. If a single share was bought on day P and sold on day Q, where 0 ≤ P ≤ Q < N, then the *profit* of such transaction is equal to A[Q] − A[P], provided that A[Q] ≥ A[P]. Otherwise, the transaction brings *loss* of A[P] − A[Q].

For example, consider the following array A consisting of six elements such that:

A[0] = 23171 A[1] = 21011 A[2] = 21123 A[3] = 21366 A[4] = 21013 A[5] = 21367

If a share was bought on day 0 and sold on day 2, a loss of 2048 would occur because A[2] − A[0] = 21123 − 23171 = −2048. If a share was bought on day 4 and sold on day 5, a profit of 354 would occur because A[5] − A[4] = 21367 − 21013 = 354. Maximum possible profit was 356. It would occur if a share was bought on day 1 and sold on day 5.

Write a function,

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers containing daily prices of a stock share for a period of N consecutive days, returns the maximum possible profit from one transaction during this period. The function should return 0 if it was impossible to gain any profit.

For example, given array A consisting of six elements such that:

A[0] = 23171 A[1] = 21011 A[2] = 21123 A[3] = 21366 A[4] = 21013 A[5] = 21367

the function should return 356, as explained above.

Assume that:

* N is an integer within the range [0..400,000];
* each element of array A is an integer within the range [0..200,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

[MaxSliceSum](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_slice_sum/)

[**START**](https://codility.com/programmers/lessons/9-maximum_slice_problem/max_slice_sum/start/)

Find a maximum sum of a compact subsequence of array elements.

A non-empty zero-indexed array A consisting of N integers is given. A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a *slice* of array A. The *sum* of a slice (P, Q) is the total of A[P] + A[P+1] + ... + A[Q].

Write a function:

class Solution { public int solution(int[] A); }

that, given an array A consisting of N integers, returns the maximum sum of any slice of A.

For example, given array A such that:

A[0] = 3 A[1] = 2 A[2] = -6 A[3] = 4 A[4] = 0

the function should return 5 because:

* (3, 4) is a slice of A that has sum 4,
* (2, 2) is a slice of A that has sum −6,
* (0, 1) is a slice of A that has sum 5,
* no other slice of A has sum greater than (0, 1).

Assume that:

* N is an integer within the range [1..1,000,000];
* each element of array A is an integer within the range [−1,000,000..1,000,000];
* the result will be an integer within the range [−2,147,483,648..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 10*

## Prime and composite numbers

[MinPerimeterRectangle](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/min_perimeter_rectangle/)

[**START**](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/min_perimeter_rectangle/start/)

Find the minimal perimeter of any rectangle whose area equals N.

An integer N is given, representing the area of some rectangle.

The *area* of a rectangle whose sides are of length A and B is A \* B, and the *perimeter* is 2 \* (A + B).

The goal is to find the minimal perimeter of any rectangle whose area equals N. The sides of this rectangle should be only integers.

For example, given integer N = 30, rectangles of area 30 are:

* (1, 30), with a perimeter of 62,
* (2, 15), with a perimeter of 34,
* (3, 10), with a perimeter of 26,
* (5, 6), with a perimeter of 22.

Write a function:

class Solution { public int solution(int N); }

that, given an integer N, returns the minimal perimeter of any rectangle whose area is exactly equal to N.

For example, given an integer N = 30, the function should return 22, as explained above.

Assume that:

* N is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(sqrt(N));
* expected worst-case space complexity is O(1).

PAINLESS

[CountFactors](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/count_factors/)

* [**START**](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/count_factors/start/)
* Count factors of given number n.

A positive integer D is a *factor* of a positive integer N if there exists an integer M such that N = D \* M.

For example, 6 is a factor of 24, because M = 4 satisfies the above condition (24 = 6 \* 4).

Write a function:

class Solution { public int solution(int N); }

that, given a positive integer N, returns the number of its factors.

For example, given N = 24, the function should return 8, because 24 has 8 factors, namely 1, 2, 3, 4, 6, 8, 12, 24. There are no other factors of 24.

Assume that:

* N is an integer within the range [1..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(sqrt(N));
* expected worst-case space complexity is O(1).

[Peaks](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/peaks/)

[**START**](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/peaks/start/)

Divide an array into the maximum number of same-sized blocks, each of which should contain an index P such that A[P - 1] < A[P] > A[P + 1].

A non-empty zero-indexed array A consisting of N integers is given.

A *peak* is an array element which is larger than its neighbors. More precisely, it is an index P such that 0 < P < N − 1,  A[P − 1] < A[P] and A[P] > A[P + 1].

For example, the following array A:

A[0] = 1 A[1] = 2 A[2] = 3 A[3] = 4 A[4] = 3 A[5] = 4 A[6] = 1 A[7] = 2 A[8] = 3 A[9] = 4 A[10] = 6 A[11] = 2

has exactly three peaks: 3, 5, 10.

We want to divide this array into blocks containing the same number of elements. More precisely, we want to choose a number K that will yield the following blocks:

* A[0], A[1], ..., A[K − 1],
* A[K], A[K + 1], ..., A[2K − 1],  
  ...
* A[N − K], A[N − K + 1], ..., A[N − 1].

What's more, every block should contain at least one peak. Notice that extreme elements of the blocks (for example A[K − 1] or A[K]) can also be peaks, but only if they have both neighbors (including one in an adjacent blocks).

The goal is to find the maximum number of blocks into which the array A can be divided.

Array A can be divided into blocks as follows:

* one block (1, 2, 3, 4, 3, 4, 1, 2, 3, 4, 6, 2). This block contains three peaks.
* two blocks (1, 2, 3, 4, 3, 4) and (1, 2, 3, 4, 6, 2). Every block has a peak.
* three blocks (1, 2, 3, 4), (3, 4, 1, 2), (3, 4, 6, 2). Every block has a peak. Notice in particular that the first block (1, 2, 3, 4) has a peak at A[3], because A[2] < A[3] > A[4], even though A[4] is in the adjacent block.

However, array A cannot be divided into four blocks, (1, 2, 3), (4, 3, 4), (1, 2, 3) and (4, 6, 2), because the (1, 2, 3) blocks do not contain a peak. Notice in particular that the (4, 3, 4) block contains two peaks: A[3] and A[5].

The maximum number of blocks that array A can be divided into is three.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the maximum number of blocks into which A can be divided.

If A cannot be divided into some number of blocks, the function should return 0.

For example, given:

A[0] = 1 A[1] = 2 A[2] = 3 A[3] = 4 A[4] = 3 A[5] = 4 A[6] = 1 A[7] = 2 A[8] = 3 A[9] = 4 A[10] = 6 A[11] = 2

the function should return 3, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [0..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N\*log(log(N)));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[Flags](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/flags/)

[**START**](https://codility.com/programmers/lessons/10-prime_and_composite_numbers/flags/start/)

Find the maximum number of flags that can be set on mountain peaks.

A non-empty zero-indexed array A consisting of N integers is given.

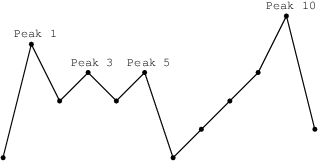
A *peak* is an array element which is larger than its neighbours. More precisely, it is an index P such that 0 < P < N − 1 and A[P − 1] < A[P] > A[P + 1].

For example, the following array A:

A[0] = 1 A[1] = 5 A[2] = 3 A[3] = 4 A[4] = 3 A[5] = 4 A[6] = 1 A[7] = 2 A[8] = 3 A[9] = 4 A[10] = 6 A[11] = 2

has exactly four peaks: elements 1, 3, 5 and 10.

You are going on a trip to a range of mountains whose relative heights are represented by array A, as shown in a figure below. You have to choose how many flags you should take with you. The goal is to set the maximum number of flags on the peaks, according to certain rules.



Flags can only be set on peaks. What's more, if you take K flags, then the distance between any two flags should be greater than or equal to K. The distance between indices P and Q is the absolute value |P − Q|.

For example, given the mountain range represented by array A, above, with N = 12, if you take:

* two flags, you can set them on peaks 1 and 5;
* three flags, you can set them on peaks 1, 5 and 10;
* four flags, you can set only three flags, on peaks 1, 5 and 10.

You can therefore set a maximum of three flags in this case.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A of N integers, returns the maximum number of flags that can be set on the peaks of the array.

For example, the following array A:

A[0] = 1 A[1] = 5 A[2] = 3 A[3] = 4 A[4] = 3 A[5] = 4 A[6] = 1 A[7] = 2 A[8] = 3 A[9] = 4 A[10] = 6 A[11] = 2

the function should return 3, as explained above.

Assume that:

* N is an integer within the range [1..400,000];
* each element of array A is an integer within the range [0..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 11*

## Sieve of Eratosthenes

PAINLESS

[CountSemiprimes](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_semiprimes/)

[**START**](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_semiprimes/start/)

Count the semiprime numbers in the given range [a..b]

A *prime* is a positive integer X that has exactly two distinct divisors: 1 and X. The first few prime integers are 2, 3, 5, 7, 11 and 13.

A *semiprime* is a natural number that is the product of two (not necessarily distinct) prime numbers. The first few semiprimes are 4, 6, 9, 10, 14, 15, 21, 22, 25, 26.

You are given two non-empty zero-indexed arrays P and Q, each consisting of M integers. These arrays represent queries about the number of semiprimes within specified ranges.

Query K requires you to find the number of semiprimes within the range (P[K], Q[K]), where 1 ≤ P[K] ≤ Q[K] ≤ N.

For example, consider an integer N = 26 and arrays P, Q such that:

P[0] = 1 Q[0] = 26 P[1] = 4 Q[1] = 10 P[2] = 16 Q[2] = 20

The number of semiprimes within each of these ranges is as follows:

* (1, 26) is 10,
* (4, 10) is 4,
* (16, 20) is 0.

Write a function:

class Solution { public int[] solution(int N, int[] P, int[] Q); }

that, given an integer N and two non-empty zero-indexed arrays P and Q consisting of M integers, returns an array consisting of M elements specifying the consecutive answers to all the queries.

For example, given an integer N = 26 and arrays P, Q such that:

P[0] = 1 Q[0] = 26 P[1] = 4 Q[1] = 10 P[2] = 16 Q[2] = 20

the function should return the values [10, 4, 0], as explained above.

Assume that:

* N is an integer within the range [1..50,000];
* M is an integer within the range [1..30,000];
* each element of arrays P, Q is an integer within the range [1..N];
* P[i] ≤ Q[i].

Complexity:

* expected worst-case time complexity is O(N\*log(log(N))+M);
* expected worst-case space complexity is O(N+M), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[CountNonDivisible](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_non_divisible/)

[**START**](https://codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_non_divisible/start/)

Calculate the number of elements of an array that are not divisors of each element.

You are given a non-empty zero-indexed array A consisting of N integers.

For each number A[i] such that 0 ≤ i < N, we want to count the number of elements of the array that are not the divisors of A[i]. We say that these elements are non-divisors.

For example, consider integer N = 5 and array A such that:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 3 A[4] = 6

For the following elements:

* A[0] = 3, the non-divisors are: 2, 6,
* A[1] = 1, the non-divisors are: 3, 2, 3, 6,
* A[2] = 2, the non-divisors are: 3, 3, 6,
* A[3] = 3, the non-divisors are: 2, 6,
* A[4] = 6, there aren't any non-divisors.

Write a function:

class Solution { public int[] solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns a sequence of integers representing the amount of non-divisors.

The sequence should be returned as:

* a structure Results (in C), or
* a vector of integers (in C++), or
* a record Results (in Pascal), or
* an array of integers (in any other programming language).

For example, given:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 3 A[4] = 6

the function should return [2, 4, 3, 2, 0], as explained above.

Assume that:

* N is an integer within the range [1..50,000];
* each element of array A is an integer within the range [1..2 \* N].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 12*

## Euclidean algorithm

PAINLESS

[ChocolatesByNumbers](https://codility.com/programmers/lessons/12-euclidean_algorithm/chocolates_by_numbers/)

[**START**](https://codility.com/programmers/lessons/12-euclidean_algorithm/chocolates_by_numbers/start/)

There are N chocolates in a circle. Count the number of chocolates you will eat.

Two positive integers N and M are given. Integer N represents the number of chocolates arranged in a circle, numbered from 0 to N − 1.

You start to eat the chocolates. After eating a chocolate you leave only a wrapper.

You begin with eating chocolate number 0. Then you omit the next M − 1 chocolates or wrappers on the circle, and eat the following one.

More precisely, if you ate chocolate number X, then you will next eat the chocolate with number (X + M) modulo N (remainder of division).

You stop eating when you encounter an empty wrapper.

For example, given integers N = 10 and M = 4. You will eat the following chocolates: 0, 4, 8, 2, 6.

The goal is to count the number of chocolates that you will eat, following the above rules.

Write a function:

class Solution { public int solution(int N, int M); }

that, given two positive integers N and M, returns the number of chocolates that you will eat.

For example, given integers N = 10 and M = 4. the function should return 5, as explained above.

Assume that:

* N and M are integers within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(log(N+M));
* expected worst-case space complexity is O(log(N+M)).

RESPECTABLE

[CommonPrimeDivisors](https://codility.com/programmers/lessons/12-euclidean_algorithm/common_prime_divisors/)

[**START**](https://codility.com/programmers/lessons/12-euclidean_algorithm/common_prime_divisors/start/)

Check whether two numbers have the same prime divisors.

A *prime* is a positive integer X that has exactly two distinct divisors: 1 and X. The first few prime integers are 2, 3, 5, 7, 11 and 13.

A prime D is called a *prime divisor* of a positive integer P if there exists a positive integer K such that D \* K = P. For example, 2 and 5 are prime divisors of 20.

You are given two positive integers N and M. The goal is to check whether the sets of prime divisors of integers N and M are exactly the same.

For example, given:

* N = 15 and M = 75, the prime divisors are the same: {3, 5};
* N = 10 and M = 30, the prime divisors aren't the same: {2, 5} is not equal to {2, 3, 5};
* N = 9 and M = 5, the prime divisors aren't the same: {3} is not equal to {5}.

Write a function:

class Solution { public int solution(int[] A, int[] B); }

that, given two non-empty zero-indexed arrays A and B of Z integers, returns the number of positions K for which the prime divisors of A[K] and B[K] are exactly the same.

For example, given:

A[0] = 15 B[0] = 75 A[1] = 10 B[1] = 30 A[2] = 3 B[2] = 5

the function should return 1, because only one pair (15, 75) has the same set of prime divisors.

Assume that:

* Z is an integer within the range [1..6,000];
* each element of arrays A, B is an integer within the range [1..2,147,483,647].

Complexity:

* expected worst-case time complexity is O(Z\*log(max(A)+max(B))2);
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 13*

## Fibonacci numbers

RESPECTABLE

[Ladder](https://codility.com/programmers/lessons/13-fibonacci_numbers/ladder/)

[**START**](https://codility.com/programmers/lessons/13-fibonacci_numbers/ladder/start/)

Count the number of different ways of climbing to the top of a ladder.

You have to climb up a ladder. The ladder has exactly N rungs, numbered from 1 to N. With each step, you can ascend by one or two rungs. More precisely:

* with your first step you can stand on rung 1 or 2,
* if you are on rung K, you can move to rungs K + 1 or K + 2,
* finally you have to stand on rung N.

Your task is to count the number of different ways of climbing to the top of the ladder.

For example, given N = 4, you have five different ways of climbing, ascending by:

* 1, 1, 1 and 1 rung,
* 1, 1 and 2 rungs,
* 1, 2 and 1 rung,
* 2, 1 and 1 rungs, and
* 2 and 2 rungs.

Given N = 5, you have eight different ways of climbing, ascending by:

* 1, 1, 1, 1 and 1 rung,
* 1, 1, 1 and 2 rungs,
* 1, 1, 2 and 1 rung,
* 1, 2, 1 and 1 rung,
* 1, 2 and 2 rungs,
* 2, 1, 1 and 1 rungs,
* 2, 1 and 2 rungs, and
* 2, 2 and 1 rung.

The number of different ways can be very large, so it is sufficient to return the result modulo 2P, for a given integer P.

Write a function:

class Solution { public int[] solution(int[] A, int[] B); }

that, given two non-empty zero-indexed arrays A and B of L integers, returns an array consisting of L integers specifying the consecutive answers; position I should contain the number of different ways of climbing the ladder with A[I] rungs modulo 2B[I].

For example, given L = 5 and:

A[0] = 4 B[0] = 3 A[1] = 4 B[1] = 2 A[2] = 5 B[2] = 4 A[3] = 5 B[3] = 3 A[4] = 1 B[4] = 1

the function should return the sequence [5, 1, 8, 0, 1], as explained above.

Assume that:

* L is an integer within the range [1..50,000];
* each element of array A is an integer within the range [1..L];
* each element of array B is an integer within the range [1..30].

Complexity:

* expected worst-case time complexity is O(L);
* expected worst-case space complexity is O(L), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[FibFrog](https://codility.com/programmers/lessons/13-fibonacci_numbers/fib_frog/)

[**START**](https://codility.com/programmers/lessons/13-fibonacci_numbers/fib_frog/start/)

Count the minimum number of jumps required for a frog to get to the other side of a river.

The Fibonacci sequence is defined using the following recursive formula:

F(0) = 0 F(1) = 1 F(M) = F(M - 1) + F(M - 2) if M >= 2

A small frog wants to get to the other side of a river. The frog is initially located at one bank of the river (position −1) and wants to get to the other bank (position N). The frog can jump over any distance F(K), where F(K) is the K-th Fibonacci number. Luckily, there are many leaves on the river, and the frog can jump between the leaves, but only in the direction of the bank at position N.

The leaves on the river are represented in a zero-indexed array A consisting of N integers. Consecutive elements of array A represent consecutive positions from 0 to N − 1 on the river. Array A contains only 0s and/or 1s:

* 0 represents a position without a leaf;
* 1 represents a position containing a leaf.

The goal is to count the minimum number of jumps in which the frog can get to the other side of the river (from position −1 to position N). The frog can jump between positions −1 and N (the banks of the river) and every position containing a leaf.

For example, consider array A such that:

A[0] = 0 A[1] = 0 A[2] = 0 A[3] = 1 A[4] = 1 A[5] = 0 A[6] = 1 A[7] = 0 A[8] = 0 A[9] = 0 A[10] = 0

The frog can make three jumps of length F(5) = 5, F(3) = 2 and F(5) = 5.

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers, returns the minimum number of jumps by which the frog can get to the other side of the river. If the frog cannot reach the other side of the river, the function should return −1.

For example, given:

A[0] = 0 A[1] = 0 A[2] = 0 A[3] = 1 A[4] = 1 A[5] = 0 A[6] = 1 A[7] = 0 A[8] = 0 A[9] = 0 A[10] = 0

the function should return 3, as explained above.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer that can have one of the following values: 0, 1.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 14*

## Binary search algorithm

RESPECTABLE

[MinMaxDivision](https://codility.com/programmers/lessons/14-binary_search_algorithm/min_max_division/)

[**START**](https://codility.com/programmers/lessons/14-binary_search_algorithm/min_max_division/start/)

Divide array A into K blocks and minimize the largest sum of any block.

You are given integers K, M and a non-empty zero-indexed array A consisting of N integers. Every element of the array is not greater than M.

You should divide this array into K blocks of consecutive elements. The size of the block is any integer between 0 and N. Every element of the array should belong to some block.

The sum of the block from X to Y equals A[X] + A[X + 1] + ... + A[Y]. The sum of empty block equals 0.

The *large sum* is the maximal sum of any block.

For example, you are given integers K = 3, M = 5 and array A such that:

A[0] = 2 A[1] = 1 A[2] = 5 A[3] = 1 A[4] = 2 A[5] = 2 A[6] = 2

The array can be divided, for example, into the following blocks:

* [2, 1, 5, 1, 2, 2, 2], [], [] with a large sum of 15;
* [2], [1, 5, 1, 2], [2, 2] with a large sum of 9;
* [2, 1, 5], [], [1, 2, 2, 2] with a large sum of 8;
* [2, 1], [5, 1], [2, 2, 2] with a large sum of 6.

The goal is to minimize the large sum. In the above example, 6 is the minimal large sum.

Write a function:

class Solution { public int solution(int K, int M, int[] A); }

that, given integers K, M and a non-empty zero-indexed array A consisting of N integers, returns the minimal large sum.

For example, given K = 3, M = 5 and array A such that:

A[0] = 2 A[1] = 1 A[2] = 5 A[3] = 1 A[4] = 2 A[5] = 2 A[6] = 2

the function should return 6, as explained above.

Assume that:

* N and K are integers within the range [1..100,000];
* M is an integer within the range [0..10,000];
* each element of array A is an integer within the range [0..M].

Complexity:

* expected worst-case time complexity is O(N\*log(N+M));
* expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[NailingPlanks](https://codility.com/programmers/lessons/14-binary_search_algorithm/nailing_planks/)

[**START**](https://codility.com/programmers/lessons/14-binary_search_algorithm/nailing_planks/start/)

Count the minimum number of nails that allow a series of planks to be nailed.

You are given two non-empty zero-indexed arrays A and B consisting of N integers. These arrays represent N planks. More precisely, A[K] is the start and B[K] the end of the K−th plank.

Next, you are given a non-empty zero-indexed array C consisting of M integers. This array represents M nails. More precisely, C[I] is the position where you can hammer in the I−th nail.

We say that a plank (A[K], B[K]) is nailed if there exists a nail C[I] such that A[K] ≤ C[I] ≤ B[K].

The goal is to find the minimum number of nails that must be used until all the planks are nailed. In other words, you should find a value J such that all planks will be nailed after using only the first J nails. More precisely, for every plank (A[K], B[K]) such that 0 ≤ K < N, there should exist a nail C[I] such that I < J and A[K] ≤ C[I] ≤ B[K].

For example, given arrays A, B such that:

A[0] = 1 B[0] = 4 A[1] = 4 B[1] = 5 A[2] = 5 B[2] = 9 A[3] = 8 B[3] = 10

four planks are represented: [1, 4], [4, 5], [5, 9] and [8, 10].

Given array C such that:

C[0] = 4 C[1] = 6 C[2] = 7 C[3] = 10 C[4] = 2

if we use the following nails:

* 0, then planks [1, 4] and [4, 5] will both be nailed.
* 0, 1, then planks [1, 4], [4, 5] and [5, 9] will be nailed.
* 0, 1, 2, then planks [1, 4], [4, 5] and [5, 9] will be nailed.
* 0, 1, 2, 3, then all the planks will be nailed.

Thus, four is the minimum number of nails that, used sequentially, allow all the planks to be nailed.

Write a function:

class Solution { public int solution(int[] A, int[] B, int[] C); }

that, given two non-empty zero-indexed arrays A and B consisting of N integers and a non-empty zero-indexed array C consisting of M integers, returns the minimum number of nails that, used sequentially, allow all the planks to be nailed.

If it is not possible to nail all the planks, the function should return −1.

For example, given arrays A, B, C such that:

A[0] = 1 B[0] = 4 A[1] = 4 B[1] = 5 A[2] = 5 B[2] = 9 A[3] = 8 B[3] = 10 C[0] = 4 C[1] = 6 C[2] = 7 C[3] = 10 C[4] = 2

the function should return 4, as explained above.

Assume that:

* N and M are integers within the range [1..30,000];
* each element of arrays A, B, C is an integer within the range [1..2\*M];
* A[K] ≤ B[K].

Complexity:

* expected worst-case time complexity is O((N+M)\*log(M));
* expected worst-case space complexity is O(M), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 15*

## Caterpillar method

PAINLESS

[AbsDistinct](https://codility.com/programmers/lessons/15-caterpillar_method/abs_distinct/)

[**START**](https://codility.com/programmers/lessons/15-caterpillar_method/abs_distinct/start/)

Compute number of distinct absolute values of sorted array elements.

A non-empty zero-indexed array A consisting of N numbers is given. The array is sorted in non-decreasing order. The *absolute distinct count* of this array is the number of distinct absolute values among the elements of the array.

For example, consider array A such that:

A[0] = -5 A[1] = -3 A[2] = -1 A[3] = 0 A[4] = 3 A[5] = 6

The absolute distinct count of this array is 5, because there are 5 distinct absolute values among the elements of this array, namely 0, 1, 3, 5 and 6.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N numbers, returns absolute distinct count of array A.

For example, given array A such that:

A[0] = -5 A[1] = -3 A[2] = -1 A[3] = 0 A[4] = 3 A[5] = 6

the function should return 5, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [−2,147,483,648..2,147,483,647];
* array A is sorted in non-decreasing order.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

PAINLESS

[CountDistinctSlices](https://codility.com/programmers/lessons/15-caterpillar_method/count_distinct_slices/)

[**START**](https://codility.com/programmers/lessons/15-caterpillar_method/count_distinct_slices/start/)

Count the number of distinct slices (containing only unique numbers).

An integer M and a non-empty zero-indexed array A consisting of N non-negative integers are given. All integers in array A are less than or equal to M.

A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a *slice* of array A. The slice consists of the elements A[P], A[P + 1], ..., A[Q]. A *distinct slice* is a slice consisting of only unique numbers. That is, no individual number occurs more than once in the slice.

For example, consider integer M = 6 and array A such that:

A[0] = 3 A[1] = 4 A[2] = 5 A[3] = 5 A[4] = 2

There are exactly nine distinct slices: (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2), (3, 3), (3, 4) and (4, 4).

The goal is to calculate the number of distinct slices.

Write a function:

class Solution { public int solution(int M, int[] A); }

that, given an integer M and a non-empty zero-indexed array A consisting of N integers, returns the number of distinct slices.

If the number of distinct slices is greater than 1,000,000,000, the function should return 1,000,000,000.

For example, given integer M = 6 and array A such that:

A[0] = 3 A[1] = 4 A[2] = 5 A[3] = 5 A[4] = 2

the function should return 9, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* M is an integer within the range [0..100,000];
* each element of array A is an integer within the range [0..M].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(M), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

PAINLESS

[CountTriangles](https://codility.com/programmers/lessons/15-caterpillar_method/count_triangles/)

[**START**](https://codility.com/programmers/lessons/15-caterpillar_method/count_triangles/start/)

Count the number of triangles that can be built from a given set of edges.

A zero-indexed array A consisting of N integers is given. A triplet (P, Q, R) is *triangular* if it is possible to build a triangle with sides of lengths A[P], A[Q] and A[R]. In other words, triplet (P, Q, R) is triangular if 0 ≤ P < Q < R < N and:

* A[P] + A[Q] > A[R],
* A[Q] + A[R] > A[P],
* A[R] + A[P] > A[Q].

For example, consider array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 12

There are four triangular triplets that can be constructed from elements of this array, namely (0, 2, 4), (0, 2, 5), (0, 4, 5), and (2, 4, 5).

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers, returns the number of triangular triplets in this array.

For example, given array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 12

the function should return 4, as explained above.

Assume that:

* N is an integer within the range [0..1,000];
* each element of array A is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N2);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[MinAbsSumOfTwo](https://codility.com/programmers/lessons/15-caterpillar_method/min_abs_sum_of_two/)

[**START**](https://codility.com/programmers/lessons/15-caterpillar_method/min_abs_sum_of_two/start/)

Find the minimal absolute value of a sum of two elements.

Let A be a non-empty zero-indexed array consisting of N integers.

The *abs sum of two* for a pair of indices (P, Q) is the absolute value |A[P] + A[Q]|, for 0 ≤ P ≤ Q < N.

For example, the following array A:

A[0] = 1 A[1] = 4 A[2] = -3

has pairs of indices (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2).   
The abs sum of two for the pair (0, 0) is A[0] + A[0] = |1 + 1| = 2.   
The abs sum of two for the pair (0, 1) is A[0] + A[1] = |1 + 4| = 5.   
The abs sum of two for the pair (0, 2) is A[0] + A[2] = |1 + (−3)| = 2.   
The abs sum of two for the pair (1, 1) is A[1] + A[1] = |4 + 4| = 8.   
The abs sum of two for the pair (1, 2) is A[1] + A[2] = |4 + (−3)| = 1.   
The abs sum of two for the pair (2, 2) is A[2] + A[2] = |(−3) + (−3)| = 6.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the minimal abs sum of two for any pair of indices in this array.

For example, given the following array A:

A[0] = 1 A[1] = 4 A[2] = -3

the function should return 1, as explained above.

Given array A:

A[0] = -8 A[1] = 4 A[2] = 5 A[3] =-10 A[4] = 3

the function should return |(−8) + 5| = 3.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [−1,000,000,000..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 16*

## Greedy algorithms

PAINLESS

[MaxNonoverlappingSegments](https://codility.com/programmers/lessons/16-greedy_algorithms/max_nonoverlapping_segments/)

[**START**](https://codility.com/programmers/lessons/16-greedy_algorithms/max_nonoverlapping_segments/start/)

Find a maximal set of non-overlapping segments.

Located on a line are N segments, numbered from 0 to N − 1, whose positions are given in zero-indexed arrays A and B. For each I (0 ≤ I < N) the position of segment I is from A[I] to B[I] (inclusive). The segments are sorted by their ends, which means that B[K] ≤ B[K + 1] for K such that 0 ≤ K < N − 1.

Two segments I and J, such that I ≠ J, are *overlapping* if they share at least one common point. In other words, A[I] ≤ A[J] ≤ B[I] or A[J] ≤ A[I] ≤ B[J].

We say that the set of segments is *non-overlapping* if it contains no two overlapping segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.

For example, consider arrays A, B such that:

A[0] = 1 B[0] = 5 A[1] = 3 B[1] = 6 A[2] = 7 B[2] = 8 A[3] = 9 B[3] = 9 A[4] = 9 B[4] = 10

The segments are shown in the figure below.

https://codility-frontend-prod.s3.amazonaws.com/media/task_static/max_nonoverlapping_segments/static/images/auto/68b279360bc48af61d9d3bdfbe1d30fe.png

The size of a non-overlapping set containing a maximal number of segments is 3. For example, possible sets are {0, 2, 3}, {0, 2, 4}, {1, 2, 3} or {1, 2, 4}. There is no non-overlapping set with four segments.

Write a function:

class Solution { public int solution(int[] A, int[] B); }

that, given two zero-indexed arrays A and B consisting of N integers, returns the size of a non-overlapping set containing a maximal number of segments.

For example, given arrays A, B shown above, the function should return 3, as explained above.

Assume that:

* N is an integer within the range [0..30,000];
* each element of arrays A, B is an integer within the range [0..1,000,000,000];
* A[I] ≤ B[I], for each I (0 ≤ I < N);
* B[K] ≤ B[K + 1], for each K (0 ≤ K < N − 1).

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

PAINLESS

[TieRopes](https://codility.com/programmers/lessons/16-greedy_algorithms/tie_ropes/)

[**START**](https://codility.com/programmers/lessons/16-greedy_algorithms/tie_ropes/start/)

Tie adjacent ropes to achieve the maximum number of ropes of length >= K.

There are N ropes numbered from 0 to N − 1, whose lengths are given in a zero-indexed array A, lying on the floor in a line. For each I (0 ≤ I < N), the length of rope I on the line is A[I].

We say that two ropes I and I + 1 are *adjacent*. Two adjacent ropes can be tied together with a knot, and the length of the tied rope is the sum of lengths of both ropes. The resulting new rope can then be tied again.

For a given integer K, the goal is to tie the ropes in such a way that the number of ropes whose length is greater than or equal to K is maximal.

For example, consider K = 4 and array A such that:

A[0] = 1 A[1] = 2 A[2] = 3 A[3] = 4 A[4] = 1 A[5] = 1 A[6] = 3

The ropes are shown in the figure below.

https://codility-frontend-prod.s3.amazonaws.com/media/task_static/tie_ropes/static/images/auto/f13a51b17fba1ea9b8ea7fd37006f767.png

We can tie:

* rope 1 with rope 2 to produce a rope of length A[1] + A[2] = 5;
* rope 4 with rope 5 with rope 6 to produce a rope of length A[4] + A[5] + A[6] = 5.

After that, there will be three ropes whose lengths are greater than or equal to K = 4. It is not possible to produce four such ropes.

Write a function:

class Solution { public int solution(int K, int[] A); }

that, given an integer K and a non-empty zero-indexed array A of N integers, returns the maximum number of ropes of length greater than or equal to K that can be created.

For example, given K = 4 and array A such that:

A[0] = 1 A[1] = 2 A[2] = 3 A[3] = 4 A[4] = 1 A[5] = 1 A[6] = 3

the function should return 3, as explained above.

Assume that:

* N is an integer within the range [1..100,000];
* K is an integer within the range [1..1,000,000,000];
* each element of array A is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 17*

## Dynamic programming

RESPECTABLE

[NumberSolitaire](https://codility.com/programmers/lessons/17-dynamic_programming/number_solitaire/)

[**START**](https://codility.com/programmers/lessons/17-dynamic_programming/number_solitaire/start/)

In a given array, find the subset of maximal sum in which the distance between consecutive elements is at most 6.

A game for one player is played on a board consisting of N consecutive squares, numbered from 0 to N − 1. There is a number written on each square. A non-empty zero-indexed array A of N integers contains the numbers written on the squares. Moreover, some squares can be marked during the game.

At the beginning of the game, there is a pebble on square number 0 and this is the only square on the board which is marked. The goal of the game is to move the pebble to square number N − 1.

During each turn we throw a six-sided die, with numbers from 1 to 6 on its faces, and consider the number K, which shows on the upper face after the die comes to rest. Then we move the pebble standing on square number I to square number I + K, providing that square number I + K exists. If square number I + K does not exist, we throw the die again until we obtain a valid move. Finally, we mark square number I + K.

After the game finishes (when the pebble is standing on square number N − 1), we calculate the result. The result of the game is the sum of the numbers written on all marked squares.

For example, given the following array:

A[0] = 1 A[1] = -2 A[2] = 0 A[3] = 9 A[4] = -1 A[5] = -2

one possible game could be as follows:

* the pebble is on square number 0, which is marked;
* we throw 3; the pebble moves from square number 0 to square number 3; we mark square number 3;
* we throw 5; the pebble does not move, since there is no square number 8 on the board;
* we throw 2; the pebble moves to square number 5; we mark this square and the game ends.

The marked squares are 0, 3 and 5, so the result of the game is 1 + 9 + (−2) = 8. This is the maximal possible result that can be achieved on this board.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A of N integers, returns the maximal result that can be achieved on the board represented by array A.

For example, given the array

A[0] = 1 A[1] = -2 A[2] = 0 A[3] = 9 A[4] = -1 A[5] = -2

the function should return 8, as explained above.

Assume that:

* N is an integer within the range [2..100,000];
* each element of array A is an integer within the range [−10,000..10,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

AMBITIOUS

[MinAbsSum](https://codility.com/programmers/lessons/17-dynamic_programming/min_abs_sum/)

[**START**](https://codility.com/programmers/lessons/17-dynamic_programming/min_abs_sum/start/)

Given array of integers, find the lowest absolute sum of elements.

For a given array A of N integers and a sequence S of N integers from the set {−1, 1}, we define val(A, S) as follows:

val(A, S) = |sum{ A[i]\*S[i] for i = 0..N−1 }|

(Assume that the sum of zero elements equals zero.)

For a given array A, we are looking for such a sequence S that minimizes val(A,S).

Write a function:

class Solution { public int solution(int[] A); }

that, given an array A of N integers, computes the minimum value of val(A,S) from all possible values of val(A,S) for all possible sequences S of N integers from the set {−1, 1}.

For example, given array:

A[0] = 1 A[1] = 5 A[2] = 2 A[3] = -2

your function should return 0, since for S = [−1, 1, −1, 1], val(A, S) = 0, which is the minimum possible value.

Assume that:

* N is an integer within the range [0..20,000];
* each element of array A is an integer within the range [−100..100].

Complexity:

* expected worst-case time complexity is O(N\*max(abs(A))2);
* expected worst-case space complexity is O(N+sum(abs(A))), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 90*

## Tasks from Indeed Prime 2015 challenge

PAINLESS

[LongestPassword](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/longest_password/)

[**START**](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/longest_password/start/)

Given a string containing words, find the longest word that satisfies specific conditions.

You would like to set a password for a bank account. However, there are three restrictions on the format of the password:

* it has to contain only alphanumerical characters (a−z, A−Z, 0−9);
* there should be an even number of letters;
* there should be an odd number of digits.

You are given a string S consisting of N characters. String S can be divided into *words* by splitting it at, and removing, the spaces. The goal is to choose the longest word that is a valid password. You can assume that if there are K spaces in string S then there are exactly K + 1 words.

For example, given "test 5 a0A pass007 ?xy1", there are five words and three of them are valid passwords: "5", "a0A" and "pass007". Thus the longest password is "pass007" and its length is 7. Note that neither "test" nor "?xy1" is a valid password, because "?" is not an alphanumerical character and "test" contains an even number of digits (zero).

Write a function:

class Solution { public int solution(string S); }

that, given a non-empty string S consisting of N characters, returns the length of the longest word from the string that is a valid password. If there is no such word, your function should return −1.

For example, given S = "test 5 a0A pass007 ?xy1", your function should return 7, as explained above.

Assume that:

* N is an integer within the range [1..200];
* string S consists only of printable ASCII characters and spaces.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

RESPECTABLE

[FloodDepth](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/flood_depth/)

[**START**](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/flood_depth/start/)

Find the maximum depth of water in mountains after a huge rainfall.

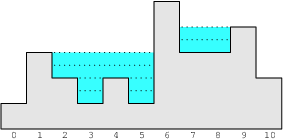
You are helping a geologist friend investigate an area with mountain lakes. A recent heavy rainfall has flooded these lakes and their water levels have reached the highest possible point. Your friend is interested to know the maximum depth in the deepest part of these lakes.

We simplify the problem in 2-D dimensions. The whole landscape can be divided into small blocks and described by an array A of length N. Each element of A is the altitude of the rock floor of a block (i.e. the height of this block when there is no water at all). After the rainfall, all the low-lying areas (i.e. blocks that have higher blocks on both sides) are holding as much water as possible. You would like to know the maximum depth of water after this entire area is flooded. You can assume that the altitude outside this area is zero and the outside area can accommodate infinite amount of water.

For example, consider array A such that:

A[0] = 1 A[1] = 3 A[2] = 2 A[3] = 1 A[4] = 2 A[5] = 1 A[6] = 5 A[7] = 3 A[8] = 3 A[9] = 4 A[10] = 2

The following picture illustrates the landscape after it has flooded:



The gray area is the rock floor described by the array A above and the blue area with dashed lines represents the water filling the low-lying areas with maximum possible volume. Thus, blocks 3 and 5 have a water depth of 2 while blocks 2, 4, 7 and 8 have a water depth of 1. Therefore, the maximum water depth of this area is 2.

Write a function:

class Solution { public int solution(int[] A); }

that, given a non-empty zero-indexed array A consisting of N integers, returns the maximum depth of water.

Given array A shown above, the function should return 2, as explained above.

For the following array:

A[0] = 5 A[1] = 8

the function should return 0, because this landscape cannot hold any water.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [1..100,000,000].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

AMBITIOUS

[SlalomSkiing](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/slalom_skiing/)

[**START**](https://codility.com/programmers/lessons/90-tasks_from_indeed_prime_2015_challenge/slalom_skiing/start/)

Given a sequence, find the longest subsequence that can be decomposed into at most three monotonic parts.

You are a skier participating in a giant slalom. The slalom track is located on a ski slope, goes downhill and is fenced by barriers on both sides. The barriers are perpendicular to the starting line located at the top of the slope. There are N slalom gates on the track. Each gate is placed at a distinct distance from the starting line and from the barrier on the right-hand side (looking downhill).

You start from any place on the starting line, ski down the track passing as many gates as possible, and finish the slalom at the bottom of the slope. *Passing* a gate means skiing through the position of the gate.

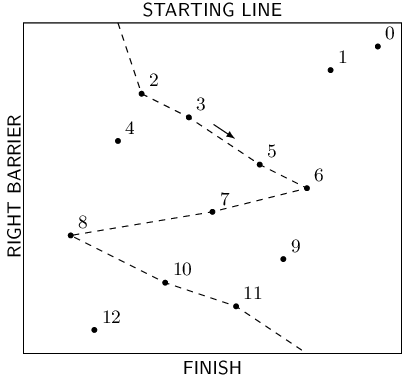
You can ski downhill in either of two directions: to the left or to the right. When you ski to the left, you pass gates of increasing distances from the right barrier, and when you ski to the right, you pass gates of decreasing distances from the right barrier. You want to ski to the left at the beginning.

Unfortunately, changing direction (left to right or vice versa) is exhausting, so you have decided to change direction *at most two* times during your ride. Because of this, you have allowed yourself to miss some of the gates on the way down the slope. You would like to know the maximum number of gates that you can pass with at most two changes of direction.

The arrangement of the gates is given as an array A consisting of N integers, whose elements specify the positions of the gates: gate K (for 0 ≤ K < N) is at a distance of K+1 from the starting line, and at a distance of A[K] from the right barrier.

For example, consider array A such that:

A[0] = 15 A[1] = 13 A[2] = 5 A[3] = 7 A[4] = 4 A[5] = 10 A[6] = 12 A[7] = 8 A[8] = 2 A[9] = 11 A[10] = 6 A[11] = 9 A[12] = 3



The picture above illustrates the example track with N = 13 gates and a course that passes eight gates. After starting, you ski to the left (from your own perspective). You pass gates 2, 3, 5, 6 and then change direction to the right. After that you pass gates 7, 8 and then change direction to the left. Finally, you pass gates 10, 11 and finish the slalom. There is no possible way of passing more gates using at most two changes of direction.

Write a function:

class Solution { public int solution(int[] A); }

that, given a zero-indexed array A consisting of N integers, describing the positions of the gates on the track, returns the maximum number of gates that you can pass during one ski run.

For example, given the above data, the function should return 8, as explained above.

For the following array A consisting of N = 2 elements:

A[0] = 1 A[1] = 5

the function should return 2.

Assume that:

* N is an integer within the range [1..100,000];
* each element of array A is an integer within the range [1..1,000,000,000];
* the elements of A are all distinct.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 91*

## Tasks from Indeed Prime 2016 challenge

RESPECTABLE

[RectangleBuilderGreaterArea](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/rectangle_builder_greater_area/)

[**START**](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/rectangle_builder_greater_area/start/)

Count the distinct rectangle sizes, of area greater than or equal to X, that can be built out of a given set of segments.

Halfling Woolly Proudhoof is an eminent sheep herder. He wants to build a pen (enclosure) for his new flock of sheep. The pen will be rectangular and built from exactly four pieces of fence (so, the pieces of fence forming the opposite sides of the pen must be of equal length). Woolly can choose these pieces out of N pieces of fence that are stored in his barn. To hold the entire flock, the area of the pen must be greater than or equal to a given threshold X.

Woolly is interested in the number of different ways in which he can build a pen. Pens are considered different if the sets of lengths of their sides are different. For example, a pen of size 1×4 is different from a pen of size 2×2 (although both have an area of 4), but pens of sizes 1×2 and 2×1 are considered the same.

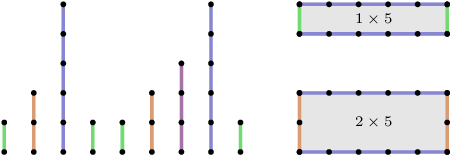
Write a function:

class Solution { public int solution(int[] A, int X); }

that, given a zero-indexed array A of N integers (containing the lengths of the available pieces of fence) and an integer X, returns the number of different ways of building a rectangular pen satisfying the above conditions. The function should return −1 if the result exceeds 1,000,000,000.

For example, given X = 5 and the following array A:

A[0] = 1 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 1 A[5] = 2 A[6] = 3 A[7] = 5 A[8] = 1



the function should return 2. The figure above shows available pieces of fence (on the left) and possible to build pens (on the right). The pens are of sizes 1x5 and 2x5. Pens of sizes 1×1 and 1×2 can be built, but are too small in area. It is not possible to build pens of sizes 2×3 or 3×5, as there is only one piece of fence of length 3.

Assume that:

* N is an integer within the range [0..100,000];
* X is an integer within the range [1..1,000,000,000];
* each element of array A is an integer within the range [1..1,000,000,000].

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

RESPECTABLE

[DwarfsRafting](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/dwarfs_rafting/)

[**START**](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/dwarfs_rafting/start/)

Find out how many dwarfs can fit on a raft such that it's balanced when crossing a river.

A company of dwarfs is travelling across the New Zealand. On reaching the Clutha River the dwarfs need to get across, but recent storms have washed away the bridge. Luckily, a small ferry, in the form of a square raft, is operating.

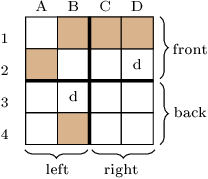
The raft is square and has N rows of seats, numbered from 1 to N. Each row contains N seats, labeled with consecutive letters (A, B, C, etc.). Each seat is identified by a string composed of its row number followed by its column number; for example, "9C" denotes the third seat in the 9th row.

The raft has already been loaded with barrels in some seat positions, and other seats are already occupied by dwarfs. Our company of dwarfs may only take the remaining unoccupied seats. The ferryman wants to accommodate as many dwarfs as possible, but the raft needs to be stable when making the crossing. That is, the following conditions must be satisfied:

* the front and back halves of the raft (in terms of the rows of seats) must each contain the same number of dwarfs;
* similarly, the left and right sides of the raft (in terms of the columns of seats) must each contain the same number of dwarfs.

You do not have to worry about balancing the barrels; you can assume that their weights are negligible.

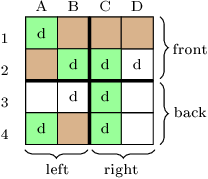
For example, a raft of size N = 4 is shown in the following illustration:



Barrels are marked as brown squares, and seats that are already occupied by dwarfs are labeled d.

The positions of the barrels are given in string S. The occupied seat numbers are given in string T. The contents of the strings are separated by single spaces and may appear in any order. For example, in the diagram above, S = "1B 1C 4B 1D 2A" and T = "3B 2D".

In this example, the ferryman can accommodate at most six more dwarfs, as indicated by the green squares in the following diagram:



The raft is then balanced: both left and right halves have the same number of dwarfs (four), and both front and back halves have the same number of dwarfs (also four).

Write a function:

class Solution { public int solution(int N, string S, string T); }

that, given the size of the raft N and two strings S, T that describes the positions of barrels and occupied seats, respectively, returns the maximum number of dwarfs that can fit on the raft. If it is not possible to balance the raft with dwarfs, your function should return -1.

For instance, given N = 4, S = "1B 1C 4B 1D 2A" and T = "3B 2D", your function should return 6, as explained above.

Assume that:

* N is an even integer within the range [2..26];
* strings S, T consist of valid seat numbers, separated with spaces;
* each seat number can appear no more than once in the strings;
* no seat number can appear in both S and T simultaneously.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

AMBITIOUS

[TreeProduct](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/tree_product/)

[**START**](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/tree_product/start/)

Remove at most two edges from a tree graph to maximize the product of the components' sizes.

Elves in the forest of Glandishar are preparing for an Orc invasion. They have a network of N + 1 guard posts located on the open platforms in the treetops. The posts are numbered from 0 to N and are connected by N bridges, so that one can get from any one guard post to any other guard post in a unique way. In other words, guard posts and bridges form a tree graph.

The Elves are afraid that if the Orcs manage to get hold of one of the guard posts, then they will have easy access to all the other guard posts. Therefore, the Elves have decided to destroy at most two bridges and split the guard posts into at most three separate areas, so that the guards can move within each area but it's not possible to move between the areas.

In each area there will be one guard who will move from guard post to guard post during the night. If the Orcs attack, the guards will raise an alarm. However, if the Orcs manage to guess the guard post in which the guards are currently located, they may manage to take out the guards without raising the alarm. The Elves want to avoid the situation that all guard posts fail this way, by maximizing:

* X \* Y \* Z, if the guard posts have been divided into three areas consisting of X, Y and Z guard posts, respectively;
* X \* Y, if the guard posts have been divided into two areas consisting of X and Y posts, respectively;
* N + 1, if the guard posts haven't been divided;

depending on which option gives the largest result.

You are given a map of the network in the form of two arrays A, B of length N. For each K (0 ≤ K < N) there is a bridge between posts A[K] and B[K].

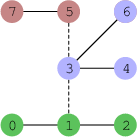
Write a function:

class Solution { public string solution(int[] A, int[] B); }

that, given two zero-indexed arrays A and B of N integers, returns the largest possible result. Since the result can be large, you should return it as a string.

For example, given the following arrays:

A[0] = 0 B[0] = 1 A[1] = 1 B[1] = 2 A[2] = 1 B[2] = 3 A[3] = 3 B[3] = 4 A[4] = 3 B[4] = 5 A[5] = 6 B[5] = 3 A[6] = 7 B[6] = 5



the function should return "18" since the Elves can destroy bridges 1−3 and 3−5 (marked as dashed lines in the image above). The created areas consist of 3, 3 and 2 guard posts.

Therefore, the result is 3 \* 3 \* 2 = 18. It is not possible to obtain a better result.

Given the following arrays:

A[0] = 0 B[0] = 1 A[1] = 1 B[1] = 2

the function should return "3" (it is optimal not to destroy any bridge).

Assume that:

* N is an integer within the range [1..50,000];
* each element of arrays A, B is an integer within the range [0..N];
* distance from guard post 0 to any other post is not greater than 900 bridges.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

AMBITIOUS

[HilbertMaze](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/hilbert_maze/)

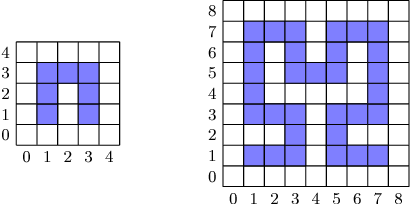
[**START**](https://codility.com/programmers/lessons/91-tasks_from_indeed_prime_2016_challenge/hilbert_maze/start/)

Find the shortest path between two fields in a Hilbert maze.

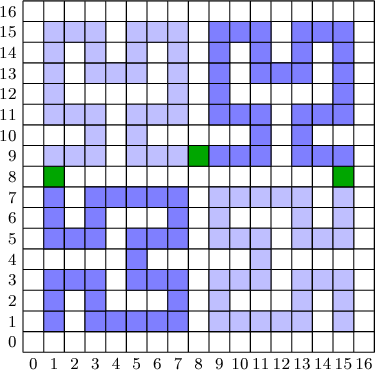
A halfling is searching for treasure hidden in a maze in the dwarfs' mine. He has a map of the maze and would like to find the shortest path to the treasure.

The maze has a specific shape. It is placed on a square grid with M2 cells, where M = 2N+1+1 for some given size N. Each cell has coordinates (x, y), where 0 ≤ x, y < M, and can either be empty or contain a rock.

The mazes of sizes N = 1 and N = 2 are presented in the pictures below:

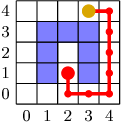


A maze of size N is constructed recursively from the layout of the maze of size N−1 (like the Hilbert curve). It contains four mazes of size N−1, each maze in one quarter. The maze in the bottom-left quarter is rotated by 90 degrees clockwise and the maze in the bottom-right quarter is rotated by 90 degrees counter-clockwise. The mazes in the top quarters are not rotated. There are three additional rocks (squares marked in green in the picture below) in the areas where the mazes intersect. The construction of the maze of size N = 3 is shown below:



The halfling would like to reach the treasure in the smallest number of steps possible. At each step, he can move to any one of the four adjacent cells (north, south, west, east) that does not contain a rock and is not outside of the grid.

For example, given N = 1, the halfling needs 8 steps to move from cell (2, 1) to cell (3, 4):



Write a function:

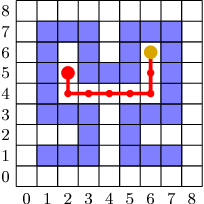
class Solution { public int solution(int N, int A, int B, int C, int D); }

that, given the size of the maze N, coordinates of the halfling (A, B) and coordinates of the treasure (C, D), returns the minimum number of steps required for the halfling to reach the treasure.

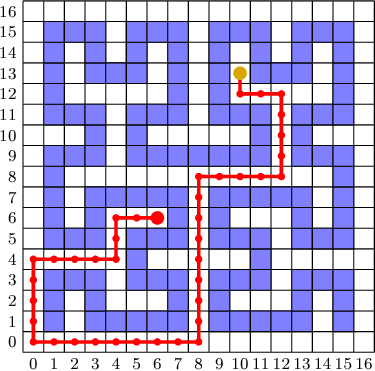
## Examples

Given N = 1, A = 2, B = 1, C = 3 and D = 4 the function should return 8, as shown above.

Given N = 2, A = 2, B = 5, C = 6 and D = 6 the function should return 7:



Given N = 3, A = 6, B = 6, C = 10 and D = 13 the function should return 39:



Assume that:

* N is an integer within the range [1..25];
* A, B, C, D are integers within the range [0..2N+1];
* cells (A, B) and (C, D) do not contain rocks;
* the result will be an integer smaller than 2,147,483,647.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N).

*Lesson 92*

## Tasks from Indeed Prime 2016 College Coders challenge

EFFORTLESS

[TennisTournament](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/tennis_tournament/)

[**START**](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/tennis_tournament/start/)

Given the numbers of players and available courts, calculate the maximum number of parallel tennis games.

You are hosting a tennis tournament. P players, who will take part in the first round of this tournament, are already registered and you have reserved C tennis courts for the matches. Exactly two players play in each game and only one game can be played on each court at any given time. You want to host the maximum possible number of games starting at the same time (in order to finish the first round quickly).

How many games can be hosted in parallel simultaneously?

Write a function:

class Solution { public int solution(int P, int C); }

that, given the number of players P and the number of reserved courts C, returns the maximum number of games that can be played in parallel.

For example, given P = 5 players and C = 3 available courts, the function should return 2, as two games can be played simultaneously (for instance, the first and second players can play on the first court, and the third and fourth players on the second court, and the third court will be empty because the fifth player doesn't have a partner to play with).

Given P = 10 players and C = 3 courts, the function should return 3, as at most three games can be hosted in parallel.

Assume that:

* P and C are integers within the range [1..30,000].

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

RESPECTABLE

[SocksLaundering](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/socks_laundering/)

[**START**](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/socks_laundering/start/)

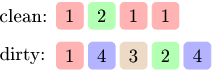
From drawers containing both clean and dirty socks, choose socks to launder in order to obtain the maximum number of clean pairs of socks.

Bob is about to go on a trip. But first he needs to take care of his supply of socks. Each sock has its own color. Bob wants to take as many pairs of clean socks as possible (both socks in the pair should be of the same color).

Socks are divided into two drawers: clean and dirty socks. Bob has time for only one laundry and his washing machine can clean at most K socks. He wants to pick socks for laundering in such a way that after washing he will have a maximal number of clean, same-colored pairs of socks. It is possible that some socks cannot be paired with any other sock, because Bob may have lost some socks over the years.

Bob has exactly N clean and M dirty socks, which are described in arrays C and D, respectively. The colors of the socks are represented as integers (equal numbers representing identical colors).

For example, given four clean socks and five dirty socks:



If Bob's washing machine can clean at most K = 2 socks, then he can take a maximum of three pairs of clean socks. He can wash one red sock and one green sock, numbered 1 and 2 respectively. Then he will have two pairs of red socks and one pair of green socks.

Write a function:

class Solution { public int solution(int K, int[] C, int[] D); }

that, given an integer K (the number of socks that the washing machine can clean), two arrays C and D (containing the color representations of N clean and M dirty socks respectively), returns the maximum number of pairs of socks that Bob can take on the trip.

For example, given K = 2, C = [1, 2, 1, 1] and D = [1, 4, 3, 2, 4], the function should return 3, as explained above.

Assume that:

* K is an integer within the range [0..50];
* each element of arrays C, D is an integer within the range [1..50];
* C and D are not empty and each of them contains at most 50 elements.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

AMBITIOUS

[ArrayRecovery](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/array_recovery/)

[**START**](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/array_recovery/start/)

Recover a broken array using partial information in another array.

Bob once had an array A with N elements. Each element was a positive integer not exceeding M.

Bob wrote a program to find an array B, defined as follows. For every index J, let's find the biggest index K such that K < J and A[K] < A[J]. Then set B[J] = A[K]. If there is no such index K, then set B[J] = 0. Intuitively, the J-th element of B contains the last value smaller than A[J] that appears before it, or 0 if there is no such element.

For example, let A = [2, 5, 3, 7, 9, 6]. Then B = [0, 2, 2, 3, 7, 3]. For instance, B[5] = 3, as A[5] is 6 and the last value before A[5] smaller than 6 is 3.

Bob computed an array B and then mistakenly deleted A. He now intends to find every valid array A from which his program would produce B. Count the number of such arrays A. Since the answer could be very big, return it modulo 109+7.

Write a function:

class Solution { public int solution(int[] B, int M); }

that, given an integer M and a zero-indexed array B with N integers, returns the remainder from the division by 109+7 of the number of valid arrays A from which Bob would get B. You can assume that there is at least one such array.

For example, given: M = 4, B = [0, 2, 2] the function should return 3. The possible removed arrays A were [2, 3, 3], [2, 4, 3] and [2, 4, 4].

For the following data: M = 10, B = [0, 3, 5, 6] the function should return 4, as the possible arrays A were [3, 5, 6, 7], [3, 5, 6, 8], [3, 5, 6, 9] and [3, 5, 6, 10].

For the following data: M = 105, B = [0, 0] there are 5000050000 possible arrays (the first element in array A can be anything in the range 1..105 and the second element can be either equal or smaller), so the function should return 49965 (as we are taking modulo 109+7).

Assume that:

* N is an integer within the range [1..100,000];
* M is an integer within the range [1..1,000,000,000];
* each element of array B is an integer within the range [0..M];
* there exists at least one valid array A from which Bob would get array B.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

AMBITIOUS

[DiamondsCount](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/diamonds_count/)

[**START**](https://codility.com/programmers/lessons/92-tasks_from_indeed_prime_2016_college_coders_challenge/diamonds_count/start/)

Given points on a plane, count the number of sets of four points that form regular diamonds.

A *diamond* is a quadrilateral whose four sides all have the same length and whose diagonals are parallel to the coordinate axes.

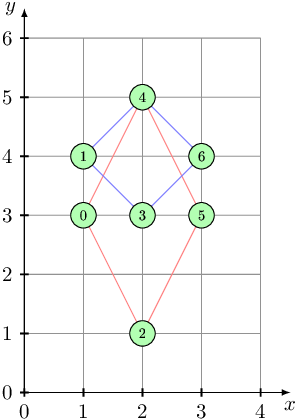
You are given N distinct points on a plane. Count the number of different diamonds that can be constructed using these points as vertices (two diamonds are different if their sets of vertices are different). Do not count diamonds whose area is empty.

Write a function:

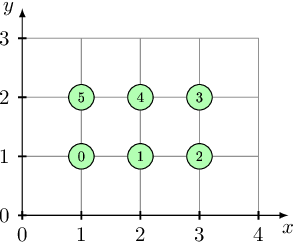
class Solution { public int solution(int[] X, int[] Y); }

that, given two zero-indexed arrays X and Y, each containing N integers, representing N points (where X[K], Y[K] are the coordinates of the K-th point), returns the number of diamonds on the plane.

For example, for N = 7 points whose coordinates are specified in arrays X = [1, 1, 2, 2, 2, 3, 3] and Y = [3, 4, 1, 3, 5, 3, 4], the function should return 2, since we can find two diamonds as shown in the picture below:



Given arrays: X = [1, 2, 3, 3, 2, 1], Y = [1, 1, 1, 2, 2, 2], the function should return 0, since there are no diamonds on the plane:



Assume that:

* N is an integer within the range [4..1,500];
* each element of arrays X, Y is an integer within the range [0..N−1];
* given N points are pairwise distinct.

Complexity:

* expected worst-case time complexity is O(N2);
* expected worst-case space complexity is O(N2), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

*Lesson 99*

## Future training

EFFORTLESS

[SqlSum](https://codility.com/programmers/lessons/99-future_training/sql_sum/)

[**START**](https://codility.com/programmers/lessons/99-future_training/sql_sum/start/)

Calculate sum of elements.

Given a table elements with the following structure:

create table elements ( v integer not null );

write an SQL query that returns the sum of the numbers in column v.

For example, given:

v --- 2 10 20 10

your query should return 42.

PAINLESS

[StrSymmetryPoint](https://codility.com/programmers/lessons/99-future_training/str_symmetry_point/)

[**START**](https://codility.com/programmers/lessons/99-future_training/str_symmetry_point/start/)

Find a symmetry point of a string, if any.

Write a function:

class Solution { public int solution(string S); }

that, given a string S, returns the index (counting from 0) of a character such that the part of the string to the left of that character is a reversal of the part of the string to its right. The function should return −1 if no such index exists.

*Note:* reversing an empty string (i.e. a string whose length is zero) gives an empty string.

For example, given a string:

"racecar"

the function should return 3, because the substring to the left of the character "e" at index 3 is "rac", and the one to the right is "car".

Given a string:

"x"

the function should return 0, because both substrings are empty.

Assume that:

* the length of S is within the range [0..2,000,000].

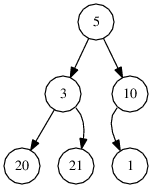
Complexity:

* expected worst-case time complexity is O(length(S));
* expected worst-case space complexity is O(1) (not counting the storage required for input arguments).
* PAINLESS
* [TreeHeight](https://codility.com/programmers/lessons/99-future_training/tree_height/)
* [**START**](https://codility.com/programmers/lessons/99-future_training/tree_height/start/)
* Compute the height of a binary tree.

In this problem we consider binary trees, represented by pointer data structures.

A *binary tree* is either an empty tree or a node (called the *root*) consisting of a single integer value and two further binary trees, called the *left subtree* and the *right subtree*.

For example, the figure below shows a binary tree consisting of six nodes. Its root contains the value 5, and the roots of its left and right subtrees have the values 3 and 10, respectively. The right subtree of the node containing the value 10, as well as the left and right subtrees of the nodes containing the values 20, 21 and 1, are empty trees.



A *path* in a binary tree is a non-empty sequence of nodes that one can traverse by following the pointers. The *length* of a path is the number of pointers it traverses. More formally, a path of length K is a sequence of nodes P[0], P[1], ..., P[K], such that node P[I + 1] is the root of the left or right subtree of P[I], for 0 ≤ I < K. For example, the sequence of nodes with values 5, 3, 21 is a path of length 2 in the tree from the above figure. The sequence of nodes with values 10, 1 is a path of length 1. The sequence of nodes with values 21, 3, 20 is not a valid path.

The *height* of a binary tree is defined as the length of the longest possible path in the tree. In particular, a tree consisting of only one node has height 0 and, conventionally, an empty tree has height −1. For example, the tree shown in the above figure is of height 2.

## Problem

Write a function:

class Solution { public int solution(Tree T); }

that, given a non-empty binary tree T consisting of N nodes, returns its height. For example, given tree T shown in the figure above, the function should return 2, as explained above. Note that the values contained in the nodes are not relevant in this task.

## Technical details

A binary tree can be given using a pointer data structure. Assume that the following declarations are given:

class Tree { public int x; public Tree l; public Tree r; };

An empty tree is represented by an empty pointer (denoted by null). A non-empty tree is represented by a pointer to an object representing its root. The attribute x holds the integer contained in the root, whereas attributes l and r hold the left and right subtrees of the binary tree, respectively.

For the purpose of entering your own test cases, you can denote a tree recursively in the following way. An empty binary tree is denoted by None. A non-empty tree is denoted as (X, L, R), where X is the value contained in the root and L and R denote the left and right subtrees, respectively. The tree from the above figure can be denoted as:

(5, (3, (20, None, None), (21, None, None)), (10, (1, None, None), None))

## Assumptions

Assume that:

* N is an integer within the range [1..1,000];
* the height of tree T (number of edges on the longest path from root to leaf) is within the range [0..500].

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N).
* RESPECTABLE
* [ArrayInversionCount](https://codility.com/programmers/lessons/99-future_training/array_inversion_count/)
* [**START**](https://codility.com/programmers/lessons/99-future_training/array_inversion_count/start/)
* Compute number of inversion in an array.

A zero-indexed array A consisting of N integers is given. An *inversion* is a pair of indexes (P, Q) such that P < Q and A[Q] < A[P].

Write a function:

class Solution { public int solution(int[] A); }

that computes the number of inversions in A, or returns −1 if it exceeds 1,000,000,000.

Assume that:

* N is an integer within the range [0..100,000];
* each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

For example, in the following array:

A[0] = -1 A[1] = 6 A[2] = 3 A[3] = 4 A[4] = 7 A[5] = 4

there are four inversions:

(1,2) (1,3) (1,5) (4,5)

so the function should return 4.

Complexity:

* expected worst-case time complexity is O(N\*log(N));
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

AMBITIOUS

[PolygonConcavityIndex](https://codility.com/programmers/lessons/99-future_training/polygon_concavity_index/)

[**START**](https://codility.com/programmers/lessons/99-future_training/polygon_concavity_index/start/)

Check whether a given polygon in a 2D plane is convex; if not, return the index of a vertex that doesn't belong to the convex hull.

An array A of points in a 2D plane is given. These points represent a polygon: every two consecutive points describe an edge of the polygon, and there is an edge connecting the last point and the first point in the array.

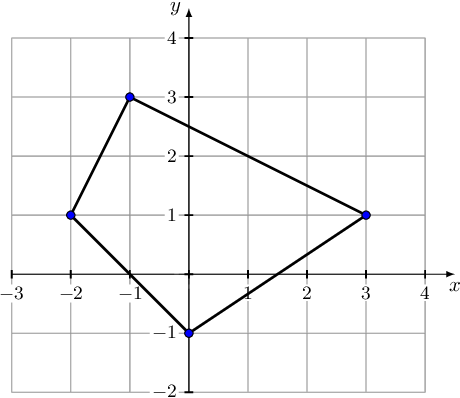
A set of points in a 2D plane, whose boundary is a straight line, is called a *semiplane*. More precisely, any set of the form {(x, y) : ax + by ≥ c} is a semiplane. The semiplane contains its boundary.

A polygon is *convex* if and only if, no line segment between two points on the boundary ever goes outside the polygon.

For example, the polygon consisting of vertices whose Cartesian coordinates are consecutively:

(-1, 3) (3, 1) (0, -1) (-2, 1)

is convex.

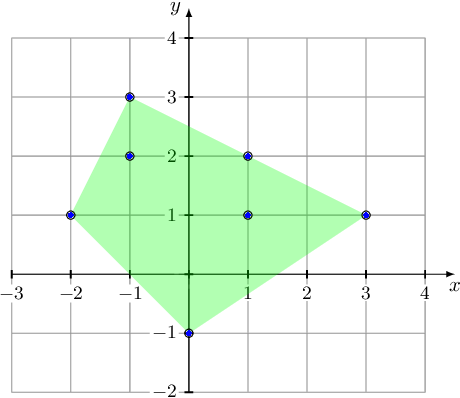


The *convex hull* of a finite set of points in a 2D plane is the smallest convex polygon that contains all points in this set. For example, the convex hull of a set consisting of seven points whose Cartesian coordinates are:

(-1, 3) (1, 2) (3, 1) (1, 1) (0, -1) (-2, 1) (-1, 2)

is a polygon that has five vertices. When traversed clockwise, its vertices are:

(-1, 3) (1, 2) (3, 1) (0, -1) (-2, 1)



If a polygon is concave (that is, it is not convex), it has a vertex which does not lie on its convex hull border. Your assignment is to find such a vertex.

Assume that the following declarations are given:

class Point2D { public int x; public int y; };

Write a function:

class Solution { public int solution(Point2D[] A); }

that, given a non-empty zero-indexed array A consisting of N elements describing a polygon, returns −1 if the polygon is convex. Otherwise, the function should return the index of any point that doesn't belong to the convex hull border. Note that consecutive edges of the polygon may be collinear (that is, the polygon might have 180−degrees angles).

To access the coordinates of the K-th point (where 0 ≤ K < N), use the following syntax:

* A[K].x to access the x-coordinate,
* A[K].y to access the y-coordinate.

For example, given array A such that:

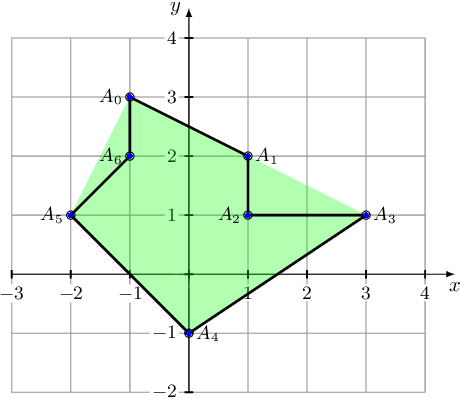
A[0].x = -1 A[0].y = 3 A[1].x = 1 A[1].y = 2 A[2].x = 3 A[2].y = 1 A[3].x = 0 A[3].y = -1 A[4].x = -2 A[4].y = 1

the function should return −1, as explained in the example above.

However, given array A such that:

A[0].x = -1 A[0].y = 3 A[1].x = 1 A[1].y = 2 A[2].x = 1 A[2].y = 1 A[3].x = 3 A[3].y = 1 A[4].x = 0 A[4].y = -1 A[5].x = -2 A[5].y = 1 A[6].x = -1 A[6].y = 2

the function should return either 2 or 6. These are the indices of the polygon lying strictly in its convex hull (that is, not on the convex hull border).



Assume that:

* N is an integer within the range [3..10,000];
* the coordinates of each point in array A are integers within the range [−1,000,000,000..1,000,000,000];
* no two edges of the polygon A intersect, other than meeting at their endpoints;
* array A does not contain duplicate points.

Complexity:

* expected worst-case time complexity is O(N);
* expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.