Homework #1

Due Time: 2022/03/25 00:00

Contact TAs: algota@noj.tw

Introduction and Rules

1. The judge system is located at https://noj.tw, please submit your code by the deadline.
2. 0215 Slides. (About Homework)
3. Can I refer to resources from the Internet or other sources that are not from textbooks or lecture slides?
   - Yes, but you must include a reference source, such as a website or book title and page number, and attach it as a comment at the top of the code.
   - Although you can refer to external resources, please write your own code after the reference.
   - Remember to specify the references; otherwise we'll view it as cheating.
4. The Non-Programming part is no need to hand in. Just for practicing!

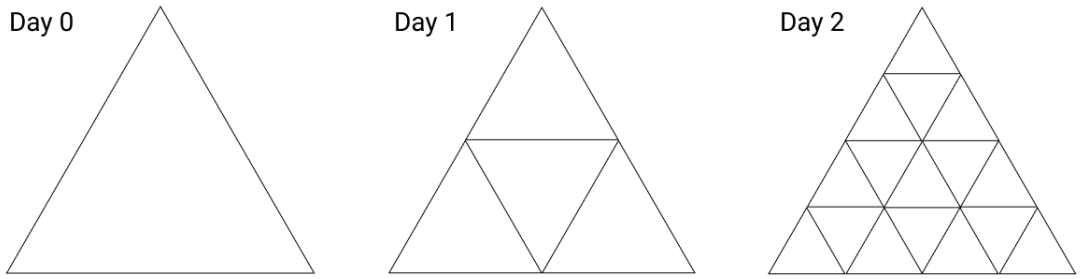
Problem A: Triangle

Time Limit per test: 1 second
Memory Limit per test: 134218 KB

Description

On day 0, there is a single triangle pointing upward, and on the next day, day 1, it will be divided into four triangles, three of which point upwards and the other downwards.

After another day, each triangle will be divided into four triangles, three of which point in the original direction and the other in the opposite direction, and so on every day.



Find out how many triangles that point upwards will be on day d.

Input

Input has only one line, which contains an integer d.

For all test data, it is guaranteed:

- $0 \leq d \leq 10^{18}$

Subtask 1 (25%)

- $0 \leq d \leq 10^5$

Subtask 2 (75%)

- No other restrictions.

Output

Print a single integer —the number of triangles that will point upwards on day d. Since the number may be huge, you only need to answer the number module $1000000007\,(10^9+7)$.

Sample Input 1

```
0
```

Sample Output 1

```
1
```

Sample Input 2

```
1
```

Sample Output 2

```
3
```

Sample Input 3

```
48763
```

Sample Output 3

```
734478132
```

Hint

If you have no idea, see Non-Programming part Problem 2-(2).

About 10^9 iterations will take around 1 second on platforms. If you get **TLE** (Time Limit Exceed), it may mean that the time complexity of your solution is not good enough to solve this problem.

Since the answer may be huge (over 2^{64}), please use Mod to prevent integer overflows. Some distributive properties of modulo:

$$(a+b)\%c = ((a\%c)+(b\%c))\%c$$

$$(a\times b)\%c = ((a\%c)\times(b\%c))\%c$$

$$(a-b)\%c = ((a\%c)-(b\%c))\%c$$

Modulo is not distributive over $\div$.

Problem B: Football Players

Time Limit per test: 1 second
Memory Limit per test: 268436 KB

Description

Strength and speed are crucial capacities in football games. Each football player P_i has two attributes st_i and sp_i, representing strength and speed in comparison with the average of players in the league, respectively.

We said that player P_a dominates player P_b if $st_a > st_b$ and $sp_a > sp_b$ at the same time.

Give you a league of football players, please find out how many players are not being dominated by any other player.

Input

The first line contains a single integer n —the number of players.

In the next n lines, each line contains two integers st_i and sp_i, representing strength and speed of of player P_i ($1 \leq i \leq n$).

For all test data, it is guaranteed:

- $1 \leq n \leq 10^5$
- $-10^9 \leq st_i, sp_i \leq 10^9$

Subtask 1 (10%)

- $1 \leq n \leq 5$
- $-10 \leq st_i, sp_i \leq 10$

Subtask 2 (20%)

- $1 \leq n \leq 10^3$
- $-10^5 \leq st_i, sp_i \leq 10^5$

Subtask 3 (70%)

- No other restrictions.

Output

Print an integer, indicating the number of players that are not being dominated by anyone.

Sample Input 1

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

Sample Output 1

```
3
```

Sample Input 2

```
8
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
```

Sample Output 2

```
8
```

Because the input files are large, please add

```
std::ios_base::sync_with_stdio(false);
std::cin.tie(nullptr);
```

to the beginning of the main function if you are using `std::cin`.

Problem C: Selection sort

Time Limit per test: 1 second
Memory Limit per test: 134218 KB

Description

Alice recently learned about selection sort and wanted to apply it to a robot to sort boxes of different weights in ascending order.

According to the rules of selection sort, in the beginning, the robot will position one of the two robot arms on the leftmost box, and the other arm will go to the right to find the box with the lightest weight and swap the two boxes. Then, the second box on the left is swapped with the box with the least weight in the unsorted part, and so on.

It will cost $(w_i + w_j) \times abs(i - j)$ units of electricity when swapping the box at index i with weight w_i and the box at index j with weight w_j. Note that the power cost of moving the robot arm is so small that it is negligible.

Please help Alice to calculate the total amount of electricity needed to sort the given sequence of boxes.

Input

First line contains an integer n, indicating the number of boxes.

Second line contains n space-separated integers w_1, w_2, ..., w_n, indicating the weights of boxes.

For all test data, it is guaranteed:

- $1 \leq n \leq 2 \times 10^5$
- $1 \leq w_i \leq 2^{31} - 1$
- all the weights are distinct.

Subtask 1 (10%)

- $1 \leq n \leq 10^2$
- $1 \leq w_i \leq 10^6$

Subtask 2 (20%)

- $1 \leq n \leq 10^4$

Subtask 3 (70%)

- No other restrictions.

Output

Print an integer, indicating the number of units of electricity needed to sort the boxes in ascending order.

Since the number may be huge, please answer the number module 1000000007 $(10^9 + 7)$.

Sample Input 1

```
3
3 2 1
```

Sample Output 1

```
8
```

Sample Input 2

```
5
7 14 17 25 30
```

Sample Output 2

```
0
```

Sample Input 3

```
10
513861 353325 293337 603099 144308 581385 669893 580172 635162 305747
```

Sample Output 3

```
19751762
```

Hint

Because the input files are large, please add

```
std::ios_base::sync_with_stdio(false);
std::cin.tie(nullptr);
```

to the beginning of the main function if you are using `std::cin`.

Problem D: oo-peh sort

Time Limit per test: 1 second
Memory Limit per test: 268436 KB

Description

After learning the sorting algorithm, Alice creates a new sorting algorithm on her own, called oo-peh sort.

Here's the pseudo code of oo-peh sort.

```
// A is a 0-indexed array of integers
oo_peh_sort(A[0..n-1])
    Sorted = false
    Counter = 0
    while Sorted == false
        Sorted = true
        for i=0; i<len(Arr)-2; i=i+1
            if A[i] > A[i+2]
                Sorted = false
                Reverse the subarray A[i..i+2]
                Counter = Counter + 1
```

The following is the demonstration with $A = [3, 4, 5, 2, 1]$.
First time in while loop:

- check if $A[0] > A[2]$, $3 > 5$, nothing happened
- check if $A[1] > A[3]$, $4 > 2$, $A = [3, 2, 5, 4, 1]$, $Counter = 1$.
- check if $A[2] > A[4]$, $5 > 1$, $A = [3, 2, 1, 4, 5]$, $Counter = 2$.

Second time in while loop:

- check if $A[0] > A[2]$, $3 > 1$, $A = [1, 2, 3, 4, 5]$, $Counter = 3$.
- check if $A[1] > A[3]$, $2 > 4$, nothing happened
- check if $A[2] > A[4]$, $3 > 5$, nothing happened

Third time in while loop: for all $A[i] > A[i + 2]$ is false, and program is over. In the case, A is successfully sorted with $Counter = 3$.

Unfortunately, Alice found that this method may not sort the sequence correctly, for example, $A = [2, 3, 1]$.

To make the algorithm perfect, she wants to do a research about what situation will oo-peh sort fail. Therefore, she needs your help. Given a sequence of n numbers, please answer her whether or not this sequence can be sorted in ascending order by oo-peh sort.

She wants to analyze the performance as well, so please also tell her the value of *Counter* after sorting with oo-peh sort.

Input

First line contains an integer n, indicating the length of the sequence given by Alice.

Second line contains n space-separated integers $a_1,\ a_2,\ ...,\ a_n$, indicating the numbers in the sequence.

For all test data, it is guaranteed:

- $3 \leq n \leq 2 \times 10^5$
- $0 \leq a_i \leq 10^9,\ \text{for } 1 \leq i \leq n$
- all the numbers in the given sequence are distinct

Subtask 1 (30%)

- $3 \leq N \leq 3 \times 10^3$

Subtask 2 (70%)

- No other restrictions.

Output

On the first line, print "yes" (without quote), if the given sequence can be sorted in ascending order by oo-peh sort, and "no" otherwise.

On the second line, print an integer, indicating the value of *Counter*.

Sample Input 1

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

Sample Output 1

```
yes
3
```

Sample Input 2

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

Sample Output 2

```
no
6
```

Hint

Because the input files are large, please add

```
std::ios_base::sync_with_stdio(false);
std::cin.tie(nullptr);
```

to the beginning of the main function if you are using `std::cin`.

Non-Programming Part

1. Complexity

For each function of the following C program, find out the tightest bound using Θ notation.

(1).

```
void f(int n) {
    int cnt = 0;
    for ( int i=1; i<=n; i++ ) {
        for ( int j=i+1; j<=n; j++ ) {
            cnt = cnt + 1;
        }
    }
}
```

(2).

```
void h(int n, int m) {
    int sum = 0;
    for ( int i=0; i<n; i++ ) {
        sum = sum + i;
    }
    for ( int i=0; i<m; i++ ) {
        sum = sum + i;
    }
}
```

(3).

```
void g(int n) {
    if ( n <= 0 ) {
        return;
    }
    g(n-2);
    g(n-1);
}
```

(4).

```
void j(int n) {
    if ( n == 1 ) {
        return;
    }
    j(n/2);
    j(n/2);
    int total = 0;
    for ( int i=0; i<n; i++ ) {
        total = total + i;
    }
}
```

(5).

```
void f(int n) {
    int cnt = 0;
    for ( int i=1; i<=n; i++ ) {
        for ( int j=i; j<=n; j+=i ) {
            cnt = cnt + 1;
        }
    }
}
```

Hint: 'Squeeze theorem'

For each recurrence relation below, find out the tightest bound using Θ notation. Show your work.

(6). $T(n) = 9T(n/3) + n$

(7). $T(n) = 2T(n/2) + n$

(8). $T(n) = 2T(n/2) + n \lg n$

(9). $T(n) = 2T(n/2) + n\sqrt{n}$

2. Fibonacci

The Fibonacci numbers form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1.

That is, $F_0 = 0,\ F_1 = 1$, and $F_N = F_{N-1} + F_{N-2},\ for\ N > 1$

The beginning of the sequence is thus: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ..., and it can be described in Matrix form:

$$\begin{pmatrix} F_{k+2} \\ F_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix}$$

Thus, we can figure out F_N with $F_0,\ F_1$ by:

$$\begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{N-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

(1).

To multiple two n-dimension matrix, we can implement by a Naïve algorithm.

```
for ( int i=0; i<n; i++ ) {
    for ( int j=0; j<n; j++ ) {
        c[i][j] = 0;
        for ( int k=0; k<n; k++ ) {
            c[i][j] = c[i][j] + a[i][k]*b[k][j];
        }
    }
}
```

What is the complexity of this algorithm? Using Θ notation with n.

(2).

Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

To obtain A^{N-1} is quite simple:

```
Power(A):
    ans = I; // Identity Matrix
    for ( int i=0; i<N-1; i++ ) {
        ans *= A; // Matrix Multiplication
    }
    return ans;
```

The complexity is $O(N)$, but we've learned that Divide & Conquer can calculate x^n more efficient.

Please provide a Divide & Conquer algorithm or method, which can obtain A^{N-1} in $o(N)$, and analyze the complexity of your solution by Master Theorem.

You can provide C code, Pseudo code or just explain the method in plain text. However, make sure your method is clearly described.

3. Quick Sort Attack

The following sorting algorithms will make the input sequence ascending if it doesn't mention.

The pseudo code of quick sort algorithm:

```
Quick Sort(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q - 1)
        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
    x = A[r]
    i = p - 1
    for j = p to r - 1
        if A[j]   x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i + 1
```

(1).

Find out the worst, the best, and the average time complexity (using Θ notation) of the above quick sort algorithm, and prove or explain why your answer is correct.

(2).

Please complete following C-code function to generate a sequence to make the above algorithm always sort it with the worst time complexity, and briefly explain why your code can do this.

```c
int *attack(int n){
    int *arr = malloc(n * sizeof(int));
    // your code here
    return arr;
}
```

4. Bubble Sort

Given a sequence $A = [a_1, a_2, a_3..., a_n]$, we could say (i, j) an inversion if $1 \leq i \leq j \leq n$ and $a_i > a_j$. Let $I(A)$ mean the number of inversions.

```
BubbleSort(A)
    for i = 1 to A.length - 1
        for j = A.length downto i + 1
            if A[j] < A[j - 1]
                exchange A[j] with A[j-1]
```

Above is the pseudo code of bubble sort.

(1).

(a)

Suppose a_i is the $k - th$ small element, which index is it at after the whole array is sorted by bubble sort? (assuming index started from 1)

(b)

Let x be the number of elements which are larger than a_i and are on his left-hand side in the initial array (before sorting), and let y be the number of elements which are smaller than a_i and are on his right-hand side. How many swapping does a_i encounter during the bubble sort procedure?

(c)

Following the previous question. How many inversions are formed by a_i ? That is, the number of inversions containing a_i.

(2).

Please try to explain or prove why the number of exchanges equal to $I(A)$, the pairs of inversion.

(a)

Please prove why after one swapping, the $I(A)$ decrease exactly 1.

(b)

Using the conclusion from (a), please prove or explain why $I(A)$ equals to the numbers of bubble sort swap.

Hint

$A = [1, 5, 2, 4, 3]$ $I(A) = 4$
because (5, 2), (5, 3), (5, 4), (4,3) are all inversions.