problem	grupe	šifre	mapa

source file	`grupe.pas`
	`grupe.c`
	`grupe.cpp`
		`sifre.pas`
		`sifre.c`
		`sifre.cpp`
			`mapa.pas`
			`mapa.c`
			`mapa.cpp`

input data	stdin	library	stdin
output data	stdout	library	stdout
memory limit (heap)	32 MB		
memory limit (stack)	1 MB		
time limit (pentium4 1.6 ghz)	4 seconds	5 seconds	6 seconds
points	**40**	**50**	**80**
	170		

grupe

First N positive integers (numbers from 1 to N) are written on the blackboard in some arbitrary order (from left to right).

A group is set of integers which form an interval. For example, sets [2], [4 5] and [3 5 6 4] are groups, but [5 7 2] and [2 4 5] aren't. At the beginning, we assume that each number on the blackboard forms a single group with only itself in it.

There is **only one allowed operation** - **concatenating two adjacent groups**, but **only if the new set would be a group**.

Write a program which **will determine wheather** sequence of N-1 operations **exists**, after which all written numbers will be in the same group. If such sequence exists, your program must find **at least one** of them.

Example (one possible solution for third example):

```
[6] [3] [2] [1] [4] [5]
[6] [3] [2 1] [4] [5]
[6] [3 2 1] [4] [5]
[6] [3 2 1] [4 5]
[6] [3 2 1 4 5]
[6 3 2 1 4 5]
```

input data

In the first line, there will be integer N, $1 \leq N \leq 500{,}000$.

In the second line, there will be N positive integers, separated by single space. These numbers are written on the blackboard (i.e. this is an initial state).

output data

In the first line there must be word `'DA'` (yes) or `'NE'` (no), depending wheather requested sequence exists. If given answer is `'NE'`, then first line must be the last.

If the written word is `'DA'`, then the next N–1 lines should consist of two integers *a* and *b* so that those numbers in the line (i+1) are the smallest and the biggest number in the group made in i-th gathering.

examples

input

```
2
2 1
```

output

```
DA
1 2
```

input

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

output

```
NE
```

input

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

output

```
DA
1 2
1 3
4 5
1 5
1 6
```

Irene has recently celebrated her 18th birthday with a big party and she has put the photos on the Internet so that all her friends can see them. But she doesn't want her rival Rachel to see the pictures, so she protected them by giving all her friends a password to open the photos.

But Rachel would like to see the pictures and luckily she knows that Irene's program works in this way:

Every password consists of **lowercase letters of the English alphabet** and all the passwords are written in a file. When somebody tries to access the photos he writes a certain sequence of characters as a password and the program **compares** the given sequence with **every** password from the file.

Program will compare two sequences A and B in K+1 miliseconds, if K is the **biggest number** so that **first K letters of a sequence A match the first K letters of a sequence B**. For example, the program will take 1 milisecond to compare sequences 'mirko' and 'slavko', 3 miliseconds for 'ma' and 'mama', 5 miliseconds for 'tata' and 'tata' and for 'mirko' and 'mirjana' program will take 4 miliseconds.

Write a program which will find at least one correct password by calling the function **`Probaj`** no more than **1700** times.

The file consists of **at most 100** passwords, which are all **different from each other**. Length of each password will be at most 100 characters.

library

You are given a library `sifrelib` which includes three functions:

`Init` - you should call this function **exactly once at the begging** of your program, without arguments. This function doesn't return value.

procedure Init;

void Init(void);

`Probaj` - this function is to be called with one argument, sequence of characters. Function returns the total time (in miliseconds) neccessary for a given sequnce to compare with **all the passwords** in a file. Sequence **isn't allowed to be empty** and has **to consist of only small letters of the English alphabet**. If you don't obey these rules you will lose points.

function Probaj(a : string) : longint;

int Probaj(const char *a);

`Rjesenje` - you should call this function **at the end** of your program; as an argument you should pass any of the passwords from the file. This function doesn't return any value and will **regulary end** your program.

procedure Rjesenje(a : string);

void Rjesenje(const char *a);

mapa

One of the biggest problems in modern cartography is writing geographic names on the map. If you write on a map the name of each city, town and a village, then some of the names will overlap and the map will become unreadable.

You are given a coordinates of all the points in a Cartesian plane, which are to be marked. The area in which you should write the name is the **rectangle** which has sides **paralel to the axes**, and the width is **exactly three times** the height.

For each point there has to be exactly one rectangle, so that the point which it marks is in his **top left corner** (i.e. the area for the name has to be bottom-right from the point). Areas for all names must have **same dimensions**.

Write a program which will find the **maximum size of the area** for the name, after every point is marked and **no two areas for the names overlap** (they can touch by an edge or a vertex).

All the coordinates of the given points will be positive integers less than or equal to 1,000,000 (one million). It is allowed for parts of rectangles to leave that area (i.e. some parts of rectangles might exceed the given limit).

input data

In the first line, there will be an integer N (number of points on the map), $2 \leq N \leq 100{,}000$.

In the next N lines there will be two integers X and Y, $0 \leq X, Y \leq 1{,}000{,}000$, coordinates of points. No two points will have the same coordinates.

output data

In the first and only line of output you should write **the height of rectangle**, real decimal number **rounded to two decimal places**.

Your output value must be within **0.01** absolute error of the correct value.

examples

input

```
3
0 0
5 4
3 7
```

output

```
3.00
```

input

```
5
1 1
6 5
18 3
9 9
16 15
```

output

```
4.00
```

input

```
10
26 77
12 37
14 18
19 96
71 95
91 9
98 43
66 77
2 75
94 91
```

output

```
7.67
```