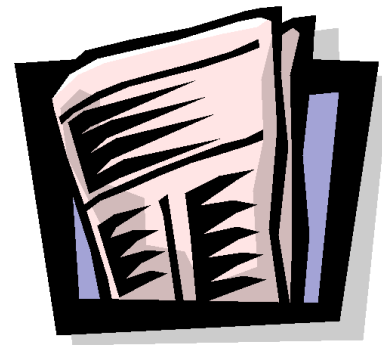
Consider the problem of laying out text in lines of a fixed maximum width L (a.k.a., "line filling").



If you do a poor job,
the ends of the lines are unnecessarily
ragged - like
this paragraph. Now,
by convention, we allow the last line of a paragraph to be arbitrarily ragged. We don't mind if that final line contains just a few characters, but we expect the earlier lines to be of approximately uniform length, filling up the column in which we are setting the text.

The straightforward approach of filling each line with as many words as will fit and then moving to the next line does not always yield the most aesthetically pleasing results. For example, the sequence

```
See if we care.
```

could be laid out in $L = 6$ like this:

```
See if
we
care.
```

That layout is, arguably, not as visually pleasing as

```
See
if we
care.
```

Define a "*word*" as any sequence of non-whitespace characters bounded by a line start or end or by a blank. The legal "*whitespace characters*" in this problem are blanks and the line terminator characters.

Given a sequence of N words of width $w_1, w_2, \ldots, w_N$, and a maximum line width L, with the guarantee that for all i, $w_i \leq L$, define $w(i, j)$ as the width of the line containing words i through j, inclusive, plus one blank space between each pair of words.

Then we can define the raggedness of a line containing words i though j as

$$r(i, j) = (L - w(i, j))^2$$

Write a program to read paragraphs of text and to lay them out in a way that no line contains more than L characters, for a specified L, and so that you minimize the total raggedness added up over all lines except the last one. (The final line of a paragraph can be arbitrarily shorter then the lines above it.) Line terminator characters are not counted as part of the line width.

Input

Input will consist of one or more datasets.

Each dataset begins with a line containing one integer, L, denoting the maximum line width (not counting line terminator characters). You are guaranteed that $0 < L \leq 80$. A value of zero indicates the end of input.

The remainder of the dataset consists of up to 250 lines containing a paragraph of text, terminated by an empty line. Paragraphs may contain from 1 to 500 words, where a word is any consecutive sequence of non-whitespace characters.

No line of text will contain a word of length greater than L.

Output

Print each paragraph laid out optimally as described above. After each paragraph print a line containing '===' (three equal signs).

If there is more than one way to fill a paragraph with the optimal raggedness, any such layout may be printed.

Sample Input

```
6
See if we
care.

25
Raggedy, raggedy are we.
Just as raggedy as raggedy can be.
We don't get nothin' for our labor.
So raggedy, raggedy are we.
- P Seeger

0
```

Sample Output

```
See
if we
care.
===
Raggedy, raggedy are
we. Just as raggedy
as raggedy can be. We
don't get nothin' for
our labor. So raggedy,
raggedy are we. - P
Seeger
===
```

Problem H: Painting the Floodwall

The town of South Riverside has a long floodwall to protect the residents against occasional rising waters from the nearby Little Muddy river. It's very functional, but also a bit of an eyesore.

The town has decided to spruce it up by staging a competition for local artists to paint murals on sections of the wall. Artists have submitted applications for the contest, in which they have specified not only how long a section of wall they want to paint but also, based upon the surrounding scenery, where along the wall they would like to place their mural.

Obviously, the artists' work cannot overlap, so there is a possibility that not all artists' applications can be accepted. On the other hand, the town would like to see as much of the wall painted as possible.

Find the combination of artists whose applications can be accepted to maximize the amount of the wall painted without allowing any artists' work to overlap. A mural that starts at the same coordinate at which another mural stops is not considered to overlap.

Input

Input will begin with a line containing an integer N denoting the number of artists whose have submitted applications. $1 \leq N \leq 200{,}000$

This will be followed by N lines, each containing two integers x_0 and x_1, $0 \leq x_0 \leq x_1 \leq 10^{18}$, denoting a starting and ending position (inclusive) for a proposed mural.

Output

Print a single line containing the maximum total length of the fence that can be painted without allowing any two artists' work to overlap.

Example

Sample Input	Sample Output
3 100 200 190 210 200 300	200
5 0 5 12 18 4 14 7 9 17 18	13

The second example reflects a decision to accept the applications to paint portions $0 \ldots 5$, $7 \ldots 9$, and $12 \ldots 18$.

Super Edit Distance[1]

In order to transform some source string a to a target string b, the following operations are allowed:

- Insert: insert a character into the intermediate buffer, don't move cursor in a
- Delete: delete a character from a, (otherwise known as not copying it into the intermediate buffer)
- Copy: copy the character from a into the current buffer
- Substitute: take the character in a and insert some other character c into the intermediate buffer
- Twiddle: take the next two characters in a and switch them and put the switched result into the intermediate buffer
- Kill: delete the remainder of a, can only be used as the last operation

If the cost of all operations except for copy is 1, what's the minimal cost to edit one string from another.

Example

If we were to edit "algorithm" to the target string "altruistic", the procedure would be:

Operation	x	z
initial strings	a̲lgorithm	_
copy	al̲gorithm	a_
copy	alg̲orithm	al_
replace by t	algo̲rithm	alt_
delete	algor̲ithm	alt_
copy	algori̲thm	altr_
insert u	algori̲thm	altru_
insert i	algori̲thm	altrui_
insert s	algori̲thm	altruis_
twiddle	algorit̲hm	altruisti_
insert c	algorit̲hm	altruistic_
kill	algorithm_	altruistic_

Which would lead to the total cost of 7

[1]Problem modified from Introduction to Algorithms by Cormen et al.