This lab will be focused on developing pseudocode analyzation and implementation of proper code from pseudocode.

Below the pseudocode of quicksort and the implementation of quick sort are given. Analyze this and perform rest of your tasks.

Pseudocode

```
QUICKSORT(A, p, r)
1  if p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q − 1)
4       QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

C++ Implementation

```
#include <iostream>
#define N 11

using namespace std;

int Partition(int A[],int p,int r);
int Arr[N+1];

int QuickSort(int A[],int p, int r)
{
   if (p<r)
   {
      int q = Partition(A,p,r);
      QuickSort(A,p,q-1);
      QuickSort(A,q+1,r);
   }
   else
      return 0;
}
```

```
int Partition(int A[],int p,int r)
{
   int x = A[r];
   int i = p-1;
   for(int j = p;j<r;j++)
   {
      if(A[j]<=x)
      {
         i = i+1;
         int temp = A[i];
         A[i] = A[j];
         A[j] = temp;
      }
   }
   int temp = A[i+1];
   A[i+1] = A[r];
   A[r] = temp;

   return i+1;
}
```

```
int main()
{
   for(int i=1;i<=N;i++)
      cin >> Arr[i];

   QuickSort(Arr,1,11);

   for(int i=1;i<=N;i++)
      cout << Arr[i] << " ";
}
```

Set A

1. Below the pseudocode of Heap Sort is given. Try to build of the functionality of heap sort from the pseudocode.

To build heap sort you have to follow the following Procedure. Each procedure has its own pseudocode and you have to convert each procedure to build the heap sort properties.

a) First you have to build a Heap Property like Max-Heap or Min-Heap. Here the pseudocode of Max-Heap is given

```
MAX-HEAPIFY(A, i)
1   l = LEFT(i)
2   r = RIGHT(i)
3   if l ≤ A.heap-size and A[l] > A[i]
4       largest = l
5   else largest = i
6   if r ≤ A.heap-size and A[r] > A[largest]
7       largest = r
8   if largest ≠ i
9       exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

b) To use this Max-Heap procedure you have to use the Max-Heap Procedure. Pseudocode of Build-Max-Heap is given below

```
BUILD-MAX-HEAP(A)
1   A.heap-size = A.length
2   for i = ⌊A.length/2⌋ downto 1
3       MAX-HEAPIFY(A, i)
```

c) Finally, you have to Build a Heap-Sort procedure to use Build-Max-Heap & Max-Heap Procedure. Pseudocode is given below.

```
HEAPSORT(A)
1   BUILD-MAX-HEAP(A)
2   for i = A.length downto 2
3       exchange A[1] with A[i]
4       A.heap-size = A.heap-size − 1
5       MAX-HEAPIFY(A, 1)
```

1. Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \longleftarrow 3n + 1$
5. else $n \longleftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

2. An Ackermann function has the characteristic that the length of the sequence of numbers generated by the function cannot be computed directly from the input value. One particular integer Ackermann function is the following:

$$X_{n+1} := \begin{cases} \frac{X_n}{2} & \text{if } X_n \text{ is even} \\ 3X_n + 1 & \text{if } X_n \text{ is odd} \end{cases}$$

This Ackermann has the characteristic that it eventually converges on 1. A few examples follow in which the starting value is shown in square brackets followed by the sequence of values that are generated, followed by the length of the sequence in curly braces:

```
[10] 5 16 8 4 2 1 {6}
[13] 40 20 10 5 16 8 4 2 1 {9}
[14] 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 {17}
[19] 58 29 88 44 22 ... 2 1 {20}
[32] 16 8 4 2 1 {5}
[1] 4 2 1 {3}
```

Input

Your program is to read in a series of pairs of values that represent the first and last numbers in a closed sequence. For each closed sequence pair determine which value generates the longest series of values before it converges to 1. The largest value in the sequence will not be larger than can be accomodated in a 32-bit Pascal LongInt or C long. The last pair of values will be '0', '0'.

Output

The output from your program should be as follows:

`Between` L `and` H`,` V `generates the longest sequence of` S `values.`

Where:

L = the lower boundary value in the sequence
H = the upper boundary value in the sequence
V = the first value that generates the longest sequence, (if two or more values generate the longest sequence then only show the lower value)
S = the length of the generated sequence.

In the event that two numbers in the interval should both produce equally long sequences, report the first.

Sample Input

```
  1 20
 35 55
  0 0
```

Sample Output

```
Between 1 and 20, 18 generates the longest sequence of 20 values.
Between 35 and 55, 54 generates the longest sequence of 112 values.
```

3. Everybody sit down in a circle. Ok. Listen to me carefully.

"Woooooo, you scwewy wabbit!"

Now, could someone tell me how many words I just said?

Input

Input to your program will consist of a series of lines, each line containing multiple words (at least one). A "word" is defined as a consecutive sequence of letters (upper and/or lower case).

Output

Your program should output a word count for each line of input. Each word count should be printed on a separate line.

Sample Input

```
Meep Meep!
I tot I taw a putty tat.
I did! I did! I did taw a putty tat.
Shsssssssssh ... I am hunting wabbits. Heh Heh Heh Heh ...
```

Sample Output

```
2
7
10
9
```

4. Joana loves playing with odd numbers. In the other day, she started writing, in each line, an odd number of odd numbers. It looked as follows:

```
 1
 3  5  7
 9 11 13 15 17
19 21 23 25 27 29 31
...
```

On a certain line Joana wrote 55 odd numbers. Can you discover the sum of the last three numbers written in that line? Can you do this more generally for a given quantity of odd numbers?

Given the number N of odd numbers in a certain line, your task is to determine the sum of the last three numbers of that line.

Input

The input is a sequence of lines, one odd number N ($1 < N < 1000000000$) per line

Output

For each input line write the sum of the last three odd numbers written by Joana in that line with N numbers. This sum is guaranteed to be less than 2^{63}.

Sample Input

```
3
5
7
```

Sample Output

```
15
45
87
```