Indian Institute of technology, Guwahati
Department of Computer Science and Engineering
Data Structure Lab: (CS210)
Offline Assignment: 6

Date: 12th September 2017 **Total Marks: 50**
Deadline: 10PM, 8th October 2017. (Hard Deadline)

1. **[Strange Alphabat]** The ancient civilization of America used the same 26 letters as modern English: 'a'-'z'. However, we do not know the order in which these letters appeared in the American alphabet.

   One particular custom in America was that in a good word the letters appear in non-decreasing order. For example, in English the word "ciel" is not a good word because in the alphabet 'i' is after 'e'. The word "ceil" is a good word because 'c' <= 'e' <= 'i' <= 'l'.

   You are given a set of words where each word is a nonempty string of lowercase English letters. Return "Possible" if it is possible that all words were good words in American, and "Impossible" otherwise. In other words, return "Possible" if and only if there is at least one possible American alphabet ordering such that the letters in word are in non-decreasing alphabetical order. **[10]**

   Constraints
   words has between 2 and 100 elements inclusive.
   The size of each element of **words** will be between 1 and 100 inclusive.
   Elements of **words** contains only English lowercase letters from 'a' to 'z'.

   Input Format
   Number of words k followed by k space-separated words.

   Output Format
   Print either “Possible” if constraints are met, otherwise “Impossible”

   Test 1:

   Input:
   3
   single round match

   Output:
   Possible

   Explanation:
   A possible alphabet ordering would be "bfjkmapqrositchundglevwxyz"

Test 2:

Input:
2
algorithm titan

Output:
Impossible

Explanation:
The word "titan" can never be a good word. The character 't' cannot be both before 'i' and after 'i' in the alphabet.

2. [**Cake Shop]** Subash owns a cake shop. In a normal restaurant, a customers are served by following the first-come, first-served (FCFS) rule, Subash simply minimizes the average waiting time of his customers. So he decides who is served first, regardless of how sooner or later a person comes.

   Different varieties of cakes take different amounts of time to cook. Also, once he starts cooking a cake, he cannot cook another cake until the first one is completely cooked. The waiting time is calculated as the difference between the time a customer orders cake (the time at which they enter the shop) and the time he is served.

   Help Subash achieve the minimum average waiting time. For the sake of simplicity, just find the integer part of the minimum average waiting time**.** **[10]**

   Note: Cook does not know about the future orders.

   Input Format

   The first line contains an integer **N**, the number of customers.

   Following N lines contain two space-separated integers o, time when order is placed and c, cooking time for order.

   Output Format

   Display the integer part of the minimum average waiting time.

   Constraints

   - $1 \leq N \leq 10^5$
   - $0 \leq T_i \leq 10^9$
   - $1 \leq L_i \leq 10^9$

Test 1:

Input:
3
0 3
1 9
2 6

Output:

9

Test 2:

Input:
3
0 3
1 9
2 5

Output:

8

Explanation: Minimum Average waiting time is (3 + 6 +16) / 3 = 8.33. So answer will be 8

3. Harshit has recently learned "Binary Heap" and implemented the following operations,
(i) insert element to heap
(ii) get the minimum value from the heap
(iii) extract the minimum element from the heap.

To learn this data structure better, Harshit took an empty heap and applied some operations. Also, he wrote down all the operations for logging purpose, in the following format
(i) insert x = inserted the element x into heap.
(ii) getmin x = the value of minimum element in heap was x.
(iii) removemin = deleted the minimum element from the heap. (delete only one instance, if many minimum elements) [10]

Constraints:
1) At any time, the heap can contain many integers or none, where some might be equal.
2) When getmin / removemin is applied, the heap is guaranteed to have at least one element.

When harshit was away for dinner, his little sister(Khyati) came and erased few operations in the log.

Now, Khyati is worried about making Harshit's sequence inconsistent. For example, if we apply operations in the order of log, results of getMin operations might differ from the results of Harshit, and some of getMin or removeMin operations may be inconsistent, as the heap is empty when applied.

Now, Khyati wants to add some operations in log to make them consistent. (result of getmin should match with the result in the record and heap is non-empty during getmin and removemin)
Now, Khyati wants to fix this asap and she is asking you to add least possible number of operations to the log.

Note: arbitrary number of operations can be added at the beginning of log/between operations / end of the log.

Input Format:
first line will have n(1<= n <= 1,00,000), current number of records in Harshit's log
The following "n" line are the operations described in the above format .
All the numbers in input, will be integers not exceeding 10^9.

Output Format:
first line print m, minimum possible number of records in modified sequence.
Next m lines, describe the corrected sequence of operations in the above format.(one per line)

Note: the input sequence must be the subsequence of the output sequence.

It is guaranteed that, there exists one correct answer.

Sample Input and Output:
Test 1:

Input
2
insert 6
getMin 7

Output
4
insert 6
removeMin
insert 7
getMin 7

Test 2:
Input

4
insert 6
insert 6
removeMin
getMin 9

Output
6
insert 6
insert 6
removeMin
removeMin
insert 9
getMin 9

Test 3:
Input
2
removeMin
getMin 31

Output
4
insert 0
removeMin
insert 31
getMin 31

4. **Implementation of Binomial Heap:** Implementing binomial heaps will require coding the operations: **(20)**
   - **MAKE-BINOMIAL-HEAP**
   - **BINOMIAL-HEAP-UNION**
   - **BINOMIAL-HEAP-INSERT**
   - **BINOMIAL-HEAP-EXTRACT-MIN**

   In addition, you need to write the routine showBinomialHeap which displays the binomial heap structure graphically (see below), rotated 90 degrees.
   You need to implement a command based interface to test your programs. A sample command is given below

• MAKE-BINOMIAL-HEAP • BINOMIAL-HEAP-UNION • BINOMIAL-HEAP-INSERT • BINOMIAL-HEAP-EXTRACT-MIN	c u i d

• Turn off Print after extract min • Turn on print after extract min	+ -

Sample Test Case 1:

```
+-------------------------------------------------------------------+
|# create empty heap                                                |
|                                                                   |
|# insert 20 keys                                                   |
|                                                                   |
|# show heap structure                                              |
|                                                                   |
|Structure of binomial heap (rotated 90 degrees counterclockwise):|
|                                                                   |
|    1     2                                                        |
|                                                                   |
|          3     4                                                  |
|                                                                   |
|          5     6                                                  |
|                                                                  |
|                7     8                                            |
|                                                                   |
|          9     10                                                 |
|                                                                  |
|                11    12                                           |
|                                                                   |
|                13    14                                           |
|                                                                  |
|                      15    16                                     |
|                                                                   |
|                                                                   |
|    17    18                                                       |
|                                                                   |
|          19    20                                                 |
|                                                                   |
|                                                                   |
|# do 10 deletes (extract minimums) with print key flag off         |
|                                                                   |
|# show heap structure                                              |
|                                                                   |
|Structure of binomial heap (rotated 90 degrees counterclockwise):|
|                                                                   |
|    13    14                                                       |
|                                                                   |
|          15    16                                                 |
|                                                                   |
|          17    18                                                 |
|                                                                   |
|                19    20                                           |
|                                                                   |
|                                                                   |
|    11    12                                                       |
|                                                                   |
|                                                                   |
|# do 10 deletes (extract minimums) with print key flag on          |
```

```
|                                                                    |
|Minimum extracted: 11                                               |
|Minimum extracted: 12                                               |
|Minimum extracted: 13                                               |
|Minimum extracted: 14                                               |
|Minimum extracted: 15                                               |
|Minimum extracted: 16                                               |
|Minimum extracted: 17                                               |
|Minimum extracted: 18                                               |
|Minimum extracted: 19                                               |
|Minimum extracted: 20                                               |
|                                                                    |
|# show heap structure (should be empty)                             |
|                                                                    |
|Structure of binomial heap (rotated 90 degrees counterclockwise):|
|                                                                    |
|                                                                    |
|                                                                    |
|# do a delete from empty heap                                       |
|Heap Empty                                                          |
|                                                                    |
|# quit                                                              |
+--------------------------------------------------------------------+
```

Sample Test Case 2: See binomialHeapTest.txt

Displaying the Structure of a Binomial Heap

One way to do this is to let **`showBinomialHeap`** be a non-recursive "driver" that calls a recursive routine **`showHeap`** to display a binomial heap rotated 90 degrees.
So **`showHeap`** could be something like:

```
showHeap( x, depth )
   if ( sibling[x] != NIL ) then      |> Do sibling list first
      showHeap( sibling[x], depth )

   |> May have to print a blank line here

   if ( ( child[x] != NIL ) or ( p[x] = NIL ) ) then
      print key[x] shifted 6*depth + 4 spaces     |> First key on a line
      if ( child[x] = NIL ) print a blank line    |> Special case
   else
      print key[x] shifted 6 spaces, then a blank line  |> Last key on line

   if ( child[x] != NIL ) then        |> Take care of children last
      showHeap( child[x], depth + 1 )
```

And **showBinomialHeap** would be something like:

```
showBinomialHeap(H)
   print "Stucture of binomial heap (rotated 90 degrees ccwise):"
   if ( head[H] = NIL ) then print "Empty heap"
                        else showHeap( head[H], 0 )
```

You may have to adjust the formatting to get good looking output.