PROBLEMA O

ONDE ESTÃO AS BOLHAS?

Nome do arquivo fonte: Bolha.{py|java|c|cpp}

Uma das operações mais frequentes em computação é ordenar uma sequência de objetos. Portanto, não é surpreendente que essa operação seja também uma das mais estudadas.

Um algoritmo bem simples para ordenação é chamado *Bubblesort*. Ele consiste de vários turnos. A cada turno o algoritmo simplesmente itera sobre a sequência trocando de posição dois elementos consecutivos se eles estiverem fora de ordem. O algoritmo termina quando nenhum elemento trocou de posição em um turno.

O nome *Bubblesort* (ordenação das bolhas) deriva do fato de que elementos menores ("mais leves") movem-se na direção de suas posições finais na sequência ordenada (movem-se na direção do início da sequência) durante os turnos, como bolhas na água. A figura abaixo mostra uma implementação do algoritmo em pseudo-código:

Para i **variando de** 1 **a** N **faça**
    Para j **variando de** $N - 1$ **a** i **faça**
        Se $seq[j-1] > seq[j]$ **então**
            Intercambie os elementos $seq[j-1]$ e $seq[j]$
        Fim-Se
    Fim-Para
    Se nenhum elemento trocou de lugar **então**
        Final do algoritmo
    Fim-Se
Fim-Para

Por exemplo, ao ordenar a sequência [5, 4, 3, 2, 1] usando o algoritmo acima, quatro turnos são necessários. No primeiro turno ocorrem quatro intercâmbios: 1×2, 1×3, 1×4 e 1×5; no segundo turno ocorrem três intercâmbios: 2 × 3, 2 × 4 e 2 × 5; no terceiro turno ocorrem dois intercâmbios: 3 × 4 e 3 × 5; no quarto turno ocorre um intercâmbio: 4 × 5; no quinto turno nenhum intercâmbio ocorre e o algoritmo termina.

Embora simples de entender, provar correto e implementar, o algoritmo *bubblesort* é muito ineficiente: o número de comparações entre elementos durante sua execução é, em média, diretamente proporcional a N^2, onde N é o número de elementos na sequência.

Você foi requisitado para fazer uma "engenharia reversa" no *bubblesort*, ou seja, dados o comprimento da sequência, o número de turnos necessários para a ordenação e o

número de intercâmbios ocorridos em cada turno, seu programa deve descobrir uma possível sequência que, quando ordenada, produza exatamente o mesmo número de intercâmbios nos turnos.

ENTRADA

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e M que indicam respectivamente o número de elementos ($1 \leq N \leq 100.000$) na sequência que está sendo ordenada, e o número de turnos ($0 \leq M \leq 100.000$) necessários para ordenar a sequência usando *bubblesort*. A segunda linha de um caso de teste contém M inteiros X_i, indicando o número de intercâmbios em cada turno i ($1 \leq X_i \leq N - 1$, para $1 \leq i \leq M$).

O final da entrada é indicado por $N = M = 0$. A entrada deve ser lida da entrada padrão.

SAÍDA

Para cada caso de teste da entrada seu programa deve produzir uma linha na saída, contendo uma permutação dos números {1,2,...,N}, que quando ordenada usando *bubblesort* produz o mesmo número de intercâmbios no mesmo número de turnos especificados na entrada. Ao imprimir a permutação, deixe um espaço em branco entre dois elementos consecutivos. Se mais de uma permutação existir, imprima a maior na ordem lexicográfica padrão para sequências de números (a ordem lexicográfica da permutação $a_1, a_2, \ldots a_N$ é maior do que a da permutação $b_1, b_2, \ldots b_N$ se para algum $1 \leq i \leq N$ temos $a_i > b_i$ e o prefixo $a_1, a_2, \ldots a_{i-1}$ é igual ao prefixo $b_1, b_2, \ldots b_{i-1}$) .

Em outras palavras, caso exista mais de uma solução, imprima aquela onde o primeiro elemento da permutação é o maior possível. Caso exista mais de uma solução satisfazendo essa restrição, imprima, dentre estas, aquela onde o segundo elemento é o maior possível. Caso exista mais de uma solução satisfazendo as duas restrições anteriores, imprima, dentre estas, a solução onde o terceiro elemento é o maior possível, e assim sucessivamente. Para toda entrada haverá pelo menos uma permutação solução.

EXEMPLO DE ENTRADA	EXEMPLO DE SAÍDA
3 1 1 5 4 4 3 2 1 6 5 2 2 2 2 1 0 0	2 1 3 5 4 3 2 1 6 5 1 2 3 4