**Problem A**
**Author: S. M. Masrur Ahmed**
Try all numbers from 1 to N. For a fixed number X, extract all its digits and check if there is at least one 1 in the number.

**Problem B**
**Author: Tariq Sajal**
If you make a negative cycle which is reachable from source then the shortest path of every node in this cycle and every node which is reachable from this cycle will be undefined. So decompose the graph into several strongly connected components and consider one scc a node. For each scc, if there are more than one node, call them good scc. For each good scc ‘U’ , if there is no good scc ‘V’ such that U is reachable from V then we need to make a negative cycle in U. So we do it in topsort order. For each unvisited good scc ‘U’, we find a cycle of minimum length in U and make it negative and mark each good scc ‘V’ visited which is reachable from U. To find a cycle of minimum length, iterate over edges which are inside good scc ‘U’. Let’s say the edge is from node ‘a’ to node ‘b’ with weight ‘c’, then the cycle with minimum length which contains this edge is c + shortest dis from ‘b’ to ‘a’.

**Problem C**
**Author: Sabbir Rahman Abir**
Via one swap we can create at most 2 new mismatches. So we’ll first try to apply swaps at indices i, i+1, where p[i] = i and p[i+1] = i+1. After this process if there are still swaps remaining, then it is possible some isolated matchings i will remain, i.e. p[i] = i but p[i-1] != i-1 and p[i+1] != i+1. Now only one mismatch can be created by one swap, we can swap these isolated i with a neighbour to create those mismatches (no new matching is created). So in the end what happens is, consecutive matching blocks of even size are paired and consecutive matching blocks of odd size are paired with one isolated matching index remaining.

**Problem D**
**Author: Arghya Pal**
No solution when n = even, proof is left as an exercise to the readers :) We can always build a solution when n is odd.

Let’s make everything 0 indexed. So we need to put numbers (0...n-1) in the matrix a[0….n-1][0….n-1] such that a[i][i] = i and a[i][j] = a[j][i] and each row have all the distinct numbers from 0...n-1 exactly once.

We can reformulate the problem like this, build a complete graph of n nodes where each edge has a color from 0 to n-1, and also all the nodes have a color from 0 to n-1. For any node the n colors(it’s own color and the colors of n-1 edges incident to it) should be distinct.(n distinct numbers from 0 to n-1).

Intuitively, color the nodes with 0 to n-1 and write (i+j) mod n to the edge between (i,j). Now for all nodes the edges incident to it have different colors. How about nodes ? Just write (2*i) mod n in the node colored with i. As n is odd, all the numbers (2*i) mod n where i=0...n-1 should be different. And for each node it's color is different from all it's incident edges. Now we can put the i'th node's color in cell a[i][i] and (i,j) edge's color in cell a[i][j] and a[j][i].

It turns out that we can also simply write, a[i][j] = (( i+j )/2) mod n.

Alternatively you can simply notice some pattern and use it.

**Problem E**
**Author: Ashiqul Islam**
Sweep from left to right and fix K. We need to find how many pairs (i, j) are there on the left such that a[i] >= a[j], i < j, a[i] <= a[k] and a[j] <= a[k]. Let's count the reverse - how many pairs (i, j) are there in the left such that a[i] < a[j], i < j, a[i] <= a[k] and a[j] <= a[k]. To calculate that, we will build a segment tree on values. For each value, V, we will keep their frequency and maintain the count of (i, j) such that, a[j] = V, a[i] < a[j] and i < j. Then the problem reduces to basic range sum query and point update problem.

**Problem F**
**Author: Arghya Pal**
It turns out that the answer for a query is the count of inversion in the substring.
To calculate inversion in a substring, let's precalculate,
Inversion[i] = inversion of prefix $s_1s_2....s_i$
Then, Inversion(l, r) = Inversion[r] - Inversion[l-1] - (Inversion count where the left element comes from $s_1s_2..s_{l-1}$ and right element from $s_ls_{l+1}..s_r$)
The last term in the above can be calculated in O(ALPHA) using prefix sum.
ALPHA is the count of distinct possible characters, here ALPHA = 26.

**Problem G**
**Author: Arghya Pal**
At first map the initial permutation to identity permutation(1,2,3….n). Then map the final permutation accordingly. Now think about any operation sequence and the elements not touched by any operation of that sequence. They would form an increasing suffix of the final permutation. Let's say that the mapped final permutation is ,

5 3 1 6 2 4 7

We can separate it into two parts 5 3 1 6 and the increasing suffix 2 4 7.

Note that the numbers untouched by the operation sequence forms an increasing suffix.
So the operation sequence must include (5,3,1,6) and might include some prefix of the (2,4,7).

Now let's consider the last operation in the sequence, it must be 5 and it doesn't matter where else this 5 occurs in the operation sequence. So if we erase all 5 from the operation sequence, what is the last element in the remaining operation sequence ? Yeah, it is 3. Now if we erase all

the 3's from the operation sequence, what is the last element in the remaining sequence ? It is 1.

So, it turns out that the solution in this case is S(m,4) + S(m,5) + S(m,6) + S(m,7).

Where S(i,j) is how many ways we can group i elements in j groups., which is actually the stirling number of the second kind. You can easily compute it with recurrences.

Note that you can also solve it for n,m <= 10^5, where you'll be calculating the stirling number using NTT, although it wasn't necessary for this problem as the constraint was much smaller.

**Problem H**
**Author: Ashiqul Islam**

Let's assume, N = $2^K$
In this case, it's always optimal to connect 0, 1, 2, ..., N-2 nodes to N-1 forming a star.
But what if N is not a power of two. In this case, find the largest k such that $2^K$ < N and split the nodes into two parts.
0, 1, 2, ...., $2^K$-1
and
$2^K$, $2^{K+1}$, ..., N-1

For the first part, all the nodes 0, 1, 2, ...., $2^K$-2 will connect to $2^K$-1 forming a connected component. Now add an edge between $2^K$-1 and N-1.
But we have to merge the second parts as well. Now if we erase the MSB of all the values, the problem reduces to the same problem we are solving with smaller number of nodes which leads to an easy linear dp solution.
Note that you can also optimize this solution to logN if you notice that, after each step, we reduce the number of nodes to at least half.

**Problem I**
**Author: Sabbir Rahman Abir**

$(\sqrt{n}+\sqrt{n+1})^2 = A + B\sqrt{S}$ `where` $A = 2n+1, B = 2, S = n(n+1)$
These numbers of format $A + B\sqrt{S}$ multiply to again form numbers of same format (like complex numbers).

So we can write $(A + B\sqrt{S})^k = C + D\sqrt{S} = 2C - (C - D\sqrt{S})$

The $C$ and $D$ can be found by keeping track of $A$, $B$ and multiplying in a manner similar to binary exponentiation. And since $(\sqrt{n+1} - \sqrt{n})^{2k} = (A - B\sqrt{S})^k = C - D\sqrt{S}$ is less than 1. So $\lfloor C + D\sqrt{S} \rfloor = 2C - 1$