Qooqle is expanding due to the sudden availability of swarms of US software engineers who want to move as far north as possible. As a reward for your excellent work on project scheduling, you are now in charge of matching each of the new employees with a task.

Qooqle is providing you with an $n \times n$ matrix $Q$ in which each row represents a new employee and each column represents a task. In the matrix $Q[i][j]$ represents the number of hours it will take person $i$ to complete task $j$. The times can vary significantly because each new employee has a different skill set.

Each employee needs to be assigned to a different task (and thus each task must be assigned to exactly one person). Your goal is to minimize the total number of hours required to complete all the tasks.

For example, consider this instance of the problem

<table>
<thead>
<tr>
<th></th>
<th>Module 1</th>
<th>Module 2</th>
<th>Module 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kim</td>
<td>7</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>Pat</td>
<td>3</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Lee</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

If Kim is assigned Module 1 and Pat gets Module 2 and Lee gets Module 3, then the total number of hours is 20. But if Kim is assigned Module 2 and Pat gets Module 1 and Lee gets Module 3, the total number of hours is 12.

There are $n!$ possible assignments of persons to tasks. We certainly don’t want to check them all. We can see that a simple greedy algorithm won’t work – consider using “make the least-cost assignment out of all available assignments, and apply the same rule to all remaining possible assignments”. This would assign Pat Module 1, Lee Module 2 and Kim Module 3, for a total cost of 21. We have already seen that this is not optimal.

(It turns out that there is a very clever and fast algorithm for solving this problem called The Hungarian Method. However the founder of Qooqle has a goulash allergy and won’t let you use this algorithm.)
You are required to implement a Branch&Bound solution to this problem. You will need to:

1. Define the objective function (it is actually given already)
2. Describe solutions as sequences of decisions (e.g., “The \( i^{th} \) decision will be to choose a task for person \( i \)”)
3. Compute an initial upper bound
4. Design lower bound and upper bound functions for partial solutions
5. Implement a min-heap for keeping track of partial solutions

For the purposes of this assignment, a collection of data files of various sizes is provided. The first line of each file contains a single integer that gives the number of rows and columns in the matrix. The remaining \( n \) lines each contain \( n \) integers, representing a complete row of the matrix. (The employees and tasks are not named, but feel free to invent names for them if you like.)

In this assignment you have room for a lot of creativity. Your main challenge is to design lower and upper bound functions that reduce the number of iterations of the algorithm and allow you to solve larger instances of the problem in a reasonable amount of time (which we will define as “no more than 120 seconds of elapsed real time”). You will not be graded on the sophistication of your bounding functions as long as they are non-trivial.

Your deliverables are

1. Your fully documented code, including an explanation of your bounding functions
2. The minimum cost solution for as many of the problem instances as you were able to solve. You do not need to show the details of the optimal assignments – you only need to show the final total cost of the optimal assignment.

As always, you can work in pairs on this assignment. Your solution must be coded in Python, Java, C or C++.

This assignment is due at midnight on Monday December 5.