The package that we used for the solution in solver.py are constraint and operator.

The first step, we construct the problem instance with the problem() method in constraint.Problem().

Second, after we have a problem, we add variables, domain of variables to the problem instance so that the problems knows what variables are required to solve.

Third, we add constraints to the problem instance. Since the constraints are written in form "wizard1 wizard2 wizard 3" where wizard3' s age is not between that of wizard1 and wizard2, we can say that the constraints is "wizard3 < wizard1 and wizard3 < wizard2" or "wizard3 > wizard1 and wizard3 > wizard2" . Then, we add the translated all constraints to the problem instances.

Fourth, after adding all constraints, we now use a backtracking solver to solve the problem so that there is a solution that satisfies all constraints. We try one value for each variable, if it satisfies the constraint, then we move forward to the next constraint, otherwise we trace back to last time we assign a value and assign a different value. Repeat this process until we have a solution that satisfies all constraints. After getting a result, we output result as the instruction said.

Code is in next page:

```
import argparse
from constraint import *
import operator

"""
==================================================
==================
   Complete the following function.
==================================================
==================
"""

def solve(num_wizards, num_constraints, wizards, constraints):
    """
    Write your algorithm here.
    Input:
        num_wizards: Number of wizards
        num_constraints: Number of constraints
        wizards: An array of wizard names, in no particular order
        constraints: A 2D-array of constraints,
                        where constraints[0] may take the form ['A', 'B', 'C']i

    Output:
        An array of wizard names in the ordering your algorithm returns
    """
    ages = []
    resultList = []

    for i in range(1, num_wizards + 1):
        ages.append(i)

    problem = Problem()
    wizardsAges = {}
```

```
    for wizard in wizards:
        problem.addVariable(wizard, ages)

    for constraint in constraints:
        problem.addConstraint(lambda a, b, c: (c < a and c < b) or (c > a and c >
b), (constraint[0], constraint[1], constraint[2]))

    resultDict = problem.getSolution()
    sorted_tuples = sorted(resultDict.items(), key = operator.itemgetter(1))

    for each in sorted_tuples:
        resultList.append(each[0])

    return resultList


"""
==================================================
====================
    No need to change any code below this line
==================================================
====================
"""

def read_input(filename):
    with open(filename) as f:
        num_wizards = int(f.readline())
        num_constraints = int(f.readline())
        constraints = []
        wizards = set()
        for _ in range(num_constraints):
            c = f.readline().split()
            constraints.append(c)
```

```
                for w in c:
                    wizards.add(w)

    wizards = list(wizards)
    return num_wizards, num_constraints, wizards, constraints

def write_output(filename, solution):
    with open(filename, "w") as f:
        for wizard in solution:
            f.write("{0} ".format(wizard))

if __name__=="__main__":
    parser = argparse.ArgumentParser(description = "Constraint Solver.")
    parser.add_argument("input_file", type=str, help = "___.in")
    parser.add_argument("output_file", type=str, help = "___.out")
    args = parser.parse_args()

    num_wizards, num_constraints, wizards, constraints =
read_input(args.input_file)
    solution = solve(num_wizards, num_constraints, wizards, constraints)
    write_output(args.output_file, solution)
```