Python code

```
!pip install z3-solver

from z3 import *
from pathlib import Path
from timeit import default_timer as timer
import re

start = timer()
class Instance:
    def __init__(self):
        self.number_of_students = 0
        self.number_of_exams = 0
        self.number_of_slots = 0
        self.number_of_rooms = 0
        self.room_capacities = []
        self.exams_to_students = []
        self.student_exam_capacity = []

def read_file(filename):
    def read_attribute(name):
        line = f.readline()
        match = re.match(f'{name}:\\s*(\\d+)$', line)
        if match:
            return int(match.group(1))
        else:
            raise Exception("Could not parse line {line}; expected the {name} attribute")
    instance = Instance()
    with open(filename) as f:
        instance.number_of_students = read_attribute("Number of students")
        instance.number_of_exams = read_attribute("Number of exams")
        instance.number_of_slots = read_attribute("Number of slots")
        instance.number_of_rooms = read_attribute("Number of rooms")

        for r in range(instance.number_of_rooms):
            instance.room_capacities.append(read_attribute(f"Room {r} capacity"))

        while True:
            l = f.readline()
            if l == "":
                break;
            m = re.match('^\\s*(\\d+)\\s+(\\d+)\\s*$', l)
            if m:
                instance.exams_to_students.append((int(m.group(1)), int(m.group(2))))
            else:
                raise Exception(f'Failed to parse this line: {l}')

        # create an empty array for the number of exams.
        for r in range(instance.number_of_exams):
            instance.student_exam_capacity.append(0)

        # make the array loop,count and increment the number of students in an exam
        for r in instance.exams_to_students:
            instance.student_exam_capacity[r[0]] += 1
    return instance

def solve(instance):
    # Implement your solver here
    s = Solver()
    # Declaration
    exam = Int('exam')
    room = Int('room')
    ts = Int('ts')
    nex = Int('nex')
    nts = Int('nts')
    student = Int('student')
    # from the previous labs, set range
    Student_Range = Function('Student_Range', IntSort(), BoolSort())
    Exam_Range = Function('Exam_Range', IntSort(), BoolSort())
    Room_Range = Function('Room_Range', IntSort(), BoolSort())
    TimeSlot_Range = Function('TimeSlot_Range', IntSort(), BoolSort())

    # ranges that are specifically assigned for the sat/unsat txt fies
    s.add(ForAll([student], Student_Range(student) == And(student >= 0, student <
instance.number_of_students)))
    s.add(ForAll([exam], Exam_Range(exam) == And(exam >= 0, exam <
instance.number_of_exams)))
    s.add(ForAll([ts], TimeSlot_Range(ts) == And(ts >= 0, ts <
instance.number_of_slots)))
    s.add(ForAll([room], Room_Range(room) == And(room >= 0, room <
instance.number_of_rooms)))
    # functions
    ExamRoom = Function('ExamRoom', IntSort(), IntSort())  # takes exam outputs
room
    ExamTime = Function('ExamTime', IntSort(), IntSort())  # takes exam outputs slot
    ExamStudent = Function('ExamStudent', IntSort(), IntSort(), BoolSort())
# Student taking the exam

    #To add (and show) the students
    for etos in instance.exams_to_students:
        s.add(ExamStudent(etos[0], etos[1]))

    # first and second constraint
    s.add(
        ForAll([exam],
               Implies(
                   Exam_Range(exam),
                   Exists([room, ts],
                          And(Room_Range(room),
                    TimeSlot_Range(ts),
                    ExamTime(exam) == ts,
                    ExamRoom(exam) == room,
                    ForAll([nex],
                        Implies(
                            Exam_Range(nex),
                            Implies(
                                And(
                                    ExamRoom(nex) == room,
                                    ExamTime(nex) == ts
                                ),
                                exam == nex
                            )
                        )
                        )
                    )
                )
            )
            )
    )

    # third constraint
    for ex2 in range(instance.number_of_exams):
        for rm2 in range(instance.number_of_rooms):
            s.add(Implies((ExamRoom(ex2) == rm2), instance.student_exam_capacity[ex2]
<= instance.room_capacities[rm2]))

    # fourth constraint
    s.add(
        ForAll(
            [student, nex, ts, nts, exam],
              Implies(
                And(
                    Student_Range(student),
                    Exam_Range(exam),
                    Exam_Range(nex),
                    TimeSlot_Range(ts),
                    TimeSlot_Range(nts),
                    Not((exam == nex))
                ),
                Implies(
                    And(
                        ExamTime(exam) == ts,
                        ExamTime(nex) == nts,
                        ExamStudent(exam, student),
                        ExamStudent(nex, student)
                    ),
                    And(
                        (ts + 1 != nts),
                        (ts - 1 != nts),
                    (ts != nts)
                )
            )
        )
    )
  )

  if s.check() == unsat:
    print('unsat')
  else:
    print('sat')
    for ex2 in range(instance.number_of_exams):
      print("   Exam: ", ex2, "  Room: ", (s.model().eval(ExamRoom(ex2))), "  Slot:
", (s.model().eval(ExamTime(ex2))))
    print("————————————————————————————————")

if __name__ == "__main__":

  #read one file one by one manually.
  '''
  inst = read_file('test instances/unsat10.txt')
  solve(inst)
  '''

  #read through all the files in the folder
  tests_dir = Path("/content/sample_data/test instances")
  for test in tests_dir.iterdir():
    if test.name != ".idea":
      instance = read_file(str(test))
      print(f"{test.name}: ", end="")
      solve(instance)

end = timer()
print('   \nElapsed time: ', int((end-start)*1000), 'milliseconds')
```

-------------- End of **CW1** Sample Guidance -------------