Q3)

INPUT

```python
class Student:
    def __init__(self, name, id_of_student):
        self.name = name
        self.id_of_student = id_of_student
        self.modules = []

    def add_module(self, subject, module_id, marks):
        self.modules.append({'Subject': subject, 'module_id': module_id, 'marks': marks})

    def remove_module(self, module_id):
        module_removed = False
        for module in self.modules:
            if module['module_id'] == module_id:
                self.modules.remove(module)
                module_removed = True
                print(f"Module with ID {module_id} removed for student {self.name}")
                break
        if not module_removed:
            print(f"Module with ID {module_id} not found for student {self.name}")

    def see_profile(self):
        print(f"Name of Student: {self.name}")
        print(f"Allocated ID for students: {self.id_of_student}")
        print("Modules:")
        for module in self.modules:
            print(f"  Subject: {module['Subject']}, Module ID: {module['module_id']}, marks: {module['marks']}")

    def compute_average_marks_received(self):
        if not self.modules:
            return 0
        sum_of_marks_received = sum(module['marks'] for module in self.modules)
        return sum_of_marks_received / len(self.modules)


class SchoolManagementSystem:
    def __init__(self):
        self.students = []

    def add_student(self, name, id_of_student):
        newcomers_student = Student(name, id_of_student)
        self.students.append(newcomers_student)
        print(f"Student {name} with ID {id_of_student} added.")

    def take_away_student(self, id_of_student):
        taken_away_student = next((student for student in self.students if student.id_of_student == id_of_student), None)
        if taken_away_student:
            self.students.remove(taken_away_student)
            print(f"Student with ID {id_of_student} taken away.")
        else:
            print(f"Student with ID {id_of_student} not available.")

    def find_student_by_id(self, id_of_student):
        return next((student for student in self.students if student.id_of_student == id_of_student), None)

    def find_highest_average_grade_student(self):
```

```
    def find_highest_average_grade_student(self):
        return max(self.students, default=None, key=lambda student: student.compute_average_marks_received())

    def find_lowest_average_grade_student(self):
        return min(self.students, default=None, key=lambda student: student.compute_average_marks_received())

    def sort_students_by_average_marks_received(self):
        if not self.students:
            print("No students available in the system.")
        else:
            sorted_students = sorted(self.students, key=lambda student: student.compute_average_marks_received())
            print("Sort students by Average marks received (ascending order):")
            for student in sorted_students:
                print(f"Name: {student.name}, Average Marks received: {student.compute_average_marks_received()}")


# Usage example:
school = SchoolManagementSystem()

# Adding students
school.add_student("Rachel", 1266845)
school.add_student("Catia", 1288975)

# Adding modules for students
rachel = school.find_student_by_id(1266845)
if rachel:
    rachel.add_module("Math", 110, 100)
    rachel.add_module("Science", 212, 99)

catia = school.find_student_by_id( 1288975)
if catia:
    catia.add_module("History", 157, 92)
    catia.add_module("English", 168, 78)

# Observing student profiles
if rachel:
    rachel.see_profile()

# Eliminating a module from a student's records
if rachel:
    rachel.remove_module(2)

# Finding students with the highest and lowest average marks
highest_average_student = school.find_highest_average_grade_student()
lowest_average_student = school.find_lowest_average_grade_student()

if highest_average_student:
    print(f"Highest average marks received: {highest_average_student.compute_average_marks_received()} for {highest_average_student.name}")
if lowest_average_student:
    print(f"Lowest average marks received: {lowest_average_student.compute_average_marks_received()} for {lowest_average_student.name}")

# Sorting students by average marks
school.sort_students_by_average_marks_received()
```

OUTPUT

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/rm6197r/.spyder-py3/untitled3.py', wdir='C:/Users/rm6197r/.spyder-py3')
Student Rachel with ID 1266845 added.
Student Catia with ID 1288975 added.
Name of Student: Rachel
Allocated ID for students: 1266845
Modules:
  Subject: Math, Module ID: 110, marks: 100
  Subject: Science, Module ID: 212, marks: 99
Module with ID 2 not found for student Rachel
Highest average marks received: 99.5 for Rachel
Lowest average marks received: 85.0 for Catia
Sort students by Average marks received (ascending order):
Name: Catia, Average Marks received: 85.0
Name: Rachel, Average Marks received: 99.5

In [2]:
```

CODE WRITTEN:

```
class Student:
    def __init__(self, name, id_of_student):
        self.name = name
        self.id_of_student = id_of_student
        self.modules = []

    def add_module(self, subject, module_id, marks):
        self.modules.append({'Subject': subject, 'module_id': module_id, 'marks': marks})

    def remove_module(self, module_id):
        module_removed = False
        for module in self.modules:
            if module['module_id'] == module_id:
                self.modules.remove(module)
                module_removed = True
```

```
            print(f"Module with ID {module_id} removed for student {self.name}")
            break
        if not module_removed:
            print(f"Module with ID {module_id} not found for student {self.name}")

    def see_profile(self):
        print(f"Name of Student: {self.name}")
        print(f"Allocated ID for students: {self.id_of_student}")
        print("Modules:")
        for module in self.modules:
            print(f"  Subject: {module['Subject']}, Module ID: {module['module_id']}, marks: {module['marks']}")

    def compute_average_marks_received(self):
        if not self.modules:
            return 0
        sum_of_marks_received = sum(module['marks'] for module in self.modules)
        return sum_of_marks_received / len(self.modules)


class SchoolManagementSystem:
    def __init__(self):
        self.students = []

    def add_student(self, name, id_of_student):
        newcomers_student = Student(name, id_of_student)
        self.students.append(newcomers_student)
        print(f"Student {name} with ID {id_of_student} added.")

    def take_away_student(self, id_of_student):
        taken_away_student = next((student for student in self.students if student.id_of_student == id_of_student), None)
```

```
        if taken_away_student:
            self.students.remove(taken_away_student)
            print(f"Student with ID {id_of_student} taken away.")
        else:
            print(f"Student with ID {id_of_student} not available.")

    def find_student_by_id(self, id_of_student):
        return next((student for student in self.students if student.id_of_student == id_of_student), None)

    def find_highest_average_grade_student(self):
        return max(self.students, default=None, key=lambda student: student.compute_average_marks_received())

    def find_lowest_average_grade_student(self):
        return min(self.students, default=None, key=lambda student: student.compute_average_marks_received())

    def sort_students_by_average_marks_received(self):
        if not self.students:
            print("No students available in the system.")
        else:
            sorted_students = sorted(self.students, key=lambda student: student.compute_average_marks_received())
            print("Sort students by Average marks received (ascending order):")
            for student in sorted_students:
                print(f"Name: {student.name}, Average Marks received: {student.compute_average_marks_received()}")


# Usage example:
school = SchoolManagementSystem()
```

```
# Adding students
school.add_student("Rachel", 1266845)
school.add_student("Catia", 1288975)

# Adding modules for students
rachel = school.find_student_by_id(1266845)
if rachel:
    rachel.add_module("Math", 110, 100)
    rachel.add_module("Science", 212, 99)

catia = school.find_student_by_id( 1288975)
if catia:
    catia.add_module("History", 157, 92)
    catia.add_module("English", 168, 78)

# Observing student profiles
if rachel:
    rachel.see_profile()

# Eliminating a module from a student's records
if rachel:
    rachel.remove_module(2)

# Finding students with the highest and lowest average marks
highest_average_student = school.find_highest_average_grade_student()
lowest_average_student = school.find_lowest_average_grade_student()

if highest_average_student:
    print(f"Highest average marks received:
{highest_average_student.compute_average_marks_received()} for
{highest_average_student.name}")
if lowest_average_student:
```

```
    print(f"Lowest average marks received:
{lowest_average_student.compute_average_marks_received()} for
{lowest_average_student.name}")


# Sorting students by average marks

school.sort_students_by_average_marks_received()
```

EXPLANATION

Student class:

A Student object is initialised with a name, student ID, and an empty list called modules to hold details about the modules the student is enrolled in by using the __init__ method.

A student's modules can be added, together with the subject, module ID, and grades earned, using the add_module method.

The remove_module function eliminates a module by using its ID.

The student's name, ID, and a list of modules together with their information are displayed using the see_profile method.

The compute_average_marks_received method computes and returns the student's average mark.

Class: School Management System:

A school administration system is initialised using the __init__ method, which creates an empty list called students to hold student objects.

A new Student object is created and added to the list of students using the add_student method.

By using their ID, the take_away_student method eliminates a student.

The "find_student_by_id" method uses the student's ID to retrieve the student.

The student with the highest and lowest average marks is found using the find_highest_average_grade_student and find_lowest_average_grade_student methods, respectively.

The sort_students_by_average_marks_received method publishes the results after sorting the students according to their average marks.

Use Case Study:

There is a creation of a school instance of the SchoolManagementSystem class.

With the IDs 1266845 and 1288975, two new pupils, "Rachel" and "Catia," are added to the system.

Each student has their modules uploaded, along with their subjects, module IDs, and grades.

It shows Rachel's profile.

A module bearing ID 2 is eliminated from Rachel's documentation.

We locate and display the student with the highest and lowest average marks.

..

Students are arranged and shown according to their average grade.

This code replicates the features of a simple student add/removal and profile viewing system for schools. Additionally, it offers the ability to rank pupils according to their average marks and identify those with the highest and lowest average marks. The usage example shows how to add students, add modules, and carry out additional actions in order to engage with the system.