DAA Assignment 2

Ishaan Shaikh
231070063
SY CE

Q. Write an algorithm to find gross and net salary of employees.
ABC co. ltd. has 2000 employees. Your task is to calculate each employee's salary and find employees with minimum salary and maximum salary.

Algorithm:

minMaxSalary(employees):
// Input: Array of objects of employee
// Output: Minimum and maximum salary

min_salary = INT_MAX
max_salary = INT_MIN

for each employee in employees:
    net_salary = employee.calculate_net_salary()

    if net_salary < min_salary:
        min_salary = net_salary

    if net_salary > max_salary:
        max_salary = net_salary

return (min_salary, max_salary)

```
minMaxSalary(employees):
//Input: Objects of employee class as an array
//Output: Minimum and Maximum salary

if employees.length == 1:
    emp = employees[0]
    net_salary = emp.calculate_net_salary()
    return net_salary, net_salary

mid = employees.length / 2
left = employees[:mid]
right = employees[mid:]

(left_min, left_max) = minMaxSalary(left)
(right_min, right_max) = minMaxSalary(right)

overall_min = min(left_min, right_min)
overall_max = max(left_max, right_max)

return(overall_min, overall_max)
```

Test cases:

Test cases:

1) Input:

employees = [Employee(basic_salary=3000, allowances=500, taxes=200, other_deductions=100),
Employee(basic_salary=3200, allowances=600, taxes=250, other_deductions=120),
Employee(basic_salary=2900, allowances=450, taxes=190, other_deductions=90),
Employee(basic_salary=3100, allowances=550, taxes=210, other_deductions=110),
Employee(basic_salary=3400, allowances=700, taxes=230, other_deductions=140),
Employee(basic_salary=3300, allowances=650, taxes=220, other_deductions=130),
Employee(basic_salary=2800, allowances=400, taxes=180, other_deductions=80),
Employee(basic_salary=3500, allowances=750, taxes=240, other_deductions=150),
Employee(basic_salary=3000, allowances=500, taxes=200, other_deductions=100),
Employee(basic_salary=3100, allowances=550, taxes=210, other_deductions=110)]

Output:

Minimum salary: 2940
Maximum salary: 3860

2) Input :

employees = [Employee (basic_salary = 3500, allowances = 1000, taxes = 400, other_deductions = 200),
Employee (basic_salary = 2200, allowances = 800, taxes = 300, other_deductions = 100),
Employee (basic_salary = 4000, allowances = 1200, taxes = 500, other_deductions = 250),
Employee (basic_salary = 3200, allowances = 900, taxes = 350, other_deductions = 150),
Employee (basic_salary = 2800, allowances = 850, taxes = 320, other_deductions = 130),
Employee (basic_salary = 3600, allowances = 1100, taxes = 450, other_deductions = 200),
Employee (basic_salary = 3300, allowances = 950, taxes = 420, other_deductions = 180),
Employee (basic_salary = 3100, allowances = 870, taxes = 400, other_deductions = 170),
Employee (basic_salary = 3000, allowances = 800, taxes = 390, other_deductions = 160),
Employee (basic_salary = 3500, allowances = 1000, taxes = 420, other_deductions = 210)]

Output :

Minimum salary : 3100
Maximum salary : 4450

3) Input:

employees = [Employee(basic_salary = 5000, allowances = 2000, taxes = 1000, other_deductions = 500),
Employee(basic_salary = 4000, allowances = 1500, taxes = 800, other_deductions = 400),
Employee(basic_salary = 4500, allowances = 1800, taxes = 900, other_deductions = 450),
Employee(basic_salary = 4700, allowances = 1600, taxes = 950, other_deductions = 400),
Employee(basic_salary = 4200, allowances = 1400, taxes = 850, other_deductions = 300),
Employee(basic_salary = 4600, allowances = 1700, taxes = 920, other_deductions = 350),
Employee(basic_salary = 4300, allowances = 1550, taxes = 870, other_deductions = 320),
Employee(basic_salary = 4400, allowances = 1650, taxes = 880, other_deductions = 370),
Employee(basic_salary = 4800, allowances = 1900, taxes = 960, other_deductions = 410),
Employee(basic_salary = 4900, allowances = 2000, taxes = 970, other_deductions = 420)]

Output:

Minimum salary = 4300
Maximum salary = 5510

4) Input:

employees = [Employee (basic_salary = -3000, allowances = -500, taxes = -200, other_deductions = -100),
Employee (basic_salary = -3200, allowances = -600, taxes = -250, other_deductions = -80),
Employee (basic_salary = -2900, allowances = -450, taxes = -190, other_deductions = -70),
Employee (basic_salary = -3100, allowances = -550, taxes = -150, other_deductions = -90),
Employee (basic_salary = -3400, allowances = -500, taxes = -180, other_deductions = -80),
Employee (basic_salary = -3300, allowances = -600, taxes = -170, other_deductions = -100),
Employee (basic_salary = -2800, allowances = -100, taxes = -180, other_deductions = -70),
Employee (basic_salary = -3500, allowances = -250, taxes = -200, other_deductions = -80),
Employee (basic_salary = -3000, allowances = -450, taxes = -210, other_deductions = -90),
Employee (basic_salary = -3100, allowances = -410, taxes = -120, other_deductions = -110)]

Output:

Minimum salary : Invalid Input
Maximum salary : Invalid Input

8) Input:

employees = [Employee (basic_salary = 3000, allowances = 500, taxes = -200, other_deductions = 100),
Employee (basic_salary = -2500, allowances = 400, taxes = 150, other_deductions = -80),
Employee (basic_salary = -5000, allowances = -1000, taxes = 300, other_deductions = 200),
Employee (basic_salary = 4000, allowances = 900, taxes = 400, other_deductions = -180),
Employee (basic_salary = 4500, allowances = 800, taxes = -450, other_deductions = -180),
Employee (basic_salary = 3000, allowances = 300, taxes = 200, other_deductions = 100),
Employee (basic_salary = -3100, allowance = -550, taxes = 210, other_deductions = -210),
Employee (basic_salary = 3300, allowances = -650, taxes = -220, other_deductions = 110),
Employee (basic_salary = -2800, allowances = 400, taxes = -180, other_deductions = 130),
Employee (basic_salary = 3500, allowances = 750, taxes = -240, other_deductions = 80)]

Output:

Minimum salary: Invalid input
Maximum salary: Invalid input

6) Input:

employees = [Employee(basic_salary = 0, allowances = 0, taxes = 0, other_deductions = 0),
Employee(basic_salary = 1000, allowances = 200, taxes = -50, other_deductions = -20),
Employee(basic_salary = 0, allowances = -100, taxes = -10, other_deductions = -5),
Employee(basic_salary = 2000, allowances = 0, taxes = -100, other_deductions = -30),
Employee(basic_salary = 500, allowances = 0, taxes = 0, other_deductions = -50),
Employee(basic_salary = -1500, allowances = -250, taxes = -70, other_deductions = 40),
Employee(basic_salary = 0, allowances = -300, taxes = -20, other_deductions = 0),
Employee(basic_salary = -2500, allowances = 0, taxes = -200, other_deductions = 100),
Employee(basic_salary = 1000, allowances = -500, taxes = -60, other_deductions = -30),
Employee(basic_salary = 0, allowances = 0, taxes = 0, other_deductions = -10)]

Output:

Minimum salary: Invalid input
Maximum salary: Invalid input

Time Complexity:

Time complexity:

Recursive:

Input size : n

Basic operation : Compare salaries of employees and return maximum and minimum salary

Let $T(N)$ be the time required by the algorithm to run

$$T(N) = 2T\left(\frac{N}{2}\right) + O(1)$$

Using master's theorem

$$T(N) = aT\left(\frac{N}{b}\right) + f(n)$$

$a = 2, \; b = 2, \; f(n) = O(1)$

$f(n) = O(n^d)$

$\therefore n^d = 1$

$\therefore d = 0$

$$\therefore T.C = O(n^{\log_b a}) = O(n^{\log_2 2}) = O(n^1)$$

$$\therefore \boxed{T.C = O(n)}$$

Iterative:

Input size : n

Basic operation: Compare net salary of employees and return the minimum and maximum salary.

$$T.C = \sum_{i=0}^{n-1} 1$$ [Loop runs for each employees and there are n employees]

$$= [n - 1 + 1]$$

$$= n$$

$$\therefore \boxed{T.C = O(n)}$$

Program: Google Coding style for python is used

```
class Employee:
    """Represents an employee with salary details."""

    def __init__(self, basic_salary, allowances, taxes, other_deductions):
        """Initializes an employee with basic salary, allowances, taxes,
and other deductions.

        Args:
            basic_salary (float): The basic salary of the employee.
            allowances (float): The allowances of the employee.
            taxes (float): The taxes applicable to the employee.
            other_deductions (float): Any other deductions from the
employee's salary.
        """
        self.basic_salary = basic_salary
        self.allowances = allowances
        self.taxes = taxes
        self.other_deductions = other_deductions

    def calculate_gross_salary(self):
        """Calculates the gross salary.

        Returns:
            float: The gross salary, which is the sum of basic salary and
allowances.
        """
        return self.basic_salary + self.allowances

    def calculate_net_salary(self):
        """Calculates the net salary after taxes and other deductions.

        Returns:
            float: The net salary.
        """
        gross_salary = self.calculate_gross_salary()
        net_salary = gross_salary - self.taxes - self.other_deductions
```

```
        return net_salary

    def __str__(self):
        return (f"Basic Salary: {self.basic_salary}, Allowances:
{self.allowances}, "
                f"Taxes: {self.taxes}, Other Deductions:
{self.other_deductions}, "
                f"Gross Salary: {self.calculate_gross_salary()}, "
                f"Net Salary: {self.calculate_net_salary()}")


class SalaryProcessor:
    """Processes a list of employees to calculate minimum and maximum net
salaries."""

    def __init__(self, employees):
        """Initializes the salary processor with a list of employees.

        Args:
            employees (list of Employee): A list of Employee objects.
        """
        self.employees = employees

    def calculate_salaries(self):
        """Calculates the minimum and maximum net salaries.

        Returns:
            tuple: A tuple containing the minimum and maximum net salaries
or
                a message if negative salaries are detected.
        """
        if not self.employees:
            return None, None

        min_salary, max_salary = self._divide_and_conquer(self.employees)

        if min_salary < 0 or max_salary < 0:
            return ("Negative Net Salary Detected", "Negative Net Salary
Detected")
```

```
        return min_salary, max_salary

    def _divide_and_conquer(self, employee_list):
        """Recursively divides and conquers to find salary statistics.

        Args:
            employee_list (list of Employee): A list of Employee objects.

        Returns:
            tuple: The minimum and maximum net salaries.
        """
        # Base case
        if len(employee_list) == 1:
            emp = employee_list[0]
            net_salary = emp.calculate_net_salary()
            return net_salary, net_salary

        # Divide
        mid = len(employee_list) // 2
        left_half = employee_list[:mid]
        right_half = employee_list[mid:]

        # Conquer
        left_min, left_max = self._divide_and_conquer(left_half)
        right_min, right_max = self._divide_and_conquer(right_half)

        # Combine
        overall_min = min(left_min, right_min)
        overall_max = max(left_max, right_max)

        return overall_min, overall_max


def run_test_case(employees, test_case_number):
    """Runs a test case and prints the minimum and maximum salaries.

    Args:
        employees (list of Employee): A list of Employee objects.
        test_case_number (int): The test case number to identify the
output.
```

```
    """
    processor = SalaryProcessor(employees)
    min_salary, max_salary = processor.calculate_salaries()
    print(f"Test Case {test_case_number}:")
    print(f"Minimum Salary: {min_salary}")
    print(f"Maximum Salary: {max_salary}")


# Test Case 1: All Employees with Positive Salaries
employees_test_1 = [
    Employee(basic_salary=3000, allowances=500, taxes=200,
other_deductions=100),
    Employee(basic_salary=3200, allowances=600, taxes=250,
other_deductions=120),
    Employee(basic_salary=2900, allowances=450, taxes=190,
other_deductions=90),
    Employee(basic_salary=3100, allowances=550, taxes=210,
other_deductions=110),
    Employee(basic_salary=3400, allowances=700, taxes=230,
other_deductions=140),
    Employee(basic_salary=3300, allowances=650, taxes=220,
other_deductions=130),
    Employee(basic_salary=2800, allowances=400, taxes=180,
other_deductions=80),
    Employee(basic_salary=3500, allowances=750, taxes=240,
other_deductions=150),
    Employee(basic_salary=3000, allowances=500, taxes=200,
other_deductions=100),
    Employee(basic_salary=3100, allowances=550, taxes=210,
other_deductions=110),
]

run_test_case(employees_test_1, 1)

# Test Case 2: Varied Allowances and Deductions
employees_test_2 = [
    Employee(basic_salary=3500, allowances=1000, taxes=400,
other_deductions=200),
    Employee(basic_salary=2700, allowances=800, taxes=300,
other_deductions=100),
```

```
    Employee(basic_salary=4000, allowances=1200, taxes=500,
other_deductions=250),
    Employee(basic_salary=3200, allowances=900, taxes=350,
other_deductions=150),
    Employee(basic_salary=2800, allowances=850, taxes=320,
other_deductions=130),
    Employee(basic_salary=3600, allowances=1100, taxes=450,
other_deductions=200),
    Employee(basic_salary=3300, allowances=950, taxes=420,
other_deductions=180),
    Employee(basic_salary=3100, allowances=870, taxes=400,
other_deductions=170),
    Employee(basic_salary=3000, allowances=800, taxes=390,
other_deductions=160),
    Employee(basic_salary=3500, allowances=1000, taxes=420,
other_deductions=210),
]

run_test_case(employees_test_2, 2)

# Test Case 3: Employees with High Allowances and Deductions
employees_test_3 = [
    Employee(basic_salary=5000, allowances=2000, taxes=1000,
other_deductions=500),
    Employee(basic_salary=4000, allowances=1500, taxes=800,
other_deductions=400),
    Employee(basic_salary=4500, allowances=1800, taxes=900,
other_deductions=450),
    Employee(basic_salary=4700, allowances=1600, taxes=950,
other_deductions=400),
    Employee(basic_salary=4200, allowances=1400, taxes=850,
other_deductions=300),
    Employee(basic_salary=4600, allowances=1700, taxes=920,
other_deductions=350),
    Employee(basic_salary=4300, allowances=1550, taxes=870,
other_deductions=320),
    Employee(basic_salary=4400, allowances=1650, taxes=880,
other_deductions=370),
    Employee(basic_salary=4800, allowances=1900, taxes=960,
other_deductions=410),
```

```
    Employee(basic_salary=4900, allowances=2000, taxes=970,
other_deductions=420),
]

run_test_case(employees_test_3, 3)

# Test Case 4: All Negative Salaries
employees_test_4 = [
    Employee(basic_salary=-3000, allowances=-500, taxes=-200,
other_deductions=-100),
    Employee(basic_salary=-3200, allowances=-600, taxes=-250,
other_deductions=-120),
    Employee(basic_salary=-2900, allowances=-450, taxes=-190,
other_deductions=-90),
    Employee(basic_salary=-3100, allowances=-550, taxes=-210,
other_deductions=-110),
    Employee(basic_salary=-3400, allowances=-700, taxes=-230,
other_deductions=-140),
    Employee(basic_salary=-3300, allowances=-650, taxes=-220,
other_deductions=-130),
    Employee(basic_salary=-2800, allowances=-400, taxes=-180,
other_deductions=-80),
    Employee(basic_salary=-3500, allowances=-750, taxes=-240,
other_deductions=-150),
    Employee(basic_salary=-3000, allowances=-500, taxes=-200,
other_deductions=-100),
    Employee(basic_salary=-3100, allowances=-550, taxes=-210,
other_deductions=-110),
]

run_test_case(employees_test_4, 4)

# Test Case 5: Mixed Positive and Negative Values
employees_test_5 = [
    Employee(basic_salary=3000, allowances=500, taxes=-200,
other_deductions=100),
    Employee(basic_salary=-2500, allowances=400, taxes=150,
other_deductions=-80),
    Employee(basic_salary=5000, allowances=-1000, taxes=500,
other_deductions=200),
```

```
    Employee(basic_salary=4000, allowances=800, taxes=-400,
other_deductions=150),
    Employee(basic_salary=-4500, allowances=900, taxes=450,
other_deductions=-180),
    Employee(basic_salary=3000, allowances=-500, taxes=200,
other_deductions=-100),
    Employee(basic_salary=-3100, allowances=550, taxes=-210,
other_deductions=110),
    Employee(basic_salary=3300, allowances=-650, taxes=220,
other_deductions=-130),
    Employee(basic_salary=-2800, allowances=400, taxes=-180,
other_deductions=80),
    Employee(basic_salary=3500, allowances=-750, taxes=-240,
other_deductions=150),
]

run_test_case(employees_test_5, 5)

# Test Case 6: Zero and Negative Values
employees_test_6 = [
    Employee(basic_salary=0, allowances=0, taxes=0, other_deductions=0),
    Employee(basic_salary=-1000, allowances=-200, taxes=-50,
other_deductions=-20),
    Employee(basic_salary=0, allowances=-100, taxes=-10,
other_deductions=-5),
    Employee(basic_salary=-2000, allowances=0, taxes=-100,
other_deductions=-30),
    Employee(basic_salary=500, allowances=0, taxes=0,
other_deductions=-50),
    Employee(basic_salary=-1500, allowances=-250, taxes=-70,
other_deductions=-40),
    Employee(basic_salary=0, allowances=-300, taxes=-20,
other_deductions=0),
    Employee(basic_salary=-2500, allowances=0, taxes=-200,
other_deductions=-100),
    Employee(basic_salary=1000, allowances=-500, taxes=-60,
other_deductions=-30),
    Employee(basic_salary=0, allowances=0, taxes=0, other_deductions=-10),
]
```

```
run_test_case(employees_test_6, 6)
```

Output:

```
PS C:\Users\Ishaan\Desktop\ok> & C:/
Test Case 1:
Minimum Salary: 2940
Maximum Salary: 3860
Test Case 2:
Minimum Salary: 3100
Maximum Salary: 4450
Test Case 3:
Minimum Salary: 4300
Maximum Salary: 5510
Test Case 4:
Minimum Salary: Invalid input
Maximum Salary: Invalid input
Test Case 5:
Minimum Salary: Invalid input
Maximum Salary: Invalid input
Test Case 6:
Minimum Salary: Invalid input
Maximum Salary: Invalid input
```

Conclusion: Hence, in this practical, we have studied how visual studio code is used, what IDEs are and how to make coding more efficient using IDE. We have also learned divide and conquer techniques. It is to be noted that the divide and conquer method gives an efficient algorithm whereas iterative algorithms for the same problem may have larger time complexity in many cases.