Name: Bhakti Anil Baraf
Reg No: 241071907
BTech SY CE (Batch D)

Assignment 3

Aim: Understanding problem- solving using the divide and conquer algorithm. Using coupling and cohesion to improve code.

Steps Completed:

1. **Installed Visual Studio Code**:
   Downloaded from the official website and installed successfully on my system.
2. **Watched Introductory Videos**:
   - **Getting Started**:

     Understood the basics of the VS Code interface, including the Explorer, Search, Source Control, and Extensions tabs.

   - **Code Editing**:

     Explored features like IntelliSense, syntax highlighting, and auto-formatting, which enhanced the ease of writing code.

   - **Productivity Tips**:

     Learned key shortcuts, how to organize multiple projects in a workspace, and efficient debugging techniques.

   - **Extensions**:

     Installed recommended extensions, including:

     - **Python**: For writing and running Python scripts.
     - **Prettier**: To maintain consistent code formatting.
     - **GitLens**: To improve Git integration within the editor.
3. **Used VS Code for Programs**:
   Migrated existing code to VS Code and tested its debugging and code execution features. Customized the editor with themes and settings to improve the coding experience.

Divide and Conquer:

Divide and conquer is a strategy that breaks a problem into smaller, more manageable sub-problems, solves each sub-problem individually, and then combines the solutions to solve the original problem. It follows three main steps:

1. **Divide**: Split the problem into smaller sub-problems.
2. **Conquer**: Solve the sub-problems recursively.
3. **Combine**: Merge the results of the sub-problems to obtain the final solution.

Examples of divide and conquer algorithms include:

- **Merge Sort**
- **Quick Sort**
- **Binary Search**

Coupling and Cohesion:

- **Cohesion** refers to how closely related the tasks performed by a single module are. High cohesion is desirable, as it makes the module easier to understand and maintain.
- **High Cohesion**: Ensures that each module or function performs a single, well-defined task. This improves readability, reusability, and testability.
- **Coupling** refers to the degree of dependency between modules. Low coupling is preferred as it allows independent module development and reduces the impact of changes.
- **Low Coupling**: Ensures that modules are independent of each other, reducing the impact of changes in one module on others. This results in more flexible and maintainable code.

By minimizing coupling and maximizing cohesion, code becomes more modular, easier to test, and more maintainable.

Example:

Implement the **binary search** algorithm to find the position of a target element in a sorted array. The binary search algorithm works by repeatedly dividing the search interval in half. If the value of the target element is less than the middle element, the search continues on the left half, otherwise on the right half.

Binary Search Algorithm Steps:

1. **Divide**: Start with the entire array and calculate the middle index.

2. **Conquer**:
    - If the target value matches the middle element, return its index.
    - If the target value is less than the middle element, narrow the search to the left half.
    - If the target value is greater than the middle element, narrow the search to the right half.
3. **Combine**: The solution is found when the target is located at the middle index or when the search interval is reduced to zero (element not found).

Cohesion

Cohesion refers to how closely the tasks performed by a single module (or function, class, or component) are related to each other. In other words, it measures how well the elements inside a module belong together.

- **High Cohesion**: A module performs one well-defined task, making it easier to maintain, test, and understand.
- **Low Cohesion**: A module handles unrelated tasks, making it harder to manage.

Example: A function that only handles user authentication has high cohesion.

Coupling

Coupling refers to the degree of interdependence between different modules (or functions, classes, or components). It measures how much one module relies on the internal details of another module.

- **Low Coupling**: Modules are independent, making the system flexible and easy to modify.
- **High Coupling**: Modules are heavily dependent on each other, making changes risky and complex.

Example: Two independent functions with minimal interaction have low coupling.

Problem Statement

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.

Do the above task using divide and conquer technique.

Find the improvement in the complexity using divide and conquer method.

The salary calculation includes:

- Gross Salary: Basic salary + Allowances
- Net Salary: Gross Salary - Deductions (like tax, provident fund, etc.)

Explanation of Allowances and Deductions

In the context of salary calculation:

- **Allowances**: These are additional amounts paid to employees on top of their basic salary, usually to cover specific needs or as incentives. Common types of allowances include:
    - **House Rent Allowance (HRA)**: Provided to cover housing rent.
    - **Medical Allowance**: Offered for healthcare-related expenses.
    - **Transport Allowance**: To cover commuting or travel expenses.
    - **Special Allowance**: A generic allowance, which might be a performance-based or cost-of-living increment.

- **Deductions**: These are amounts subtracted from an employee's gross salary. Common deductions include:
    - **Tax Deduction**: Income tax based on the employee's salary.
    - **Provident Fund (PF)**: A retirement savings contribution deducted from salary.
    - **Insurance**: Premiums deducted for health, life, or other insurance policies.
    - **Loan Repayments**: If the employee has taken a loan from the company or a related institution, loan repayments may be deducted.

Steps:

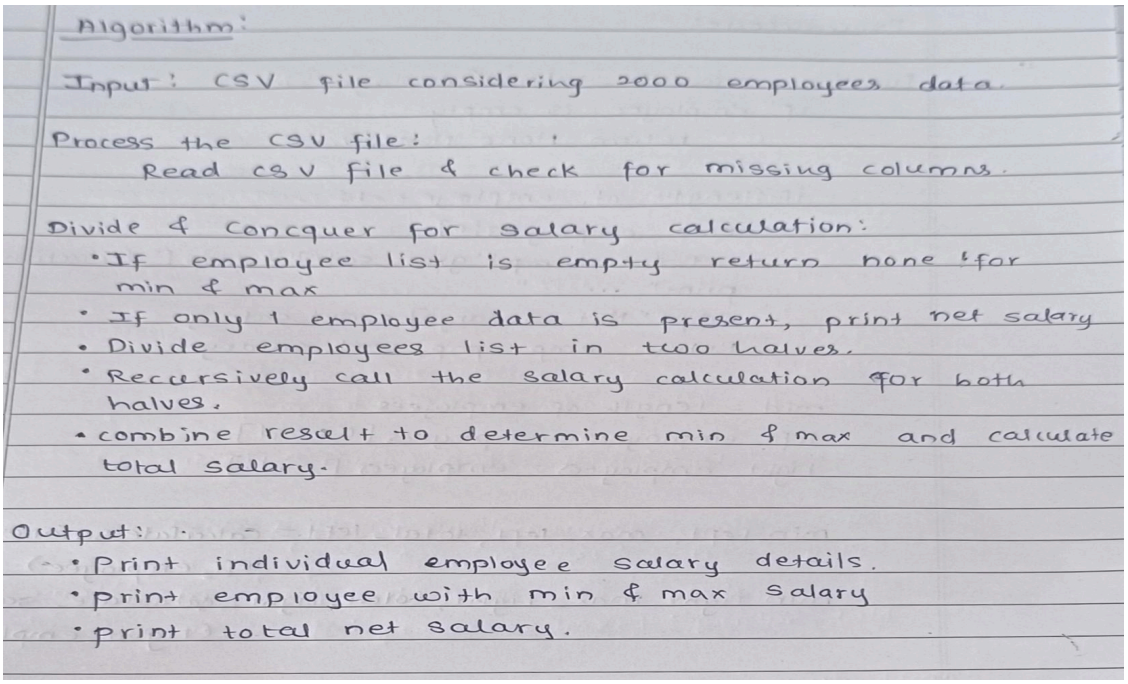
1. **Base Case**:
    - If there is only one employee, calculate their gross and net salary.
    - Return the net salary and the employee's details.
2. **Recursive Case**:
    - Split the list of employees into two halves.
    - Recursively apply the same calculation for both halves.

- After calculating for both halves, compare to find the minimum and maximum salaries.

Algorithm (Divide and Conquer):

Input: List of 2000 employees with their respective basic salary, allowances, and deductions in CSV file.

Output: Gross and net salary for each employee, and the employee with the minimum and maximum net salary.



Pseudocode:

DATE:

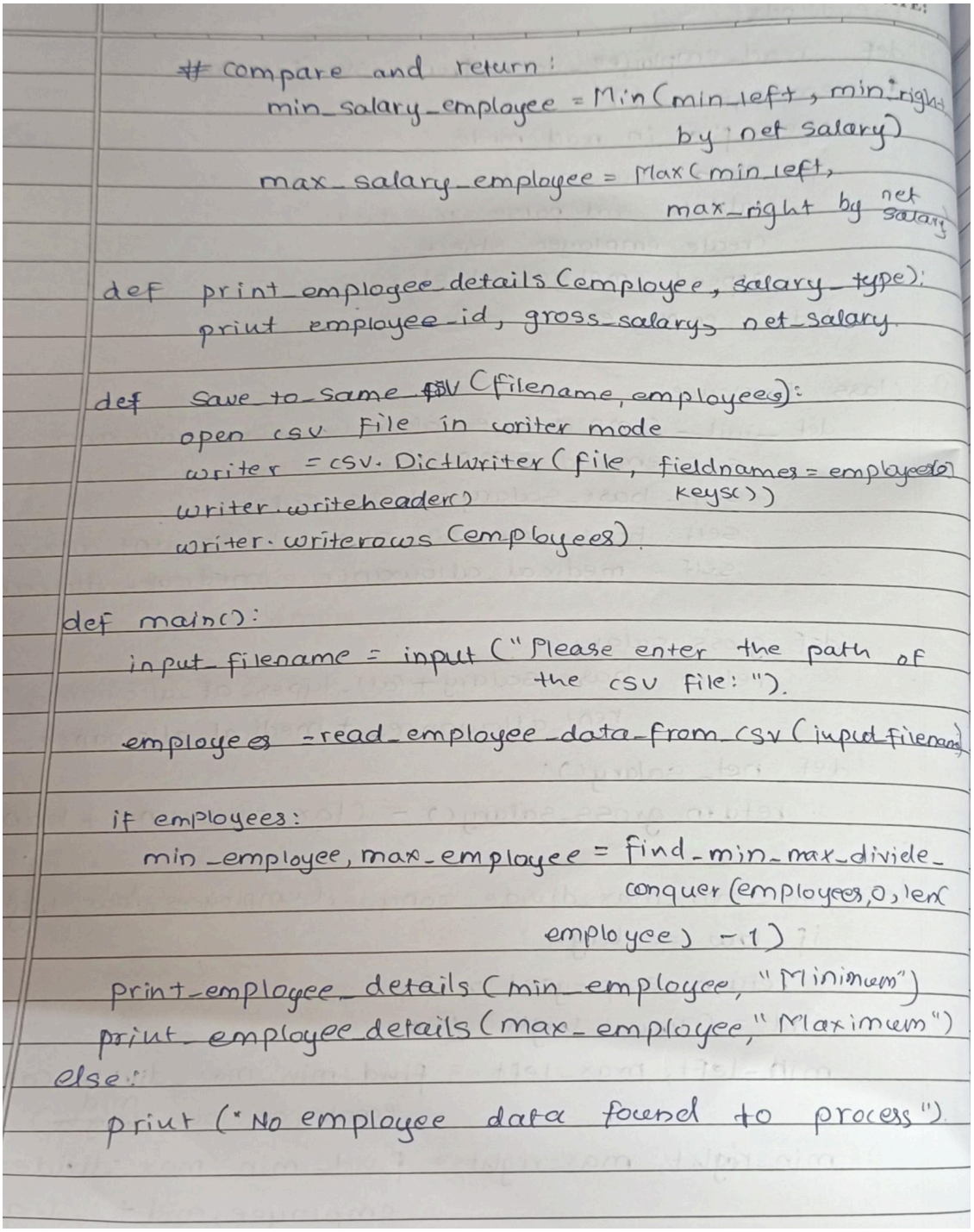
```
def read_employee_data_from_csv (filename):
    employees = []
    open. file in read mode
    for each row in file:
        validate and parse row data
        create employee object
        append employee details. to employee list.
    return employees.

class Employee:
    def __init__ (emp_id, base_salary, house_rent_allowance, ....)
        self. emp_id = emp_id
        self. base_salary = base_salary
        self. house_rent_allowance = house_rent_allowance.
        self. medical_allowance = medical_allowance.

    def gross_salary ():
        return base_salary + (all_types_of_allowance) house_
                rent_allowance + medical_allowance, ...

    def net_salary():
        return gross_salary() - (tax + insurance + loan)

def find_min_max_divide_conquer(employees, low, high):
    if low == high :
        return employee(low),
    mid = (low + high) // 2
    min_left, max_left = find_min_max_divide_conquer (
                            employees, low, mid)
    min_right, max_right = find_min_max_divide_conquer(
                            employee, mid + 1, high)
```



Python Program (Using Divide and Conquer)

```
import csv

# Employee class to store employee data and calculate salaries

class Employee:
```

```
    def __init__(self, emp_id, base_salary, house_rent_allowance,
medical_allowance, transport_allowance, tax, insurance,
loan_repayment):

        self.emp_id = emp_id

        # Ensure all salary components are non-negative

        self.base_salary = max(0, base_salary)

        self.house_rent_allowance = max(0, house_rent_allowance)

        self.medical_allowance = max(0, medical_allowance)

        self.transport_allowance = max(0, transport_allowance)

        self.tax = max(0, tax)

        self.insurance = max(0, insurance)

        self.loan_repayment = max(0, loan_repayment)



    @property

    def gross_salary(self):

        return (self.base_salary + self.house_rent_allowance +

                self.medical_allowance + self.transport_allowance)



    @property

    def net_salary(self):

        total_deductions = self.tax + self.insurance +
self.loan_repayment

        return self.gross_salary - total_deductions



def read_employee_data_from_csv(filename):

    employees = []
```

```
   with open(filename, mode='r') as file:
       reader = csv.DictReader(file)
       for row_num, row in enumerate(reader, start=1):
           try:
               # Create Employee instance and validate for non-negative
values
               employee = Employee(
                   emp_id=row['Employee ID'],
                   base_salary=float(row['Base Salary']),
                   house_rent_allowance=float(row['House Rent
Allowance']),
                   medical_allowance=float(row['Medical Allowance']),
                   transport_allowance=float(row['Transport
Allowance']),
                   tax=float(row['Tax']),
                   insurance=float(row['Insurance']),
                   loan_repayment=float(row['Loan Repayment'])
               )
               employees.append({
                   'Employee ID': employee.emp_id,
                   'Base Salary': employee.base_salary,
                   'House Rent Allowance':
employee.house_rent_allowance,
                   'Medical Allowance': employee.medical_allowance,
                   'Transport Allowance': employee.transport_allowance,
                   'Tax': employee.tax,
                   'Insurance': employee.insurance,
```

```
                    'Loan Repayment': employee.loan_repayment,
                    'Gross Salary': employee.gross_salary,
                    'Net Salary': employee.net_salary,
                })
            except ValueError as e:
                print(f"Invalid data format detected in row {row_num}: {row}. Error: {e}")
    return employees


# Divide and conquer function to find min/max salaries
def find_min_max_divide_conquer(employees, low, high):
    # Base case: when only one employee is left, return them as both min and max
    if low == high:
        return employees[low], employees[low]

    # Divide: split the array into two halves
    mid = (low + high) // 2
    min_left, max_left = find_min_max_divide_conquer(employees, low, mid)
    min_right, max_right = find_min_max_divide_conquer(employees, mid + 1, high)

    # Conquer: compare and return the overall min and max
    min_salary_employee = min(min_left, min_right, key=lambda x: x['Net Salary'])
```

```
    max_salary_employee = max(max_left, max_right, key=lambda x: x['Net Salary'])

    return min_salary_employee, max_salary_employee

def print_employee_details(employee, salary_type):
    print(f"\nEmployee with {salary_type} Salary:")
    print({
        "Employee ID": employee['Employee ID'],
        "Gross Salary": employee['Gross Salary'],
        "Net Salary": employee['Net Salary']
    })

def save_to_same_csv(filename, employees):
    # Overwrite the same file with the updated employee data
    with open(filename, mode='w', newline='') as file:
        writer = csv.DictWriter(file, fieldnames=employees[0].keys())
        writer.writeheader()
        writer.writerows(employees)

def main():
    # Prompt user to input the CSV file path
    input_filename = input("Please enter the path of the CSV file: ")

    # Read employee data from the provided CSV file
```

```
    employees = read_employee_data_from_csv(input_filename)



    if employees:

        # Use divide and conquer to find min and max salary employee

        min_employee, max_employee =
find_min_max_divide_conquer(employees, 0, len(employees) - 1)



        # Print min/max salary employee with detailed info

        print_employee_details(min_employee, "Minimum")

        print_employee_details(max_employee, "Maximum")



        # Save changes to the same CSV  file

        save_to_same_csv(input_filename, employees)

        print(f"Changes have been saved to '{input_filename}'.")

    else:

        print("No employee data found to process.")



if __name__ == "__main__":

    main()
```

Output:

```
PS C:\Users\Bhakti\Documents\Python_files>
python -u "c:\Users\Bhakti\Documents\Python_files\daa_lab3.py"
Please enter the path of the CSV file: employee_data_2000.csv

Employee with Minimum Salary:
{'Employee ID': 'E1837', 'Gross Salary': 26201.33, 'Net Salary': 15855.770000000002}

Employee with Maximum Salary:
{'Employee ID': 'E1234', 'Gross Salary': 148586.11000000002, 'Net Salary': 140901.39}
Changes have been saved to 'employee_data_2000.csv'.
PS C:\Users\Bhakti\Documents\Python_files>
```

Positive TestCase:

```
PS C:\Users\Bhakti\Documents\Python_files> python -u "c:\Users\Bhakti\Documents\Python_files\daa_lab3.py"
Please enter the path of the CSV file: sample_employee_data.csv

Employee with Minimum Salary:
{'Employee ID': 'E003', 'Gross Salary': 61500.0, 'Net Salary': 50000.0}

Employee with Maximum Salary:
{'Employee ID': 'E004', 'Gross Salary': 97000.0, 'Net Salary': 76000.0}
Changes have been saved to 'sample_employee_data.csv'.
PS C:\Users\Bhakti\Documents\Python_files>
```

Negative Test Case:

```
PS C:\Users\Bhakti\Documents\Python_files> python -u "c:\Users\Bhakti\Documents\Python_
Please enter the path of the CSV file: sample_employee_data.csv
No employee data found to process.
PS C:\Users\Bhakti\Documents\Python_files>
```

```
PS C:\Users\Bhakti\Documents\Python_files> python -u "c:\Users\Bhakti\Documents\Python_files\daa_lab3.py"
Please enter the path of the CSV file: employee_data_2000.csv
Invalid data format detected in row 1: {'Employee ID': 'E0001', 'Base Salary': 'ABC', 'House Rent Allowance':
Allowance': '3903.31', 'Tax': '15343.2', 'Insurance': '1152.05', 'Loan Repayment': '2921.58', 'Gross Salary':
could not convert string to float: 'ABC'

Employee with Minimum Salary:
{'Employee ID': 'E1837', 'Gross Salary': 26201.33, 'Net Salary': 15855.770000000002}

Employee with Maximum Salary:
{'Employee ID': 'E1234', 'Gross Salary': 148586.11000000002, 'Net Salary': 140901.39}
Changes have been saved to 'employee_data_2000.csv'.
```

Time Complexity Analysis (Divide and Conquer Approach):

DATE:

1) Time Complexity of Linear Search:

Base case: if there is only one ~~elem~~ employee/element.

$= O(1)$

Average case: if there are 2 elements

$= O(1)$

Worst case:

if there will be n comparision.

$C(n) = \sum_{i=1}^{n} 1 = n - 1 + 1 = n \in O(n)$

∴ The time complexity is $O(n)$

2) Time Complexity for Divide and Conquer.

i) • Input size = n employees ~~from~~ data.

ii) Find min & max salaries using Divide & conquer:

• Best case: if there is only one employee.

$F(n) = O(1)$

• Recursive case: if there are multiple employees.
- split the list into 2 halves.

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

iii) Solving the Recurrence:

By using master's Theorem.

$a = 2$ → two subproblems
$b = 2$ → problem size reduction factor.

$\log_b a = \log_2 2 = 1$

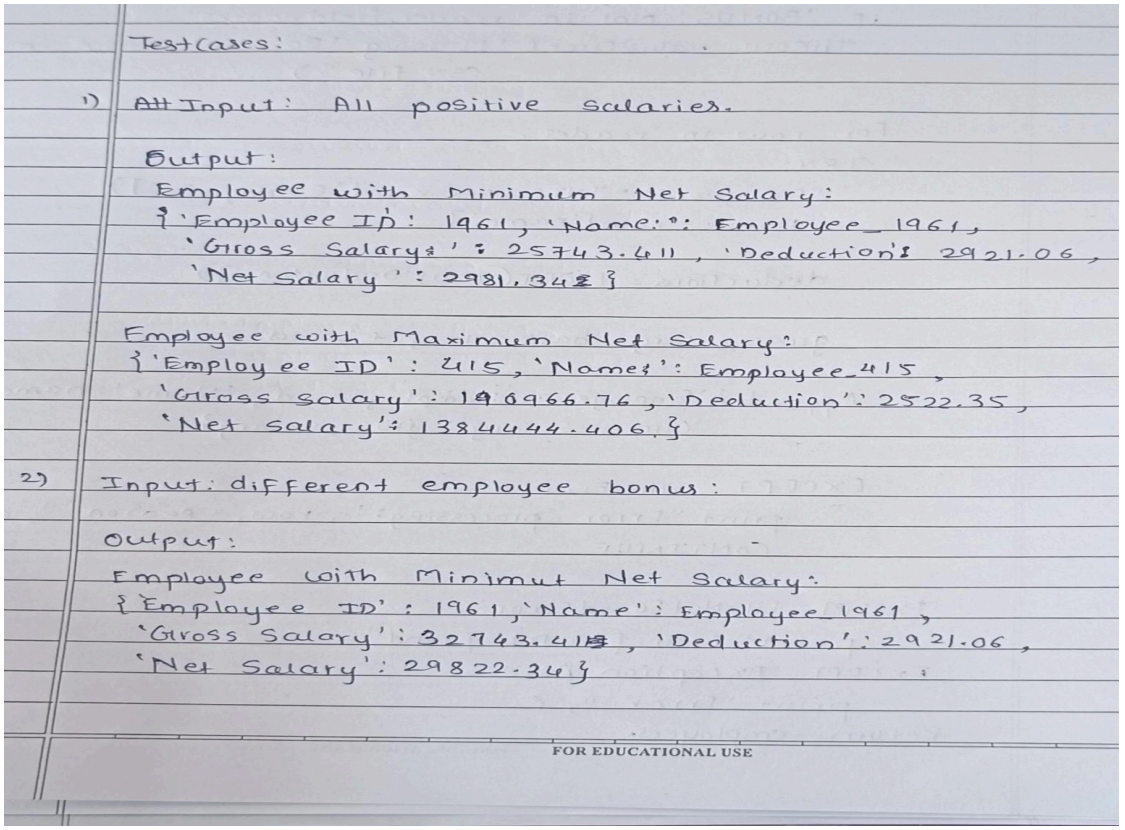
$F(n) = O(1)$ → constant time to combine results.

∵ $O(1) < \log_b(a)$

∴ Time complexity $= O(n \log n^{\log_b(a)})$
$= O(n^1)$
$= O(n)$

∴ Total time complexity $= O(n)$

Test Cases:



Test Cases:

1) All Input: All positive salaries.

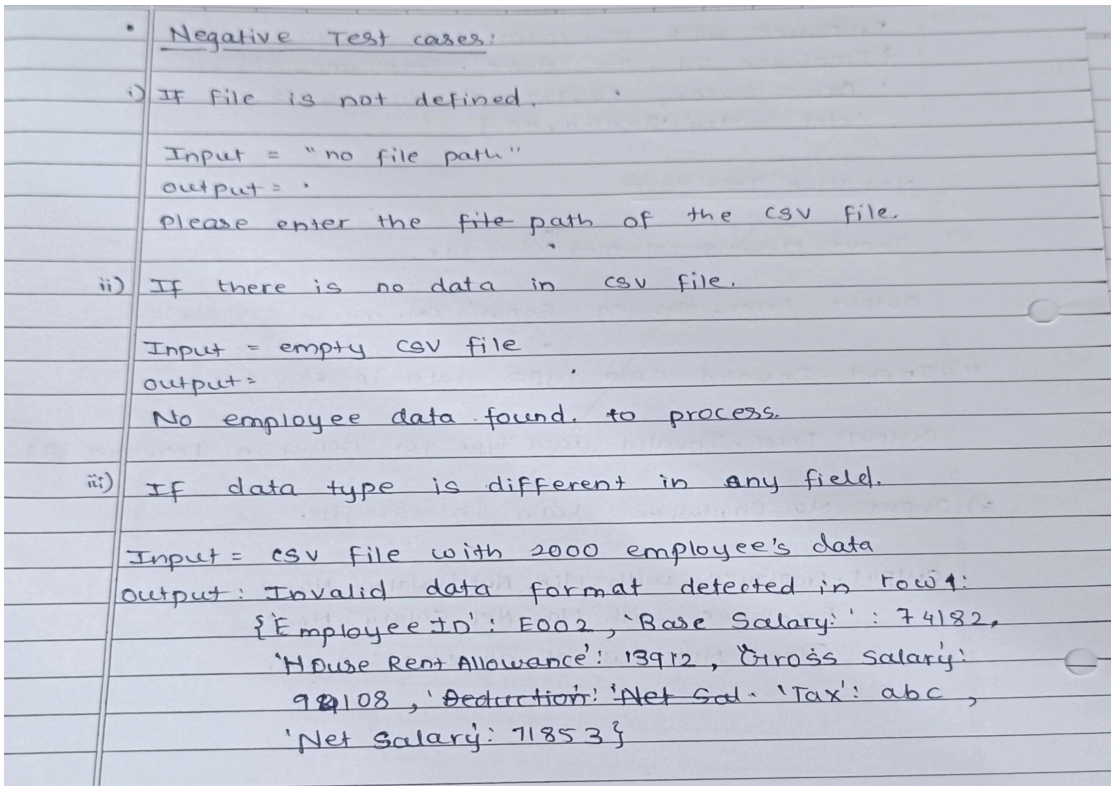
Output:
Employee with Minimum Net Salary:
{'Employee ID': 1961, 'Name:': Employee_1961,
'Gross Salary:': 25743.411, 'Deduction': 2921.06,
'Net Salary': 2981.342}

Employee with Maximum Net Salary:
{'Employee ID': 415, 'Name:': Employee_415,
'Gross salary': 190966.76, 'Deduction': 2522.35,
'Net salary': 1384444.406.}

2) Input: different employee bonus:

Output:
Employee with Minimut Net Salary:
{'Employee ID': 1961, 'Name': Employee_1961,
'Gross Salary': 32743.41, 'Deduction': 2921.06,
'Net Salary': 29822.34}

FOR EDUCATIONAL USE



• Negative Test cases:

i) If file is not defined.

Input = "no file path"
output =
Please enter the file path of the csv file.

ii) If there is no data in csv file.

Input = empty csv file
output:
No employee data found. to process.

iii) If data type is different in any field.

Input = csv file with 2000 employee's data
Output: Invalid data format detected in row 1:
{'Employee ID': E002, 'Base Salary:': 74182,
'House Rent Allowance': 13912, 'Gross Salary':
98108, 'Deduction': 'Net Sal. 'Tax': abc,
'Net Salary': 71853}

Conclusion:
I learned how to implement the Divide and conquer algorithm and learned about coupling and cohesion. I successfully implemented solution for given problem statement using divide and conquer algorithm.