NAME : PREM SAH
PRN : 123B1B234
DIV : D

ASSIGNMENT NO.2

Consider Employee database of PCCOE (at least 20 records). Database contains different fields of every employee like EMP-ID, EMP-Name and EMP-Salary.

a. Arrange list of employees according to EMP-ID in ascending order using Quick Sort
b. Arrange list of Employee alphabetically using Merge Sort

Code a)

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Structure to hold employee information
struct Employee {
    int empId;
    string empName;
    double empSalary;
};

// Function to swap two Employee records
void swap(Employee& a, Employee& b) {
    Employee temp = a;
    a = b;
    b = temp;
}

// Partition function for Quick Sort
int partition(vector<Employee>& employees, int low, int high) {
    int pivot = employees[high].empId;  // Choose the last element as pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j < high; j++) {
        if (employees[j].empId < pivot) {
            i++;
            swap(employees[i], employees[j]);  // Swap if empId is less than pivot
        }
    }
    swap(employees[i + 1], employees[high]);  // Swap pivot to the correct position
    return (i + 1);  // Return the partition index
}

// Quick Sort function
void quickSort(vector<Employee>& employees, int low, int high) {
    if (low < high) {
```

```
        int pi = partition(employees, low, high);  // Partitioning index
        quickSort(employees, low, pi - 1);  // Recursively sort before partition
        quickSort(employees, pi + 1, high);  // Recursively sort after partition
    }
}

// Function to print Employee records
void printEmployees(const vector<Employee>& employees) {
    for (const auto& emp : employees) {
        cout << "EMP-ID: " << emp.empId
            << ", EMP-NAME: " << emp.empName
            << ", EMP-SALARY: " << emp.empSalary << endl;
    }
}

int main() {
    // Sample Employee records
    vector<Employee> employees = {
        {105, "John Doe", 55000.00},
        {102, "Jane Smith", 60000.00},
        {101, "Alice Brown", 50000.00},
        {104, "Bob Johnson", 45000.00},
        {106, "Chris Lee", 70000.00},
        {108, "Diana Prince", 75000.00},
        {109, "Ethan Hunt", 52000.00},
        {107, "Fiona Apple", 48000.00},

    };

    cout << "Employee Records (Before Sorting by EMP-ID):" << endl;
    printEmployees(employees);

    // Sorting Employee records by EMP-ID using Quick Sort
    quickSort(employees, 0, employees.size() - 1);

    cout << "\nEmployee Records (After Sorting by EMP-ID):" << endl;
    printEmployees(employees);

    return 0;
}
```

Output :

```
Employee Records (Before Sorting by EMP-ID):
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000
```

Employee Records (After Sorting by EMP-ID):
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Structure to hold employee information
struct Employee {
    int empId;
    string empName;
    double empSalary;
};

// Function to swap two Employee records
void swap(Employee& a, Employee& b) {
    Employee temp = a;
    a = b;
    b = temp;
}

// Partition function for Quick Sort
int partition(vector<Employee>& employees, int low, int high) {
    int pivot = employees[high].empId;  // Choose the last element as pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j < high; j++) {
        if (employees[j].empId < pivot) {
            i++;
            swap(employees[i], employees[j]);  // Swap if empId is less than pivot
        }
    }
    swap(employees[i + 1], employees[high]);  // Swap pivot to the correct position
    return (i + 1);  // Return the partition index
}

// Quick Sort function
void quickSort(vector<Employee>& employees, int low, int high) {
    if (low < high) {
        int pi = partition(employees, low, high);  // Partitioning index
        quickSort(employees, low, pi - 1);  // Recursively sort before partition
        quickSort(employees, pi + 1, high);  // Recursively sort after partition
    }
}

// Function to print Employee records
void printEmployees(const vector<Employee>& employees) {
    for (const auto& emp : employees) {
        cout << "EMP-ID: " << emp.empId
             << ", EMP-NAME: " << emp.empName
             << ", EMP-SALARY: " << emp.empSalary << endl;
```

```
    }
}

int main() {
    // Sample Employee records
    vector<Employee> employees = {
        {105, "John Doe", 55000.00},
        {102, "Jane Smith", 60000.00},
        {101, "Alice Brown", 50000.00},
        {104, "Bob Johnson", 45000.00},
        {106, "Chris Lee", 70000.00},
        {108, "Diana Prince", 75000.00},
        {109, "Ethan Hunt", 52000.00},
        {107, "Fiona Apple", 48000.00},

    };

    cout << "Employee Records (Before Sorting by EMP-ID):" << endl;
    printEmployees(employees);

    // Sorting Employee records by EMP-ID using Quick Sort
    quickSort(employees, 0, employees.size() - 1);

    cout << "\nEmployee Records (After Sorting by EMP-ID):" << endl;
    printEmployees(employees);

    return 0;
}
```

Output

```
/tmp/lZmxyYhPxu.o
Employee Records (Before Sorting by EMP-ID):
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000

Employee Records (After Sorting by EMP-ID):
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
```

Code b)

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Structure to hold employee information
struct Employee {
    int empId;
    string empName;
    double empSalary;
};

// Function to merge two halves
void merge(vector<Employee>& employees, int left, int mid, int right) {
```

```
    int n1 = mid - left + 1; // Size of the left half
    int n2 = right - mid;    // Size of the right half

    // Create temporary vectors
    vector<Employee> leftEmployees(n1);
    vector<Employee> rightEmployees(n2);

    // Copy data to temporary vectors
    for (int i = 0; i < n1; i++)
        leftEmployees[i] = employees[left + i];
    for (int j = 0; j < n2; j++)
        rightEmployees[j] = employees[mid + 1 + j];

    // Merge the temporary vectors back into employees
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftEmployees[i].empName <= rightEmployees[j].empName) {
            employees[k] = leftEmployees[i];
            i++;
        } else {
            employees[k] = rightEmployees[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftEmployees, if any
    while (i < n1) {
        employees[k] = leftEmployees[i];
        i++;
        k++;
    }

    // Copy the remaining elements of rightEmployees, if any
    while (j < n2) {
        employees[k] = rightEmployees[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(vector<Employee>& employees, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2; // Find the mid point
        mergeSort(employees, left, mid);     // Sort first half
        mergeSort(employees, mid + 1, right); // Sort second half
        merge(employees, left, mid, right);   // Merge the sorted halves
    }
}

// Function to print the Employee records
void printEmployees(const vector<Employee>& employees) {
```

```
    for (const auto& emp : employees) {
        cout << "EMP-ID: " << emp.empId
             << ", EMP-NAME: " << emp.empName
             << ", EMP-SALARY: " << emp.empSalary << endl;
    }
}

int main() {
    // Sample Employee records
    vector<Employee> employees = {
        {105, "John Doe", 55000.00},
        {102, "Jane Smith", 60000.00},
        {101, "Alice Brown", 50000.00},
        {104, "Bob Johnson", 45000.00},
        {106, "Chris Lee", 70000.00},
        {108, "Diana Prince", 75000.00},
        {109, "Ethan Hunt", 52000.00},
        {107, "Fiona Apple", 48000.00},

    };

    cout << "Employee Records (Before Sorting by EMP-NAME):" << endl;
    printEmployees(employees);

    // Sorting Employee records by EMP-NAME using Merge Sort
    mergeSort(employees, 0, employees.size() - 1);

    cout << "\nEmployee Records (After Sorting by EMP-NAME):" << endl;
    printEmployees(employees);

    return 0;
}
```

```
Output :
Employee Records (Before Sorting by EMP-NAME):
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000

Employee Records (After Sorting by EMP-NAME):
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
```

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Structure to hold employee information
struct Employee {
    int empId;
    string empName;
    double empSalary;
};

// Function to merge two halves
void merge(vector<Employee>& employees, int left, int mid, int right) {
    int n1 = mid - left + 1; // Size of the left half
    int n2 = right - mid;    // Size of the right half

    // Create temporary vectors
    vector<Employee> leftEmployees(n1);
    vector<Employee> rightEmployees(n2);

    // Copy data to temporary vectors
    for (int i = 0; i < n1; i++)
        leftEmployees[i] = employees[left + i];
    for (int j = 0; j < n2; j++)
        rightEmployees[j] = employees[mid + 1 + j];

    // Merge the temporary vectors back into employees
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftEmployees[i].empName <= rightEmployees[j].empName) {
            employees[k] = leftEmployees[i];
            i++;
        } else {
            employees[k] = rightEmployees[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftEmployees, if any
    while (i < n1) {
        employees[k] = leftEmployees[i];
        i++;
        k++;
    }
```

```cpp

    // Copy the remaining elements of rightEmployees, if any
    while (j < n2) {
        employees[k] = rightEmployees[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(vector<Employee>& employees, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2; // Find the mid point
        mergeSort(employees, left, mid);     // Sort first half
        mergeSort(employees, mid + 1, right); // Sort second half
        merge(employees, left, mid, right);   // Merge the sorted halves
    }
}

// Function to print the Employee records
void printEmployees(const vector<Employee>& employees) {
    for (const auto& emp : employees) {
        cout << "EMP-ID: " << emp.empId
             << ", EMP-NAME: " << emp.empName
             << ", EMP-SALARY: " << emp.empSalary << endl;
    }
}

int main() {
    // Sample Employee records
    vector<Employee> employees = {
        {105, "John Doe", 55000.00},
        {102, "Jane Smith", 60000.00},
        {101, "Alice Brown", 50000.00},
        {104, "Bob Johnson", 45000.00},
        {106, "Chris Lee", 70000.00},
        {108, "Diana Prince", 75000.00},
        {109, "Ethan Hunt", 52000.00},
        {107, "Fiona Apple", 48000.00},

    };

    cout << "Employee Records (Before Sorting by EMP-NAME):" << endl;
    printEmployees(employees);

    // Sorting Employee records by EMP-NAME using Merge Sort
    mergeSort(employees, 0, employees.size() - 1);

    cout << "\nEmployee Records (After Sorting by EMP-NAME):" << endl;
    printEmployees(employees);

    return 0;
}
```

Output

```
/tmp/lPlAwa1zgc.o
Employee Records (Before Sorting by EMP-NAME):
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000

Employee Records (After Sorting by EMP-NAME):
EMP-ID: 101, EMP-NAME: Alice Brown, EMP-SALARY: 50000
EMP-ID: 104, EMP-NAME: Bob Johnson, EMP-SALARY: 45000
EMP-ID: 106, EMP-NAME: Chris Lee, EMP-SALARY: 70000
EMP-ID: 108, EMP-NAME: Diana Prince, EMP-SALARY: 75000
EMP-ID: 109, EMP-NAME: Ethan Hunt, EMP-SALARY: 52000
EMP-ID: 107, EMP-NAME: Fiona Apple, EMP-SALARY: 48000
EMP-ID: 102, EMP-NAME: Jane Smith, EMP-SALARY: 60000
EMP-ID: 105, EMP-NAME: John Doe, EMP-SALARY: 55000


=== Code Execution Successful ===
```