

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.

- Arrange list of employees according to EMP-ID in ascending order using Quick Sort
- 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
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  using namespace std;
6
7  // Structure to hold employee information
8- struct Employee {
9      int empId;
10     string empName;
11     double empSalary;
12 };
13
14 // Function to swap two Employee records
15- void swap(Employee& a, Employee& b) {
16     Employee temp = a;
17     a = b;
18     b = temp;
19 }
20
21 // Partition function for Quick Sort
22- int partition(vector<Employee>& employees, int low, int high) {
23     int pivot = employees[high].empId; // Choose the last element as pivot
24     int i = (low - 1); // Index of smaller element
25
26     for (int j = low; j < high; j++) {
27         if (employees[j].empId < pivot) {
28             i++;
29             swap(employees[i], employees[j]); // Swap if empId is less than pivot
30         }
31     }
32     swap(employees[i + 1], employees[high]); // Swap pivot to the correct position
33     return (i + 1); // Return the partition index
34 }
35
36 // Quick Sort function
37- void quickSort(vector<Employee>& employees, int low, int high) {
38     if (low < high) {
39         int pi = partition(employees, low, high); // Partitioning index
40         quickSort(employees, low, pi - 1); // Recursively sort before partition
41         quickSort(employees, pi + 1, high); // Recursively sort after partition
42     }
43 }
44
45 // Function to print Employee records
46- void printEmployees(const vector<Employee>& employees) {
47     for (const auto& emp : employees) {
48         cout << "EMP-ID: " << emp.empId
49              << ", EMP-NAME: " << emp.empName
50              << ", EMP-SALARY: " << emp.empSalary << endl;
```

```

51     }
52 }
53
54 int main() {
55     // Sample Employee records
56     vector<Employee> employees = {
57         {105, "John Doe", 55000.00},
58         {102, "Jane Smith", 60000.00},
59         {101, "Alice Brown", 50000.00},
60         {104, "Bob Johnson", 45000.00},
61         {106, "Chris Lee", 70000.00},
62         {108, "Diana Prince", 75000.00},
63         {109, "Ethan Hunt", 52000.00},
64         {107, "Fiona Apple", 48000.00},
65     };
66
67     cout << "Employee Records (Before Sorting by EMP-ID):" << endl;
68     printEmployees(employees);
69
70     // Sorting Employee records by EMP-ID using Quick Sort
71     quickSort(employees, 0, employees.size() - 1);
72
73     cout << "\nEmployee Records (After Sorting by EMP-ID):" << endl;
74     printEmployees(employees);
75
76     return 0;
77 }
78
79

```

Output

```

/tmp/1ZmxyYhPxu.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

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  using namespace std;
6
7  // Structure to hold employee information
8  struct Employee {
9      int empId;
10     string empName;
11     double empSalary;
12 };
13
14 // Function to merge two halves
15 void merge(vector<Employee>& employees, int left, int mid, int right) {
16     int n1 = mid - left + 1; // Size of the left half
17     int n2 = right - mid;    // Size of the right half
18
19     // Create temporary vectors
20     vector<Employee> leftEmployees(n1);
21     vector<Employee> rightEmployees(n2);
22
23     // Copy data to temporary vectors
24     for (int i = 0; i < n1; i++)
25         leftEmployees[i] = employees[left + i];
26     for (int j = 0; j < n2; j++)
27         rightEmployees[j] = employees[mid + 1 + j];
28
29     // Merge the temporary vectors back into employees
30     int i = 0, j = 0, k = left;
31     while (i < n1 && j < n2) {
32         if (leftEmployees[i].empName <= rightEmployees[j].empName) {
33             employees[k] = leftEmployees[i];
34             i++;
35         } else {
36             employees[k] = rightEmployees[j];
37             j++;
38         }
39         k++;
40     }
41
42     // Copy the remaining elements of leftEmployees, if any
43     while (i < n1) {
44         employees[k] = leftEmployees[i];
45         i++;
46         k++;
47     }
```

```

48
49 // Copy the remaining elements of rightEmployees, if any
50 while (j < n2) {
51     employees[k] = rightEmployees[j];
52     j++;
53     k++;
54 }
55 }
56
57 // Merge Sort function
58 void mergeSort(vector<Employee>& employees, int left, int right) {
59     if (left < right) {
60         int mid = left + (right - left) / 2; // Find the mid point
61         mergeSort(employees, left, mid); // Sort first half
62         mergeSort(employees, mid + 1, right); // Sort second half
63         merge(employees, left, mid, right); // Merge the sorted halves
64     }
65 }
66
67 // Function to print the Employee records
68 void printEmployees(const vector<Employee>& employees) {
69     for (const auto& emp : employees) {
70         cout << "EMP-ID: " << emp.empId
71             << ", EMP-NAME: " << emp.empName
72             << ", EMP-SALARY: " << emp.empSalary << endl;
73     }
74 }
75
76 int main() {
77     // Sample Employee records
78     vector<Employee> employees = {
79         {105, "John Doe", 55000.00},
80         {102, "Jane Smith", 60000.00},
81         {101, "Alice Brown", 50000.00},
82         {104, "Bob Johnson", 45000.00},
83         {106, "Chris Lee", 70000.00},
84         {108, "Diana Prince", 75000.00},
85         {109, "Ethan Hunt", 52000.00},
86         {107, "Fiona Apple", 48000.00},
87     };
88
89     cout << "Employee Records (Before Sorting by EMP-NAME):" << endl;
90     printEmployees(employees);
91
92     // Sorting Employee records by EMP-NAME using Merge Sort
93     mergeSort(employees, 0, employees.size() - 1);
94
95     cout << "\nEmployee Records (After Sorting by EMP-NAME):" << endl;
96     printEmployees(employees);
97
98     return 0;
99 }
100 }

```


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 ===|
```