



PIMPRI CHINCHWAD EDUCATION TRUST'S.  
**PIMPRI CHINCHWAD COLLEGE OF ENGINEERING**  
(An Autonomous Institute)

**S.Y. B. TECH**

**Year: 2024 – 25**

**Semester: I**

**Name: Sonawane Prachi Mahendra.**

**PRN: 124B2B018**

**Department : Computer Engineering**

**Division: B**

**Course : Data Structures Laboratory**

**Course Code: BCE23PC02**

**Date: 22/08/24**

## Assignment – 2

- **Aim:**

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

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

- **Source Code:**

**1. Arrange list of employees according to EMP-ID in ascending order using Quick Sort:**

```
#include<iostream>
```

```
#include<string>
```

```
class emp
```

```
{
```

```
    public:
```

```
    int id;
```

```
    int salary; std::string
```

```
    name;
```

```
    void read(){
```

```

    std::cout<<"Enter id:";

    std::cin>>id; std::cout<<"Enter
    name:"; std::cin>>name;

    std::cout<<"Enter salary:";

    std::cin>>salary;

}
};

void quicksort(emp x[],int f,int l)
{
    int pivot,i,j; if(f<l)

    {
        pivot=f,j=l,i=f+1; while(i<=j)

        {
            while(x[i].id < x[pivot].id)

            {

                i++;

            }

            while(x[j].id > x[pivot].id){

                j--;

            }

            if(i<j)

            {

                emp temp=x[i];

                x[i]=x[j]; x[j]=temp;

```

```

    }
}
emp temp=x[pivot];
x[pivot]=x[j]; x[j]=temp;

quicksort(x,f,j-1);

quicksort(x,j+1,l);

}
}

int main(){
    emp employees[5];

    for (int i=0; i<5; ++i){

        std::cout<<"Enter details for employee:"<<(i+1)<<":\n"; employees[i].read();

    }
    quicksort(employees, 0, 4);

    std::cout << "\nSorted employees by ID:\n";

    for (int i=0; i<5; ++i){

        std::cout<<"ID: "<<employees[i].id<<",Name:
"<<employees[i].name<<",Salary:"<<employees[i].salary<<std::endl;

    }

    return 0;
}

```

## 2. Arrange list of Employee alphabetically using Merge Sort:

```
#include <iostream>

#include <string>

struct Employee
{
    int empId;
    std::string empName; float
    empSalary;
};

void merge(Employee employees[], int left, int mid, int right)
{
    int n1 = mid - left + 1; int n2 =
    right - mid;

    Employee* L = new Employee[n1]; Employee* R =
    new Employee[n2];

    for (int i = 0; i < n1; i++)
    {
        L[i] = employees[left + i];
    }

    for (int j = 0; j < n2; j++)
    {
        R[j] = employees[mid + 1 + j];
```

```

}
int i = 0, j = 0, k = left; while (i <
n1 && j < n2)
{
    if (L[i].empName <= R[j].empName)
    {
        employees[k] = L[i]; i++;
    }
    else
    {
        employees[k] = R[j]; j++;
    }
    k++;
}
while (i < n1)
{
    employees[k] = L[i]; i++;
    k++;
}
while (j < n2)
{
    employees[k] = R[j]; j++;
    k++;
}

```

```

    delete[] L; delete[] R;
}

void mergeSort(Employee employees[], int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;
        mergeSort(employees, left, mid);
        mergeSort(employees, mid + 1, right);
        merge(employees, left, mid, right);
    }
}

void printEmployees(const Employee employees[], int n)
{
    for (int i = 0; i < n; i++) {
        std::cout << "EMP-ID: " << employees[i].empId
        << ", Name: " << employees[i].empName
        << ", Salary: " << employees[i].empSalary << std::endl;
    }
}

int main()
{
    Employee employees[20] =
    {
        {101, "Alice", 50000}, {102, "Bob", 60000}, {103, "Charlie", 55000},

```

```
{104, "David", 70000}, {105, "Eve", 80000}, {106, "Frank", 75000},  
{107, "Grace", 65000}, {108, "Hannah", 72000}, {109, "Ivy", 58000},  
{110, "Jack", 54000}, {111, "Karen", 69000}, {112, "Leo", 72000},  
{113, "Mona", 88000}, {114, "Nina", 90000}, {115, "Oscar", 65000},  
{116, "Paul", 62000}, {117, "Quinn", 57000}, {118, "Rachel", 61000},  
{119, "Steve", 72000}, {120, "Tina", 53000}  
};  
mergeSort(employees, 0, 19);  
printEmployees(employees, 20);  
return 0;  
}
```

- **Screen shots of Output:**

1.

#### Output

```
/tmp/quioG7uSrd.o
Enter details for employee:1:
Enter id:5
Enter name:Prachi
Enter salary:90000
Enter details for employee:2:
Enter id:3
Enter name:Sakshi
Enter salary:655876
Enter details for employee:3:
Enter id:4
Enter name:Janki
Enter salary:100000
Enter details for employee:4:
Enter id:1
Enter name:Tanisha
Enter salary:236567
Enter details for employee:5:
Enter id:2
Enter name:Avni
Enter salary:473633

Sorted employees by ID:
ID: 1,Name: Tanisha,Salary:236567
ID: 2,Name: Avni,Salary:473633
ID: 3,Name: Sakshi,Salary:655876
ID: 4,Name: Janki,Salary:100000
ID: 5,Name: Prachi,Salary:90000
```

2.



## Output

```
/tmp/5m5STeftCH.o
```

```
EMP-ID: 101, Name: Alice, Salary: 50000  
EMP-ID: 102, Name: Bob, Salary: 60000  
EMP-ID: 103, Name: Charlie, Salary: 55000  
EMP-ID: 104, Name: David, Salary: 70000  
EMP-ID: 105, Name: Eve, Salary: 80000  
EMP-ID: 106, Name: Frank, Salary: 75000  
EMP-ID: 107, Name: Grace, Salary: 65000  
EMP-ID: 108, Name: Hannah, Salary: 72000  
EMP-ID: 109, Name: Ivy, Salary: 58000  
EMP-ID: 110, Name: Jack, Salary: 54000  
EMP-ID: 111, Name: Karen, Salary: 69000  
EMP-ID: 112, Name: Leo, Salary: 72000  
EMP-ID: 113, Name: Mona, Salary: 88000  
EMP-ID: 114, Name: Nina, Salary: 90000  
EMP-ID: 115, Name: Oscar, Salary: 65000  
EMP-ID: 116, Name: Paul, Salary: 62000  
EMP-ID: 117, Name: Quinn, Salary: 57000  
EMP-ID: 118, Name: Rachel, Salary: 61000  
EMP-ID: 119, Name: Steve, Salary: 72000  
EMP-ID: 120, Name: Tina, Salary: 53000
```

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

- **Conclusion:**

Hence, we studied about various sorting techniques such as Quick Sort and Merge Sort with their Programs.