WAP to implement Quick sort and Merge Sort on 1D array of Employee structure (contains employee\_name, emp\_no, emp\_salary), with key as emp\_no. And count the number of swap performed.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Employee {

char employee\_name[50];

int emp\_no;

float emp\_salary;

};

// Function to swap two Employee structures

void swap(struct Employee \*a, struct Employee \*b, int \*count) {

struct Employee temp = \*a;

\*a = \*b;

\*b = temp;

(\*count)++;

}

// Quick Sort Partition function

int partition(struct Employee arr[], int low, int high, int \*count) {

int pivot = arr[high].emp\_no;

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j].emp\_no < pivot) {

i++;

swap(&arr[i], &arr[j], count);

}

}

swap(&arr[i + 1], &arr[high], count);

return (i + 1);

}

// Quick Sort function

void quickSort(struct Employee arr[], int low, int high, int \*count) {

if (low < high) {

int pi = partition(arr, low, high, count);

quickSort(arr, low, pi - 1, count);

quickSort(arr, pi + 1, high, count);

}

}

// Merge function for Merge Sort

void merge(struct Employee arr[], int l, int m, int r, int \*count) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

struct Employee L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i].emp\_no <= R[j].emp\_no) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

(\*count)++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort function

void mergeSort(struct Employee arr[], int l, int r, int \*count) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m, count);

mergeSort(arr, m + 1, r, count);

merge(arr, l, m, r, count);

}

}

int main() {

int n, i;

printf("Enter the number of employees: ");

scanf("%d", &n);

struct Employee employees[n];

printf("Enter employee details (name, emp\_no, emp\_salary):\n");

for (i = 0; i < n; i++) {

printf("Employee %d:\n", i + 1);

scanf("%s %d %f", employees[i].employee\_name, &employees[i].emp\_no, &employees[i].emp\_salary);

}

int quickSortCount = 0;

int mergeSortCount = 0;

// Copying the array for separate sorting in Quick Sort and Merge Sort

struct Employee quickSortArr[n], mergeSortArr[n];

memcpy(quickSortArr, employees, sizeof(employees));

memcpy(mergeSortArr, employees, sizeof(employees));

// Perform Quick Sort

quickSort(quickSortArr, 0, n - 1, &quickSortCount);

// Perform Merge Sort

mergeSort(mergeSortArr, 0, n - 1, &mergeSortCount);

printf("\nSorted array using Quick Sort (with emp\_no as key):\n");

for (i = 0; i < n; i++) {

printf("%s %d %.2f\n", quickSortArr[i].employee\_name, quickSortArr[i].emp\_no, quickSortArr[i].emp\_salary);

}

printf("Number of swaps performed in Quick Sort: %d\n", quickSortCount);

printf("\nSorted array using Merge Sort (with emp\_no as key):\n");

for (i = 0; i < n; i++) {

printf("%s %d %.2f\n", mergeSortArr[i].employee\_name, mergeSortArr[i].emp\_no, mergeSortArr[i].emp\_salary);

}

printf("Number of swaps performed in Merge Sort: %d\n", mergeSortCount);

return 0;

}