Experiment No.2

Title:-Quick Sort, Mearge Sort using array as a data Structure.

Problem:- E-commerce: Sorting products by price, rating, or popularity.

Programm:-

```
#include <stdio.h>
#include <string.h>
typedef struct {
    char name[50];
    int price;
    float rating;
    int popularity;
} Product;
// ------------------ Swap ------------------
void swap(Product *a, Product *b) {
    Product temp = *a;
    *a = *b;
    *b = temp;
}
// ------------------ Quick Sort ------------------
int compare_quick(Product a, Product b, char key[]) {
    if (strcmp(key, "price") == 0)
        return a.price - b.price;
    else if (strcmp(key, "rating") == 0)
        return (a.rating > b.rating) - (a.rating < b.rating); // returns 1,0,-1
    else if (strcmp(key, "popularity") == 0)
```

```
        return a.popularity - b.popularity;
    return 0;
}
int partition(Product arr[], int low, int high, char key[]) {
    Product pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (compare_quick(arr[j], pivot, key) < 0) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
void quickSort(Product arr[], int low, int high, char key[]) {
    if (low < high) {
        int pi = partition(arr, low, high, key);
        quickSort(arr, low, pi - 1, key);
        quickSort(arr, pi + 1, high, key);
    }
}
// ------------------ Merge Sort ------------------
int compare_merge(Product a, Product b, char key[]) {
    if (strcmp(key, "price") == 0)
        return a.price < b.price;
```

```
    else if (strcmp(key, "rating") == 0)
        return a.rating < b.rating;
    else if (strcmp(key, "popularity") == 0)
        return a.popularity < b.popularity;
    return 0;
}
void merge(Product arr[], int left, int mid, int right, char key[]) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    Product L[n1], R[n2];
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (compare_merge(L[i], R[j], key)) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
void mergeSort(Product arr[], int left, int right, char key[]) {
    if (left < right) {
        int mid = left + (right - left) / 2;
```

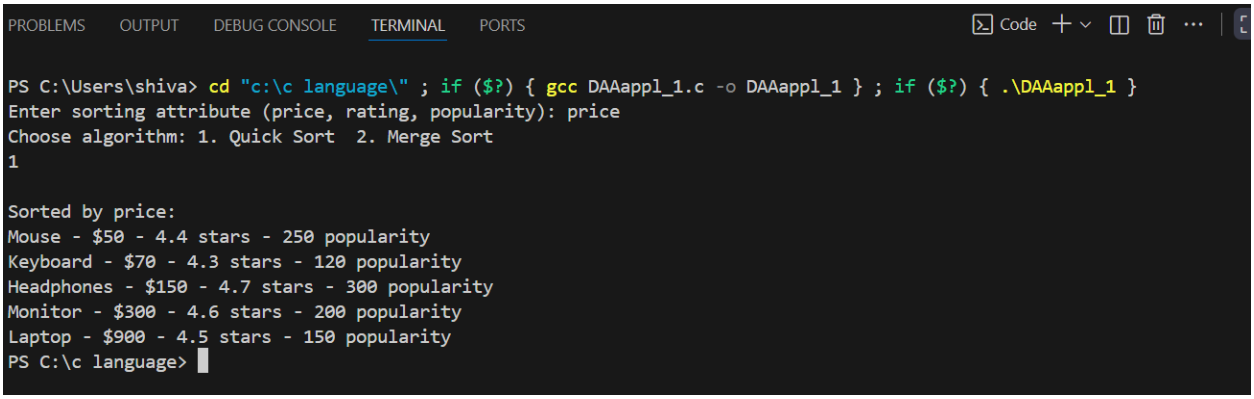
```
        mergeSort(arr, left, mid, key);
        mergeSort(arr, mid + 1, right, key);
        merge(arr, left, mid, right, key);
    }
}
// ------------------ Print Products ------------------
void printProducts(Product arr[], int n, char key[]) {
    printf("\nSorted by %s:\n", key);
    for (int i = 0; i < n; i++) {
        printf("%s - $%d - %.1f stars - %d popularity\n",
               arr[i].name, arr[i].price, arr[i].rating, arr[i].popularity);
    }
}
// ------------------ Main Function ------------------
int main() {
    Product products[] = {
        {"Laptop", 900, 4.5, 150},
        {"Headphones", 150, 4.7, 300},
        {"Keyboard", 70, 4.3, 120},
        {"Monitor", 300, 4.6, 200},
        {"Mouse", 50, 4.4, 250}
    };
    int n = sizeof(products) / sizeof(products[0]);

    char key[20];
    int choice;
```

```
    printf("Enter sorting attribute (price, rating, popularity): ");
    scanf("%s", key);

    printf("Choose algorithm: 1. Quick Sort  2. Merge Sort\n");
    scanf("%d", &choice);
    if (choice == 1)
        quickSort(products, 0, n - 1, key);
    else
        mergeSort(products, 0, n - 1, key);
    printProducts(products, n, key);
    return 0;
}
```

Output:-



Quick Sort Applications

1. E-commerce Product Sorting
    - Sorting products by price, rating, or popularity to display to customers efficiently.
2. Database Query Optimization
    - Sorting query results before applying further operations like grouping or filtering.

3. Computer Graphics
    - Sorting elements like z-values of objects for rendering (painter's algorithm) efficiently.
4. Networking
    - Sorting packets or requests by priority or timestamp before processing.
5. Game Development
    - Sorting scores, leaderboard entries, or in-game items dynamically.

Merge Sort Applications

1. External Sorting
    - Sorting large datasets that don't fit into memory (e.g., log files, transactions) using arrays/chunks.
2. Stable Sorting Requirements
    - When maintaining the original relative order is important (e.g., sorting a list of employees by salary while preserving order by name).
3. Linked List and Array Sorting in Software
    - Sorting arrays in scientific computations or sensor data where stability matters.
4. Data Compression
    - Sorting frequencies for Huffman coding in compression algorithms.
5. File Merging
    - Efficiently merging sorted data arrays or streams in applications like search engines, databases, or log aggregation.