SYMBIOSIS INSTITUTE OF TECHNOLOGY,HYDERABAD
A Constituent of Symbiosis International (Deemed University), Pune.
Re-accredited by NAAC with 'A++ Grade |Awarded Category - I by UGC

E-Commerce Product Sorter

Bachelor of Technology

in

Computer Science And Engineering

Submitted by

Dasari Ajay Kumar (PRN:24070721011)

M.Shashikiran (PRN:24070721019)

For the Fulfillment

of

Data Structure Algorithm Lab Course

Symbiosis Institute of Technology, Hyderabad

Survey Number:292, Off Bangalore Highway, Modallaguda Village,Nandigama Mandal, Ranga Reddy DIST, Near Hyderabad, Telangana- 509217.

Contents

1. Abstract 3
2. Introduction 4
3. Methodology 5
4. Code Explanation 12
5. Results and Discussion 14
6. Conclusion 14
7. Reference 15

Abstract

In the modern era of digital information systems, the ability to efficiently organize and retrieve data is crucial for both small and large-scale applications. Sorting is one of the most fundamental operations in computer science, playing an essential role in data management, searching, and presentation. This project focuses on implementing the Merge Sort algorithm to sort a collection of products based on multiple parameters such as price, rating, and category. The program is written in the C programming language and utilizes structures to represent each product with its specific attributes. Through this implementation, the project demonstrates the real-world application of sorting algorithms in managing and processing structured data efficiently.

The Merge Sort algorithm follows the divide and conquer paradigm, where the dataset is recursively divided into smaller parts until each sublist contains a single element, which is inherently sorted. The merging process then combines these sublists into larger sorted lists until the complete dataset is ordered according to the user's choice. This method ensures a consistent time complexity of O(n log n), making Merge Sort highly efficient even for large datasets. The algorithm also maintains stability, ensuring that records with equal keys retain their original order — an important property for applications that involve multi-attribute sorting.

The primary goal of this project is to strengthen the understanding of sorting algorithms, recursion, and modular programming concepts in C. It illustrates how theoretical knowledge of data structures and algorithms can be applied to solve real-life problems efficiently. Additionally, it highlights the importance of algorithm selection in determining the performance of data-processing applications. The project concludes that Merge Sort is a reliable and efficient method for handling structured datasets and can be extended to more complex systems involving file storage, databases, or graphical interfaces.

Through this project, students and developers gain valuable insights into the interplay between algorithm design and system efficiency. It demonstrates how clean code structure, proper use of functions, and user-oriented programming can result in robust and maintainable applications. The project not only fulfills academic objectives but also lays the foundation for future innovations in data management, algorithm optimization, and software development practices.

Introduction

In the field of computer science, data organization plays a vital role in ensuring efficiency and accuracy in data retrieval, analysis, and management. Sorting is one of the most fundamental and widely used operations in data processing, as it helps arrange data in a meaningful order for easy access and understanding. Whether it is in database management systems, inventory control, or e-commerce applications, sorting algorithms are indispensable in improving user experience and system performance. This project aims to demonstrate the implementation of one such powerful and efficient sorting technique — the Merge Sort algorithm — in the C programming language to organize product information based on user-selected parameters such as price, rating, and category.

The program utilizes a structured data approach by defining a Product structure containing attributes like name, price, rating, and category. This allows multiple data types to be grouped logically and processed efficiently. Users are prompted to input product details and can then choose from a menu to sort the data according to their preferences. The sorted data is displayed in a clear and formatted table, making it easy to analyze and compare products. This design not only enhances understanding of sorting algorithms but also shows how structured programming can simplify real-world data handling.

The Merge Sort algorithm is a well-known example of the divide and conquer strategy, which involves breaking down a large problem into smaller subproblems, solving them individually, and combining their results to produce the final sorted array. It works by recursively dividing the list into halves, sorting each half, and then merging the two sorted halves into a complete, ordered list. One of the key advantages of Merge Sort is its predictable and efficient performance — it consistently operates in O(n log n) time complexity, making it suitable for large datasets. Moreover, it is a stable sorting algorithm, which means that it maintains the relative order of equal elements — a desirable feature when sorting data with multiple attributes.

The implementation of Merge Sort in this project demonstrates several essential programming concepts, including recursion, modular design, and user interaction. The use of separate functions such as merge(), mergeSort(), and displayProducts() enhances the readability and maintainability of the code. The menu-driven interface allows users to interact dynamically with the program, giving them flexibility in sorting and viewing data according to different criteria. This not only makes the program more user-friendly but also helps in understanding how algorithmic logic can be integrated with real-world scenarios.

This project highlights the practical application of data structures and algorithms in solving everyday computational problems. By working with real-life examples such as product lists, the program bridges the gap between theoretical learning and practical implementation. It provides a hands-on approach to understanding how sorting algorithms influence system performance and how efficient data organization can improve decision-making processes.

In summary, this project serves as an effective demonstration of how Merge Sort can be applied in structured data management systems. It reinforces the importance of efficient algorithms in computer programming and provides a foundation for exploring more advanced techniques in data structures and algorithm design. Through this implementation, learners gain deeper insight into both the logic behind sorting and the principles of writing clean, efficient, and user-oriented C programs.

Methodology

The methodology of this project focuses on a structured and systematic approach to designing and implementing a program that sorts product information using the Merge Sort algorithm in the C programming language. The process involves multiple stages — understanding the problem, designing the data structure, implementing the algorithm, and testing its performance. Each step has been carefully planned to ensure accuracy, efficiency, and clarity in execution.

1. Problem Definition

The objective of this project is to create a C program that allows users to enter details of multiple products, including their name, price, rating, and category, and then sort this data according to the user's chosen criterion. The sorting should be flexible, allowing both ascending and descending orders. The program should also provide a clear and well-formatted output displaying the sorted product list.

2. Data Structure Design

To represent each product efficiently, a structure (struct) named Product is defined in C.It includes the following attributes:
char name[50]; → to store the product's name
float price; → to store the product's price
float rating; → to store the product's customer rating
char category[30]; → to store the category of the product
The use of structures allows multiple data types to be grouped together, providing an organized way to handle and manipulate related data.

3. Algorithm Selection

Among various sorting algorithms such as Bubble Sort, Selection Sort, Quick Sort, and Merge Sort, the Merge Sort algorithm was chosen for its efficiency and stability. Merge Sort follows the divide and conquer approach, where the dataset is divided into smaller sublists that are individually sorted and then merged into a final sorted list.
The key characteristics of Merge Sort include:
Time Complexity: O(n log n)
Space Complexity: O(n)
Stability: Maintains the order of elements with equal keys
Efficiency: Works effectively even for large datasets

4. Program Implementation

The program is implemented in modular form with clearly defined functions:
merge() → merges two sorted subarrays into one sorted array.
mergeSort() → recursively divides the array into subarrays and sorts them.
displayProducts() → displays the final list of products in a formatted table.

main() → handles input, user interaction, and menu-driven sorting choices.
The menu allows users to:
Sort by Price (Ascending/Descending)
Sort by Rating (Ascending/Descending)
Sort by Category (Alphabetical order)
Display all products
Exit the program

5. User Interaction and Input

The program begins by prompting the user to enter the number of products and their respective details. After input, the user can choose the sorting criterion from a menu. Based on the selection, the program calls the corresponding sorting function, processes the data, and displays the results.

6. Testing and Verification

After implementation, multiple test cases were executed to verify the correctness and stability of the sorting algorithm. Tests included:
Sorting by price and verifying ascending/descending order.
Sorting by rating and confirming the correct arrangement.
Sorting by category to check alphabetical ordering.
All test cases produced accurate and expected results, confirming that the algorithm functions correctly.

7. Output Display

The sorted product data is displayed in a tabular format showing all attributes neatly aligned. The output helps users clearly visualize how the sorting has reorganized the data.

Code

```
#include <stdio.h>
```

```
#include <string.h>

#define MAX 100

typedef struct {
    char name[50];
    float price;
    float rating;
    char category[30];
} Product;

// Merge function for merge sort
void merge(Product arr[], int left, int mid, int right, int choice, int ascending) {
    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) {
        int compare = 0;

        // Sorting choice
        if (choice == 1)
            compare = (L[i].price < R[j].price);
        else if (choice == 2)
            compare = (L[i].rating < R[j].rating);
        else if (choice == 3)
            compare = (strcmp(L[i].category, R[j].category) < 0);

        if ((ascending && compare) || (!ascending && !compare))
            arr[k++] = L[i++];
        else
```

```
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}

// Recursive Merge Sort
void mergeSort(Product arr[], int left, int right, int choice, int ascending) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid, choice, ascending);
        mergeSort(arr, mid + 1, right, choice, ascending);
        merge(arr, left, mid, right, choice, ascending);
    }
}

// Display products
void displayProducts(Product p[], int n) {
    printf("\n%-20s %-10s %-10s %-15s\n", "Name", "Price", "Rating", "Category");
    printf("-------------------------------------------------------------\n");
    for (int i = 0; i < n; i++)
        printf("%-20s %-10.2f %-10.1f %-15s\n", p[i].name, p[i].price, p[i].rating,
p[i].category);
}

int main() {
    Product products[MAX];
    int n, choice, asc, ch;

    printf("Enter number of products: ");
    scanf("%d", &n);
    getchar(); // consume newline

    for (int i = 0; i < n; i++) {
        printf("\nProduct %d\n", i + 1);
```

```
        printf("Name: "); fgets(products[i].name, 50, stdin);
products[i].name[strcspn(products[i].name, "\n")] = 0;
        printf("Price: "); scanf("%f", &products[i].price);
        printf("Rating: "); scanf("%f", &products[i].rating);
        getchar(); // consume newline
        printf("Category: "); fgets(products[i].category, 30, stdin);
products[i].category[strcspn(products[i].category, "\n")] = 0;
    }

    // Menu loop
    do {
        printf("\n==== PRODUCT SORTING MENU ====\n");
        printf("1. Sort by Price\n");
        printf("2. Sort by Rating\n");
        printf("3. Sort by Category\n");
        printf("4. Display Products\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
            case 2:
                printf("1. Ascending\n2. Descending\nEnter choice: ");
                scanf("%d", &asc);
                asc = (asc == 1);
                mergeSort(products, 0, n - 1, choice, asc);
                printf("\nProducts sorted successfully!\n");
                break;

            case 3:
                mergeSort(products, 0, n - 1, choice, 1);
                printf("\nProducts sorted by Category (Ascending)!\n");
                break;

            case 4:
                displayProducts(products, n);
                break;
```

```
            case 5:
                printf("\nExiting program...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);
    return 0;
}
```

-0

Output

```
Enter number of products: 3

Product 1
Name: tv
Price: 35000
Rating: 4.5
Category: electronics

Product 2
Name: shirt
Price: 3000
Rating: 4.2
Category: clothing

Product 3
Name: watch
Price: 20000
Rating: 3.5
Category: electronics

==== PRODUCT SORTING MENU ====
1. Sort by Price
2. Sort by Rating
3. Sort by Category
4. Display Products
5. Exit
Enter your choice: 1
1. Ascending
2. Descending
Enter choice: 1

Products sorted successfully!

==== PRODUCT SORTING MENU ====
1. Sort by Price
2. Sort by Rating
3. Sort by Category
4. Display Products
5. Exit
Enter your choice: 4

Name                Price      Rating     Category
-------------------------------------------------------------
shirt               3000.00    4.2        clothing
watch               20000.00   3.5        electronics
tv                  35000.00   4.5        electronics

==== PRODUCT SORTING MENU ====
1. Sort by Price
2. Sort by Rating
3. Sort by Category
4. Display Products
5. Exit
Enter your choice:
```

Code Explanation

1. Structure Definition

main.c

```c
#include <stdio.h>
#include <string.h>

#define MAX 100

typedef struct {
    char name[50];
    float price;
    float rating;
    char category[30];
} Product;
```

Explanation:
The Product structure defines how data is stored for each item. It includes fields for name, price, rating, and category, making it easy to handle multiple pieces of related information together.

2. Merge Function

```c
// Merge function for merge sort
void merge(Product arr[], int left, int mid, int right, int choice, int ascending) {
```

Explanation:
This function merges two sorted subarrays into a single sorted array.
Depending on the user's choice:

Choice 1 → Sort by Price

Choice 2 → Sort by Rating

Choice 3 → Sort by Category

The merge function maintains stability and ensures that elements are arranged properly in ascending or descending order.

3. Recursive Merge Sort

```
// Recursive Merge Sort
void mergeSort(Product arr[], int left, int right, int choice, int ascending) {
```

Explanation:
Implements the divide and conquer strategy by splitting the list into smaller halves, sorting each recursively, and merging them back together.
This function ensures O(n log n) time complexity and works efficiently for larger datasets.

4. Display Function

```
// Display products
void displayProducts(Product p[], int n) {
```

Explanation:
Prints the list of products in a neatly formatted table that shows all attributes—name, price, rating, and category.
It helps visualize how sorting has organized the data.

5. Main Function

```
int main() {
```

Explanation:
This is the main control section of the program. It:

Accepts user input for products

Displays a sorting menu

Calls mergeSort() based on the chosen criteria

Displays sorted results

It loops until the user chooses to exit.

Results and Discussion

The results validate that the implemented algorithm meets the project objectives. It sorts data accurately, displays it clearly, and allows user flexibility in selecting sorting parameters. The program also demonstrates how algorithmic design and structured programming principles can be effectively combined to produce efficient and reliable software solutions for real-world data management tasks.

Conclusion

This project successfully demonstrates the implementation of the Merge Sort algorithm in C for sorting product details based on price, rating, and category. The use of structures made data handling efficient, while the menu-driven approach provided an interactive and user-friendly experience. Merge Sort's divide and conquer technique ensured efficient performance with a time complexity of O(n log n) and maintained stability during sorting. Through this project, key programming concepts such as recursion, modular design, and structured data organization were effectively applied. Overall, the project highlights how

algorithmic logic can be used to solve real-world problems in data management and provides a strong foundation for further learning in data structures and algorithms.

REFERENCE

[1] Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, 2nd Edition, Prentice Hall, 1988.

[2] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, 2nd Edition, Universities Press, 2008.

[3] GeeksforGeeks, "Merge Sort Algorithm in C – Explained with Examples." Available at: https://www.geeksforgeeks.org/merge-sort/

[4] TutorialsPoint, "C Programming – Structures and Functions." Available at: https://www.tutorialspoint.com/cprogramming/

[5] W3Schools, "C Programming Tutorial." Available at: https://www.w3schools.com/c/

[6] Lecture Notes on Data Structures and Algorithms, Symbiosis Institute of Technology, Hyderabad, 2025.

[7] Javatpoint, "Sorting Algorithms in C." Available at: https://www.javatpoint.com/sorting-in-c