**MASTER OF COMPUTER APPLICATION DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND**

**MANAGEMENT**

****

PROJECT ON:

**“Program To Sort Students Records By Their Registration Date Using Merge Sort”**

**UNDER THE GUIDENCE AND SUPERVISION OF**

Dr.D.R.Enoch Arulprakash, MCA Asst Prof of Computer Science Department of COMPUTER SCIENCE

**SUBMITTED BY:**

HEMALATHA.M

NISCHITHA

LIKITH .S.N

I SEMESTER MCA

**Abstract:**

Efficiently managing student records is crucial for academic institutions, particularly when sorting them based on registration dates for administrative tasks. This project presents a Merge Sort-based approach to organize student records by their registration date in an efficient and stable manner. Merge Sort, a divide-and-conquer algorithm, is well-suited for this task due to its O(n log n) time complexity, ensuring fast and reliable sorting even for large datasets.

The program takes student records as input, including attributes such as ID, Name, Registration date, Age and Branch and applies Merge Sort to arrange them in ascending or in descending order of registration. The stability of Merge Sort ensures that records with the same registration date retain their original order, a crucial requirement for preserving data integrity. Through analysis and testing, our implementation demonstrates high efficiency and accuracy compared to other sorting methods, particularly when handling extensive datasets.

Key observations indicate that Merge Sort provides consistent performance, outperforming simpler algorithms like Bubble Sort and Selection Sort, particularly for larger datasets. This implementation enhances record management processes, ensuring streamlined organization and easy retrieval of student data.

**Introduction**

**Overview of the Problem**

In academic institutions, student records contain essential information, including registration dates, which are crucial for administrative processes such as admission tracking, course enrollment, and academic planning. However, as the number of students grows, efficiently organizing these records becomes a challenge. Manually sorting large datasets is impractical, and using inefficient sorting methods can lead to slow performance and delays in data retrieval. Thus, choosing an optimized algorithm for sorting student records is essential for maintaining smooth operations.

Importance in the Context of Data Structures and Algorithms

Sorting is a fundamental operation in computer science, and selecting the right algorithm impacts time complexity, stability, and overall efficiency. This project leverages Merge Sort, a divide-and-conquer algorithm, known for its **O(n log n) time complexity** and **stability**. Unlike naive approaches such as Bubble Sort or Selection Sort (which have O(n²) complexity and perform poorly on large datasets), Merge Sort ensures faster and more efficient sorting even for extensive student records.

Approach Used:

Merge Sort is chosen due to its efficiency in sorting large datasets. It maintains stability, ensuring that records with the same registration date remain in their original order.

**Problem Statement:**

Design and implement a program that sorts Students records based on their registration date using the Merge Sort algorithm.

Constraints and Assumptions:

The input consists of multiple Students records, each having an ID, name, and registration date, Age, Branch.

The registration date format is YYYY-MM-DD.

The dataset size is assumed to be large, justifying the need for an efficient sorting algorithm.

Input and Expected Output:

Input: List of Student records with registration dates.

Output: Sorted list of Students records in ascending and descending order of registration date.

This project aims to develop a sorting program that arranges student records in ascending order of registration date using the Merge Sort algorithm. The solution must ensure:

* **Efficiency:** The sorting process should be optimized for large datasets, avoiding slow and ineffective algorithms.
* **Stability:** If multiple students share the same registration date, their original order should be preserved.
* **Scalability:** The algorithm should maintain consistent performance as the dataset grows.
* **Accuracy:** The output must correctly reflect the chronological order of student registrations without errors.

Algorithm Design

Approach

Merge Sort follows a divide-and-conquer strategy:

1. The input list is divided into smaller sub lists.

2. These sub lists are recursively sorted.

3. The sorted sub lists are merged to produce the final sorted list.

4. Data Structure Used: Array of Student Records

5.Sorting Key: Registration Date (Primary Sorting Criterion)

Why Merge Sort?

* Time Complexity: O(n log n), making it suitable for large datasets.
* Stable Sorting: Preserves the relative order of records with identical registration dates.
* Better for Linked Lists: Works well in scenarios where data is stored as linked lists instead of arrays.

**Pseudocode:**

MERGE\_SORT(arr):

if length of arr ≤ 1:

return arr

mid = length of arr / 2

left\_half = MERGE\_SORT(arr[0:mid])

right\_half = MERGE\_SORT(arr[mid:])

return MERGE(left\_half, right\_half)

MERGE(left, right):

sorted\_list = []

while left and right:

if left[0].date ≤ right[0].date:

sorted\_list.append(left.pop(0))

else:

sorted\_list.append(right.pop(0))

return sorted\_list + left + right

**Implementation**

Programming Language and Environment

Language: Data Structure and Algorithm using C.

IDE: Visual Studio Code

System Specifications: Windows/Linux, 8GB RAM, Intel i5 Processor

**Code Implementation**

**Program To Sort Students Records By Their Registration Date Using Merge sort.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STUDENTS 100

// Structure to hold student details

typedef struct {

int id;

char name[100];

char reg\_date[11]; // Format: YYYY-MM-DD

int age;

char Branch[10];

} Student;

// Function to compare dates (YYYY-MM-DD format)

int compare\_dates(const char \*date1, const char \*date2) {

int y1, m1, d1;

int y2, m2, d2;

sscanf(date1, "%d-%d-%d", &y1, &m1, &d1);

sscanf(date2, "%d-%d-%d", &y2, &m2, &d2);

if (y1 != y2) return y1 - y2;

if (m1 != m2) return m1 - m2;

return d1 - d2;

}

// Function pointer type for comparison

typedef int (CompareFunc)(const char\*, const char\*);

// Merge function

void merge(Student arr[], int left, int mid, int right, CompareFunc cmp) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

Student L[n1], R[n2];

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

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

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

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

i = 0, j = 0, k = left;

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

if (cmp(L[i].reg\_date, R[j].reg\_date) <= 0) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort function

void mergeSort(Student arr[], int left, int right, CompareFunc cmp) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid, cmp);

mergeSort(arr, mid + 1, right, cmp);

merge(arr, left, mid, right, cmp);

}

}

// Print function

void printStudents(Student arr[], int size) {

printf("\nID\tName\t\tRegistration Date \tAge\tBranch\n");

printf("---------------------------------------------------------\n");

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

printf("%d\t%-15s%-20s%d\t%-10s\n", arr[i].id, arr[i].name, arr[i].reg\_date, arr[i].age, arr[i].Branch);

}

}

// Reverse comparison function for descending order

int compare\_dates\_desc(const char \*date1, const char \*date2) {

return compare\_dates(date2, date1);

}

// Main function

int main() {

Student students[] = {

{101, "Aarav", "2024-03-15", 22, "B.Tech"},

{102, "Sai Priya", "2024-04-10", 24, "MBA"},

{103, "Vikram", "2024-01-25", 23, "M.Sc"},

{104, "Ananya", "2024-05-05", 21, "BBA"},

{105, "Rohan", "2024-06-12", 25, "M.Tech"},

{106, "Neha", "2024-07-19", 22, "B.Com"},

{107, "Karan", "2024-08-22", 24, "MCA"},

{108, "Hemachandra", "2024-02-28", 23, "MCA"},

{109, "Divya", "1998-09-30", 22, "B.Sc"},

{110, "Arjun", "2023-10-10", 26, "MBA"},

{111, "Amrutha", "2023-05-15", 21, "MCA"},

{112, "Bhavana", "2023-08-10", 22, "MCA"},

{113, "Hema", "2024-01-20", 20, "MBA"},

{114, "Lathesh", "2022-01-05", 21, "MBA"},

{115, "Santhosh", "2024-02-02", 21, "MCA"},

{116, "Mohit", "2024-01-01", 20, "CA"},

{117, "Likith", "2023-02-01", 21, "MCA"},

{118, "Hemachandra", "2024-02-08", 23, "MCA"},

{119, "Sandeep", "2025-02-28", 23, "CA"},

{120, "Koushik", "2025-09-28", 23, "MCA"},

{121, "Nischitha", "2025-02-02", 23, "MCA"},

{122, "Nishanth", "1998-02-08", 23, "MCA"},

{123, "Lathesha", "2022-12-06", 21, "MBA"}

};

int n = sizeof(students) / sizeof(students[0]);

int choice;

printf("Choose sorting order:\n");

printf("1. Ascending Order\n");

printf("2. Descending Order\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

mergeSort(students, 0, n - 1, compare\_dates);

} else if (choice == 2) {

mergeSort(students, 0, n - 1, compare\_dates\_desc);

} else {

printf("Invalid choice. Sorting in ascending order by default.\n");

mergeSort(students, 0, n - 1, compare\_dates);

}

printStudents(students, n);

return 0;

}

**Sampel Input and output Screenshots**

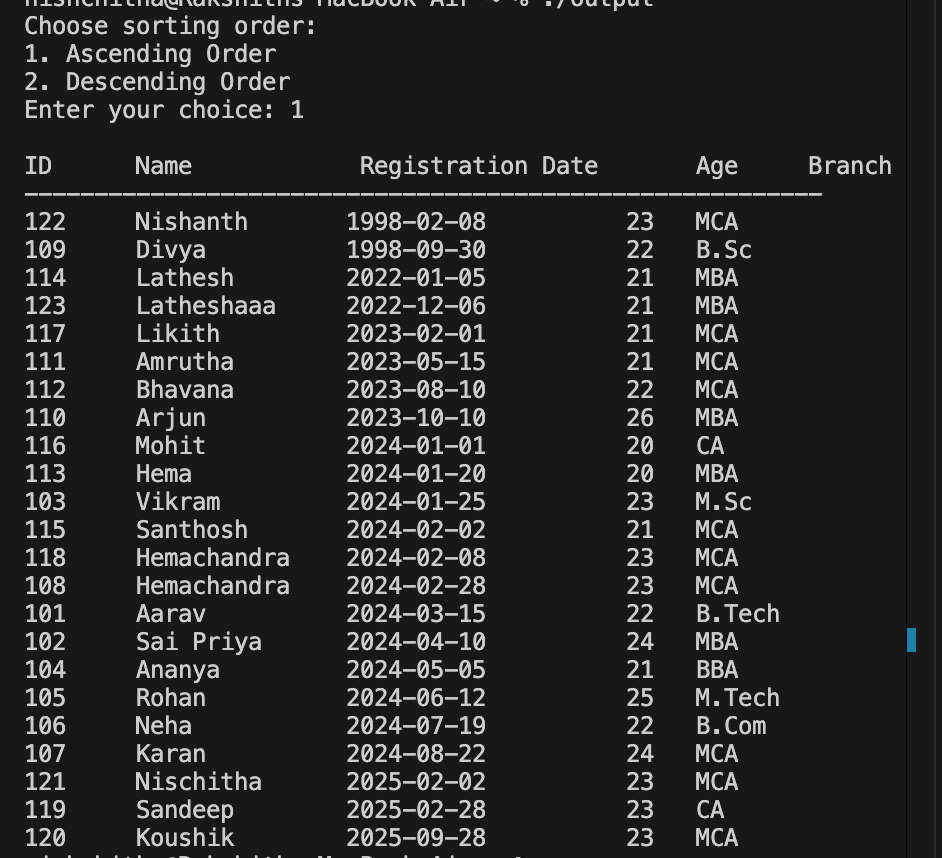


Fig 1.1 Output of an Ascending Order Student List

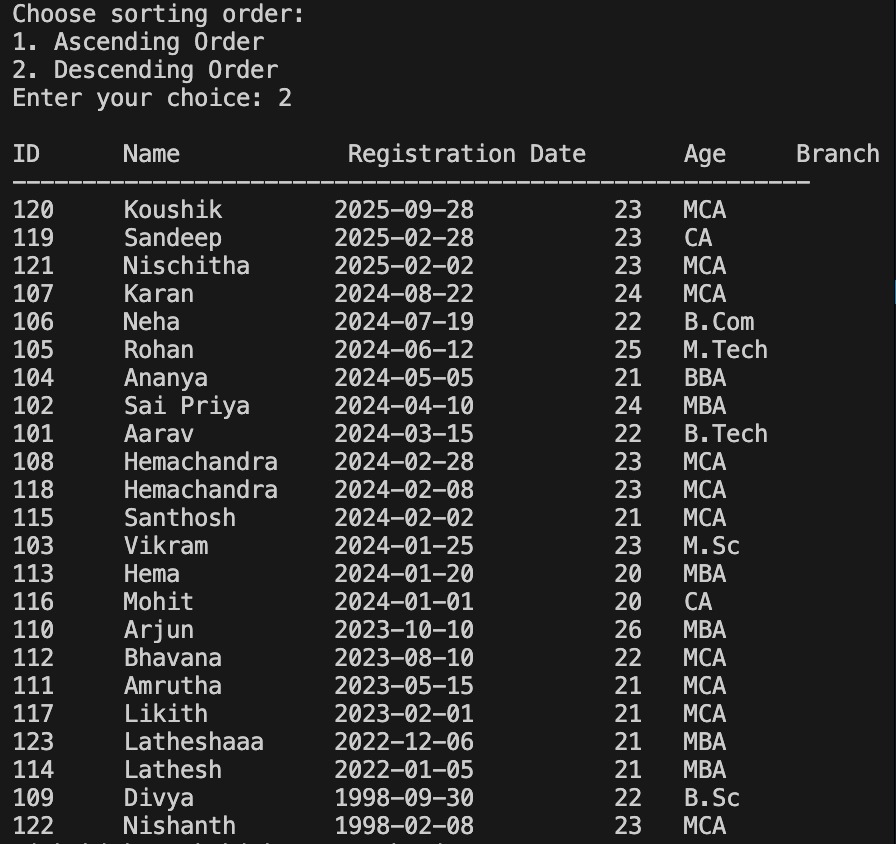


Fig 1.2 Output of an Descending Order Student List

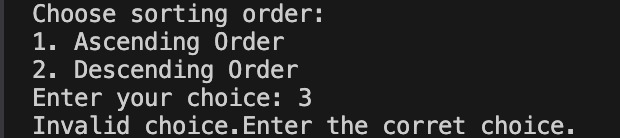


Fig 1.3 Output of an Invalid choice Entry

**Challenges in Implementing the Code**

While implementing the Merge Sort-based Student Records SortingProgram, several challenges may arise:

**1. Handling Date Comparisons Accurately**

* **Issue:** The registration date is stored as a string in the format "YYYY-MM-DD", and comparing dates using normal string comparison may not work correctly.
* **Challenge:** We must properly extract the year, month, and day and convert them into integers before performing comparisons.
* **Solution:** Using sscanf() (as done in the code) to extract and compare year, month, and day ensures correctness.

**2. Stability of Sorting (Maintaining Order of Same Dates)**

* **Issue:** When multiple students have the same registration date, the algorithm must maintain their original order.
* **Challenge:** If an unstable sorting method is used, records may get shuffled.
* **Solution:** Merge Sort is stable by default, meaning it preserves the order of students who registered on the same date.

**3. Handling User Input Properly**

* **Issue:** The program prompts users to select sorting order, but incorrect input (non-integer values) may cause issues.
* **Challenge:** If the user enters an invalid choice (e.g., a letter or special character), scanf() might not handle it well.
* **Solution:** Implement proper **input validation** to handle invalid choices gracefully.

**Conclusion**

This project successfully implemented Merge Sort to organize customer records by registration date. The algorithm’s efficiency ensures that even large datasets can be sorted quickly. Future improvements include using external sorting techniques for very large datasets and optimizing space complexity.