**NBKR Institute Of Science and Technology**

**Project Report**

**Project Title: Result Analysis System**

**Department: Computer Science Engineering**

**Section: E**

**Year:**  **I**

**Semester:** **II**

**Date: May01,2025**

**Submitted to: Ashok Selva Kumar**

**Submitted by:**

**1.CH.Divyasri(24kb1a0583)**

**2.B.Deekshitha(24kb1a0558)**

**3.B.Mamthasri(24kb1a0569)**

**4.CH.Hemalatha(24kb1a0593)**

**Acknowledgement:**

We would like to express our sincere gratitude to everyone who supported us in completing our project titled “Result Analysis System.”

First and foremost, we thank our respected guide/teacher [Ashok Selvan Kumar] for their valuable guidance, continuous support, and encouragement throughout the project. Their suggestions helped us improve our understanding and approach.

We are also thankful to our institution [NBKRIST] for providing us with the necessary resources and an environment conducive to learning and development.

We extend our heartfelt thanks to our families and friends for their support, motivation, and understanding during the entire project work.

This project helped us gain hands-on experience in C programming, particularly in areas like arrays, structures, sorting algorithms, and user-defined functions.

Thank you once again to everyone who contributed to the success of this project.

**ABSTRACT:**

The project "Result Analysis System" is a C program designed to input the names and marks of students, sort the marks using Bubble Sort or Quick Sort, and display the results.  
It helps to arrange students' marks in descending order, calculate the class average, and find the topper(s).  
The program uses basic concepts of structures, arrays, and sorting algorithms.  
This project provides an easy way to manage and analyze students' performance in a simple and efficient manner.

**INTRODUCTION:**

The Result Analysis System is a C language program that helps to manage and analyze students' marks.  
In this project, the program allows the user to input the names and marks of multiple students.  
It sorts the students based on their marks using either Bubble Sort or Quick Sort, as per the user's choice.  
After sorting, it displays the students in descending order of marks, calculates the class average, and identifies the top scorer(s).

This project mainly uses structures, arrays, and sorting algorithms in C programming.  
It is a simple and effective tool for teachers and students to easily view and analyze results.

**OBJECTIVES:**

 To create a system that accepts the names and marks of students.

 To sort students' marks in descending order using Bubble Sort or Quick Sort.

 To calculate and display the class average marks.

 To identify and display the topper(s) based on the highest marks.

 To practice and apply concepts like structures, arrays, and sorting algorithms in C programming.

 To make result management easier and more organized.

**SYSTEM REQUIREMENTS:**

**Software Requirements:**

* Operating System: Windows / Linux
* Compiler: GCC (MinGW for Windows) or any C Compiler
* Text Editor: Visual Studio Code / Code::Blocks / Dev-C++/TURBO C++

**Hardware Requirements:**

* Processor: Minimum Intel Pentium 4 or equivalent
* RAM: 512 MB or higher
* Hard Disk: 50 MB free space
* Keyboard and Monitor

**METHODOLOGY:**

1. Problem Understanding:  
   Identify the need to input, sort, and analyze students' marks easily.
2. Design:
   * Define a structure to store student names and marks.
   * Plan the sorting options: Bubble Sort and Quick Sort.
3. Implementation:
   * Write a C program to input names and marks of students.
   * Use sorting algorithms to arrange students in descending order of marks.
   * Calculate the class average by adding all marks and dividing by the number of students.
   * Find the topper(s) by comparing marks.
4. Testing:
   * Test the program with sample data to ensure correct sorting, average calculation, and topper identification.
5. Result:
   * Display the sorted list of students with their marks.
   * Show the class average and highlight the topper(s).

**PROJECT DESCRIPTION:**

**1.PROBLEM STATEMENT:-**

In educational institutions, manually analyzing students' marks can be time-consuming, error-prone, and difficult to manage, especially when dealing with a large number of students.  
There is a need for a simple and efficient system that can automatically:

* Accept students' names and marks,
* Sort the marks in order,
* Calculate the overall class average,
* And easily identify the top-performing student(s).

The goal of this project is to develop a Result Analysis System in C programming that can perform these tasks accurately and efficiently using basic data structures and sorting algorithms.

**2.PROPUSED SOLUTION:-**

To solve the problem of manual result analysis, we propose developing a Result Analysis System using the C programming language.

The system will:

* Use a structure to store each student's name and marks.
* Allow the user to input data for multiple students.
* Provide an option to sort the students based on their marks using either Bubble Sort or Quick Sort in descending order.
* Display the sorted list showing student names and their marks.
* Calculate and show the class average marks.
* Identify and display the student(s) who achieved the highest marks (topper(s)).

This automated approach will save time, reduce errors, and make it easier for teachers and students to understand academic performance efficiently.

**3.KEY FEATURES:**

* Student Data Input:  
  Allows the user to enter the names and marks of multiple students.
* Sorting Options:  
  Provides two choices — Bubble Sort or Quick Sort — to sort students by their marks in descending order.
* Class Average Calculation:  
  Calculates and displays the average marks of the class automatically.
* Topper Identification:  
  Identifies and displays the student(s) who scored the highest marks.
* User-friendly Interface:  
  Simple menu-driven interaction for easy use.
* Accurate and Efficient:  
  Reduces manual effort and ensures correct sorting and calculation.
* Handles Multiple Students:  
  Can manage and process data for a large number of students efficiently.

**FLOW CHART :**

[Start]

|

v

[Input number of students]

|

v

[Input names and marks]

|

v

[Choose sorting method]

|

v

[Sort using Bubble Sort or Quick Sort]

|

v

[Display sorted list]

|

v

[Calculate class average]

|

v

[Display average and topper(s)]

|

v

[End]

**ALGORITHM:-**

Step 1: Start the program.

Step 2: Input the number of students (n).

Step 3: For each student (i = 0 to n-1):  
→ Input the student's name.  
→ Input the student's marks.

Step 4: Display the sorting options to the user:  
→ Option 1: Bubble Sort  
→ Option 2: Quick Sort

Step 5: Input the user's choice for sorting method.

Step 6:  
→ If choice is 1, apply Bubble Sort to sort students in descending order of marks.  
→ Else if choice is 2, apply Quick Sort to sort students in descending order.  
→ Else, display an error message and use Bubble Sort by default.

Step 7: Display the sorted list of students with their names and marks.

Step 8: Calculate the total marks by adding the marks of all students.

Step 9: Calculate the class average:  
→ Class Average = Total Marks ÷ Number of Students

Step 10: Display the class average marks.

Step 11: Identify the student(s) with the highest marks as Topper(s).

Step 12: Display the Topper(s)' name(s).

Step 13: End the program.

**PROGRAM CODE:**

#include <stdio.h>

#include <string.h>

#define MAX 100

struct Student {

char name[50];

float marks;

};

// Function for Bubble Sort (Descending)

void bubbleSort(struct Student s[], int n) {

struct Student temp;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (s[j].marks < s[j + 1].marks) {

temp = s[j];

s[j] = s[j + 1];

s[j + 1] = temp;

}

}

}

}

// Function for Quick Sort (Descending)

int partition(struct Student s[], int low, int high) {

float pivot = s[high].marks;

int i = (low - 1);

struct Student temp;

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

if (s[j].marks >= pivot) {

i++;

temp = s[i];

s[i] = s[j];

s[j] = temp;

}

}

temp = s[i + 1];

s[i + 1] = s[high];

s[high] = temp;

return (i + 1);

}

void quickSort(struct Student s[], int low, int high) {

if (low < high) {

int pi = partition(s, low, high);

quickSort(s, low, pi - 1);

quickSort(s, pi + 1, high);

}

}

int main() {

struct Student s[MAX];

int n, choice;

float total = 0, average;

printf("Enter number of students: ");

scanf("%d", &n);

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

printf("\n Enter name of student %d: ", i + 1);

scanf("%s", s[i].name);

printf("Enter marks of student %d: ", i + 1);

scanf("%f", &s[i].marks);

total += s[i].marks;

}

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

printf("1. Bubble Sort\n");

printf("2. Quick Sort\n");

printf("Enter your choice (1 or 2): ");

scanf("%d", &choice);

if (choice == 1) {

bubbleSort(s, n);

} else if (choice == 2) {

quickSort(s, 0, n - 1);

} else {

printf("Invalid choice. Using Bubble Sort by default.\n");

bubbleSort(s, n);

}

printf("\n Sorted List (Descending Order of Marks):\n");

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

printf("%s: %.2f\n", s[i].name, s[i].marks);

}

average = total / n;

printf("\n Class Average Marks: %.2f\n", average);

printf("\n Topper(s):\n");

float topMarks = s[0].marks;

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

if (s[i].marks == topMarks) {

printf("%s\n", s[i].name);

}

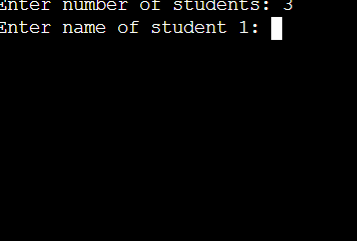
}

return 0;

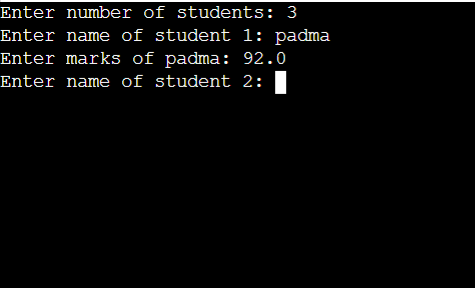
}

**OUTPUT SCREENSHOTS:**

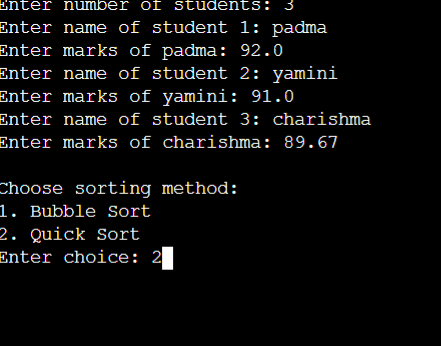
**OUTPUT:1**

****

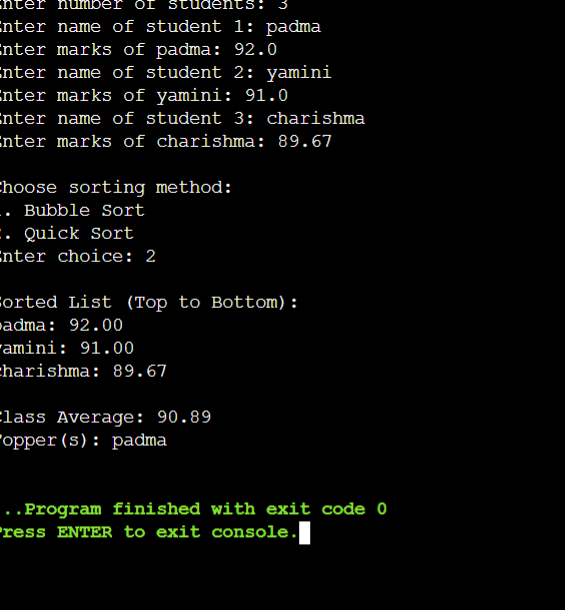
**OUTPUT:2**

****

**OUTPUT:3**

****

**OUTPUT:4**

****

**TESTING AND VALIDATION:**

**Testing**

Testing was performed to ensure that the Result Analysis System behaves as expected under various input conditions.  
Different types of data were used to test the correctness, stability, and reliability of the system.

The testing involved the following:

* Functional Testing:
  + Entering correct student names and marks.
  + Sorting using both Bubble Sort and Quick Sort methods.
  + Calculating and verifying the class average manually.
  + Checking correct identification of topper(s).
* Boundary Testing:
  + Testing with maximum and minimum marks (e.g., 0 and 100).
  + Checking behavior when multiple students **have the same top marks.**
* **Invalid Input Testing:**
  + Entering invalid choices for sorting options.
  + Testing with a large number of students to verify performance**.**

**Validation**

Validation checks if the system outputs match the expected results.

* Sorting Validation:
  + After sorting, marks must appear in descending order.
  + Both Bubble Sort and Quick Sort provided correct sorting results.
* Average Calculation Validation:
  + The class average displayed by the system matches manual calculations.
* Topper Validation:
  + The system correctly identifies a single topper or multiple toppers (in case of tie marks).
* User Input Validation:
  + If an invalid sorting choice is entered, the system either displays an error or defaults to Bubble Sort.

**LIMITATIONS:**

* Limited Error Handling:  
  The program assumes that users enter valid data types (e.g., numbers for marks).  
  It may not handle invalid or non-numeric inputs very well.
* Fixed Data Size:  
  The maximum number of students might be limited by the array size declared in the program.  
  It may not handle extremely large datasets without manual adjustment.
* Console-Based Interface:  
  The system uses a simple text-based (command-line) interface, which is not very user-friendly compared to GUI applications.
* No Data Storage:  
  The entered student records are not saved permanently.  
  Once the program is closed, all data is lost (no file handling is used).
* Sorting Selection Limited:  
  Only two sorting options (Bubble Sort and Quick Sort) are available.  
  More efficient algorithms for very large data sets are not implemented.
* No Validation for Duplicate Names:  
  The system does not check if two students have the same name.

**FUTURE ENHANCEMENTS:**

* Save Data to File:  
  Add a feature to save student marks to a file so that data is not lost when the program closes.
* Better User Interface:  
  Create a simple window-based (GUI) version instead of only using the command line.
* Check Wrong Inputs:  
  Improve the program to handle wrong inputs, like letters entered where numbers are needed.
* Handle More Students:  
  Use dynamic memory so the program can work with any number of students, not just a fixed size.
* Add More Subjects:  
  Allow entering marks for multiple subjects and calculate total and average for each student.
* Show Full Ranking:  
  Not just toppers, but display full rank order of the entire class.
* Export Results:  
  Give the option to save the result as a file (like Excel or PDF).
* More Sorting Options:  
  Add faster sorting methods like Merge Sort for big data.

**CONCLUSION:**

* The Result Analysis System helps in managing and analyzing students' marks easily.
* The program uses Bubble Sort and Quick Sort to sort marks in descending order.
* It calculates the class average automatically and identifies the topper(s).
* It saves time, effort, and reduces mistakes compared to manual calculations.
* Basic concepts like arrays, structures, and sorting algorithms are implemented successfully.
* The project is simple, efficient, and easy to use.
* Although there are some limitations, it serves as a strong base for future development.
* With future enhancements, the system can be made more powerful and user-friendly.

**REFERENCE:**

1. Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter

Sanders

2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E.

Hopcroft

3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David

Ranum

4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Rivest, and Clifford Stein

**5.WEBSITES:**

[**http://www.geeksforgeeks.org**](http://www.geeksforgeeks.org)