



PIMPRI CHINCHWAD EDUCATION TRUST'S.  
**PIMPRI CHINCHWAD COLLEGE OF ENGINEERING**  
(An Autonomous Institute)

---

**Class : SY BTech****Acad. Yr. 2025-26****Semester : I****Name of the student: Om Jitendra Khalane****PRN : 124B1B040****Department: Computer Engineering****Division : A****Course Name : Data Structures and Laboratory****Course Code: BCE23PC02****Completion Date : 23/07/2025**

---

## Assignment No. 01

### Problem Statement:

An education platform needs to rank students based on their exam scores to publish results and generate merit lists. Write a program for above scenario.

*Hint:*

*Given a list of student scores, use Quick Sort to sort them in descending order. Handle cases where many students have the same score and analyze how this affects the performance of Quick Sort.*

### Source Code :

[https://github.com/omkhalane/DSAL-SY-PCCOE/blob/main/lab\\_assignments/assignment01.cpp](https://github.com/omkhalane/DSAL-SY-PCCOE/blob/main/lab_assignments/assignment01.cpp)

```
#include<bits/stdc++.h>
using namespace std;

struct Student {
    string name;
    double score;
};

// Partition
int partition(vector<Student> &arr, int low, int high) {
    int pivot = arr[high].score;
    int i = low - 1;
```

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

            if (arr[j].score > pivot) {
                i++;
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[i + 1], arr[high]);
        return i + 1;
    }

// Quick Sort
void quickSort(vector<Student> &arr, int low, int high) {
    if (low < high) {
        int pIndex = partition(arr, low, high);

        quickSort(arr, low, pIndex - 1);
        quickSort(arr, pIndex + 1, high);
    }
}

int main() {
    int n;
    cout << "Enter number of students: ";
    cin >> n;
    cout << endl<<endl;

    vector<Student> students(n);
    for (int i = 0; i < n; i++) {
        cout << "Enter name of student " << i + 1 << ": ";
        getchar();
        getline(cin, students[i].name);

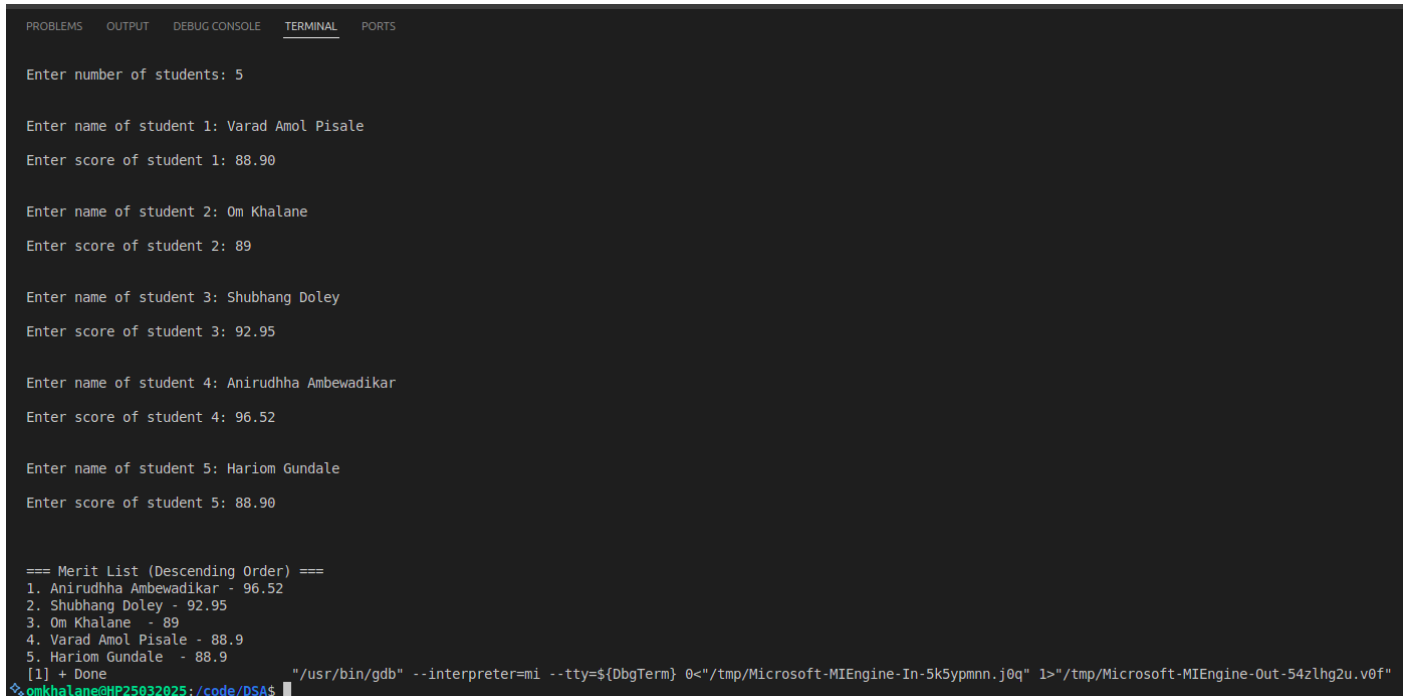
        cout<<endl<< "Enter score of student " << i + 1 << ": ";
        cin >> students[i].score;
        cout<<endl<<endl;
    }

    quickSort(students, 0, n - 1);

    cout << "\n=== Merit List (Descending Order) ===\n";
    for (int i = 0; i < n; i++) {
        cout << i + 1 << ". " << students[i].name << " - " <<
            students[i].score << endl;
    }
}
```

```
}  
return 0; }
```

### Screen Shot of Output :



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
Enter number of students: 5  
  
Enter name of student 1: Varad Amol Pisale  
Enter score of student 1: 88.90  
  
Enter name of student 2: Om Khalane  
Enter score of student 2: 89  
  
Enter name of student 3: Shubhang Doley  
Enter score of student 3: 92.95  
  
Enter name of student 4: Anirudhha Ambewadikar  
Enter score of student 4: 96.52  
  
Enter name of student 5: Hariom Gundale  
Enter score of student 5: 88.90  
  
=== Merit List (Descending Order) ===  
1. Anirudhha Ambewadikar - 96.52  
2. Shubhang Doley - 92.95  
3. Om Khalane - 89  
4. Varad Amol Pisale - 88.9  
5. Hariom Gundale - 88.9  
[1] + Done  
omkhalane@HP25032025: /code/DSA$ "
```

### Conclusion:

In this assignment, we implemented a Quick Sort-based ranking system to arrange students in descending order of their exam scores, forming the basis for generating a merit list.

Quick Sort was chosen due to its average-case efficiency and in-place sorting capability, making it a popular choice for handling large datasets where memory usage matters.

---

### Observations from Implementation

- Descending order modification was straightforward — we simply adjusted the comparison in the partition step from  $>$  to  $<$  logic (or vice versa), ensuring higher scores appear first in the list.
- The algorithm handled duplicate scores correctly by grouping them together naturally.
- However, the ordering among students with identical scores was not guaranteed because Quick Sort is not stable. If stability were required (to preserve the order of students who scored equally), Merge Sort or a stable variant of Quick Sort would be better.

## Time Complexity Analysis

Scenario	Complexity	Explanation
Best Case	$O(n \log n)$	Occurs when the pivot divides the list into two equal halves at every recursive call.
Average Case	$O(n \log n)$	In most random distributions, partitions are reasonably balanced, leading to efficient sorting.
Worst Case	$O(n^2)$	Happens when the pivot selection is poor — for example, when the list is already sorted or all scores are the same. This causes highly unbalanced partitions.

---

## Space Complexity Analysis

- Space Complexity (In-place):  $O(\log n)$   
This is because Quick Sort uses recursive calls, and the space is proportional to the depth of the recursion tree.
  - No additional arrays are needed, making it memory efficient compared to algorithms like Merge Sort which require  $O(n)$  extra space.
- 

This assignment demonstrates not just how to implement Quick Sort, but also how algorithmic choices affect performance, stability, and real-world usability. In an educational ranking context:

- Average performance is excellent ( $O(n \log n)$ ), and memory usage is low.
  - Duplicates are handled correctly, but performance with many duplicates can be improved with a 3-way partitioning.
  - This is a scalable and efficient approach for large datasets, but algorithm choice should also consider tie-breaking needs, data patterns, and stability requirements.
-