ASSIGNMENT NO.1 (USE BUBBLE SORT)

```
#include <iostream>

#include <string>


using namespace std;


// Structure to represent a student

struct Student {

   int rollNo;

   string name;

   float sgpa;

};


// Function to perform bubble sort on the student database based on roll numbers

void bubbleSort(Student students[], int n) {

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

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

         if (students[j].rollNo > students[j + 1].rollNo) {

            // Swap the students if they are in the wrong order

            swap(students[j], students[j + 1]);

         }

      }

   }

}


// Function to print the roll call list

void printRollCallList(const Student students[], int n) {

   cout << "Roll Call List:" << endl;

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

      cout << "Roll No: " << students[i].rollNo << ", Name: " << students[i].name << ", SGPA: " <<
students[i].sgpa << endl;
```

```
    }
}

int main() {
    const int maxSize = 100;

    // Input: Number of students in the database
    int n;
    cout << "Enter the number of students: ";
    cin >> n;

    // Input: Student information
    Student students[maxSize];
    cout << "Enter the student information:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Student " << i + 1 << " -" << endl;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Name: ";
        cin.ignore(); // To clear the newline character from the buffer
        getline(cin, students[i].name);
        cout << "SGPA: ";
        cin >> students[i].sgpa;
        cout << endl;
    }

    // Perform bubble sort to arrange students in ascending order of roll numbers
    bubbleSort(students, n);

    // Print the roll call list
    printRollCallList(students, n);
    return 0;
}
```

ASSIGNMENT 2: (USE INSERTION SORT)

```
#include <iostream>
#include <string>

using namespace std;

// Structure to represent a student
struct Student {
    int rollNo;
    string name;
    float sgpa;
};

// Function to perform insertion sort on the student database based on names
void insertionSort(Student students[], int n) {
    for (int i = 1; i < n; ++i) {
        Student key = students[i];
        int j = i - 1;

        // Move elements of students[0..i-1] that are greater than key.name to one position ahead of
their current position
        while (j >= 0 && students[j].name > key.name) {
            students[j + 1] = students[j];
            j = j - 1;
        }

        students[j + 1] = key;
    }
}

// Function to print the sorted list
void printSortedStudents(const Student students[], int n) {
    cout << "Sorted List of Students (by Name):" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Roll No: " << students[i].rollNo << ", Name: " << students[i].name << ", SGPA: " <<
students[i].sgpa << endl;
    }
}

int main() {
    const int maxSize = 100;

    // Input: Number of students in the database
    int n;
    cout << "Enter the number of students: ";
    cin >> n;

    // Input: Student information
    Student students[maxSize];
    cout << "Enter the student information:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Student " << i + 1 << " -" << endl;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Name: ";
        cin.ignore(); // To clear the newline character from the buffer
        getline(cin, students[i].name);
        cout << "SGPA: ";
        cin >> students[i].sgpa;
        cout << endl;
    }

    // Perform insertion sort to arrange students in alphabetical order of names
    insertionSort(students, n);

    // Print the sorted list
    printSortedStudents(students, n);

    return 0;
}
```

ASSIGNMENT NO.3 (USE QUICK SORT)

```
#include <iostream>
#include <string>

using namespace std;

// Structure to represent a student
struct Student {
    int rollNo;
    string name;
    float sgpa;
};

// Function to partition the array for quicksort
int partition(Student students[], int low, int high) {
    float pivot = students[high].sgpa;
    int i = low - 1;
    for (int j = low; j < high; ++j) {
        if (students[j].sgpa >= pivot) {
            i++;
            swap(students[i], students[j]);
        }
    }

    swap(students[i + 1], students[high]);
    return i + 1;
}

// Function to perform quicksort on the student database based on SGPA
void quickSort(Student students[], int low, int high) {
    if (low < high) {
        int partitionIndex = partition(students, low, high);

        quickSort(students, low, partitionIndex - 1);
        quickSort(students, partitionIndex + 1, high);
    }
}

// Function to print the top N students
void printTopStudents(const Student students[], int n) {
    cout << "Top " << n << " Students:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Roll No: " << students[i].rollNo << ", Name: " << students[i].name << ", SGPA: " <<
students[i].sgpa << endl;
    }
}

int main() {
    const int maxSize = 100;

    // Input: Number of students in the database
    int n;
    cout << "Enter the number of students: ";
    cin >> n;

    // Input: Student information
    Student students[maxSize];
    cout << "Enter the student information:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Student " << i + 1 << " -" << endl;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Name: ";
        cin.ignore(); // To clear the newline character from the buffer
        getline(cin, students[i].name);
        cout << "SGPA: ";
        cin >> students[i].sgpa;
        cout << endl;
    }

    // Perform quicksort to arrange students in descending order of SGPA
    quickSort(students, 0, n - 1);

    // Print the top 10 students
    int topN = min(10, n);
    printTopStudents(students, topN);

    return 0;
}
```

ASSIGNMENT NO.4:

```
#include <iostream>

#include <string>

#include <vector>


using namespace std;


// Structure to represent a student

struct Student {

    int rollNo;

    string name;

    float sgpa;

};


// Function to search students by SGPA

void searchStudentsBySGPA(const Student students[], int n, float targetSGPA) {

    vector<Student> matchingStudents;


    // Search for students with the target SGPA

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

        if (students[i].sgpa == targetSGPA) {

            matchingStudents.push_back(students[i]);

        }

    }


    // Print the list of matching students

    if (!matchingStudents.empty()) {

        cout << "Students with SGPA " << targetSGPA << ":" << endl;

        for (const Student& student : matchingStudents) {

            cout << "Roll No: " << student.rollNo << ", Name: " << student.name << ", SGPA: " <<
student.sgpa << endl;
```

```
        }
    } else {
        cout << "No students found with SGPA " << targetSGPA << endl;
    }
}

int main() {
    const int maxSize = 100;

    // Input: Number of students in the database
    int n;
    cout << "Enter the number of students: ";
    cin >> n;

    // Input: Student information
    Student students[maxSize];
    cout << "Enter the student information:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Student " << i + 1 << " -" << endl;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Name: ";
        cin.ignore(); // To clear the newline character from the buffer
        getline(cin, students[i].name);
        cout << "SGPA: ";
        cin >> students[i].sgpa;
        cout << endl;
    }

    // Input: SGPA to search
    float targetSGPA;
    cout << "Enter the SGPA to search: ";
    cin >> targetSGPA;

    // Search and print students with the given SGPA
    searchStudentsBySGPA(students, n, targetSGPA);

    return 0;
}
```

ASSIGNMENT NO.5 (USING BINARY SEARCH WITHOUT RECURSION):

```
#include <iostream>
#include <string>

using namespace std;

// Structure to represent a student
struct Student {
    int rollNo;
    string name;
    float sgpa;
};

// Function to perform binary search on the student database based on names
int binarySearch(const Student students[], int n, const string& targetName) {
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (students[mid].name == targetName) {
            return mid; // Student found
        } else if (students[mid].name < targetName) {
            low = mid + 1; // Discard left half
        } else {
            high = mid - 1; // Discard right half
        }
    }

    return -1; // Student not found
}

int main() {
    const int maxSize = 100;

    // Input: Number of students in the database
    int n;
    cout << "Enter the number of students: ";
    cin >> n;

    // Input: Student information (assuming the list is sorted by name)
    Student students[maxSize];
    cout << "Enter the student information (sorted by name):" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Student " << i + 1 << " -" << endl;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Name: ";
        cin.ignore(); // To clear the newline character from the buffer
        getline(cin, students[i].name);
        cout << "SGPA: ";
        cin >> students[i].sgpa;
        cout << endl;
    }

    // Input: Name to search
    string targetName;
    cout << "Enter the name to search: ";
    cin.ignore(); // To clear the newline character from the buffer
    getline(cin, targetName);

    // Perform binary search to find the student by name
    int result = binarySearch(students, n, targetName);

    // Print the result
    if (result != -1) {
        cout << "Student found - Roll No: " << students[result].rollNo << ", Name: " <<
students[result].name << ", SGPA: " << students[result].sgpa << endl;
    } else {
        cout << "Student with name '" << targetName << "' not found." << endl;
    }

    return 0;
}
```