ASSIGNMENT 10

1. You are given a text file, named “students.txt” that contains students’ records. Each Line contains information of a single student in the form of .

A. Read the records from the file into an array of structures.

B. Three Options will turn up: (1) Bubble Sort, (2) Binary Search, and (3) Quit. In the following we describe what your C/C++ program will do on Selecting the options.

(1) Bubble Sort: Sorts the records based on Student Name. If more than One students has the same name, then sort them on their roll no.

(2) Binary Search: Given a student name, the function will return all the Student records having the Student name.

(3) Quit: Exit the program.

Code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_STUDENTS 100
#define NAME_SIZE 50
#define DEPT_SIZE 30
typedef struct
{
    char name[NAME_SIZE];
    int rollNo;
    char department[DEPT_SIZE];
} Student;

int readFromFile(Student students[], const char *filename)
{
    FILE *file = fopen(filename, "r");
    if (!file)
    {
        printf("Error: Could not open file.\n");
        return 0;
    }
```

```
    int count = 0;
    while (fscanf(file, "%s %d %s", students[count].name,
                  &students[count].rollNo, students[count].department) == 3)
    {
        count++;
        if (count >= MAX_STUDENTS)
            break;
    }
    fclose(file);
    return count;
}

void bubbleSort(Student students[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (strcmp(students[j].name, students[j + 1].name) > 0 ||
                (strcmp(students[j].name, students[j + 1].name) == 0 &&
                 students[j].rollNo > students[j + 1].rollNo))
            {
                Student temp = students[j];
                students[j] = students[j + 1];
                students[j + 1] = temp;
            }
        }
    }
}

void binarySearch(Student students[], int n, const char *targetName)
{
    int found = 0;
    for (int i = 0; i < n; i++)
    {
        if (strcmp(students[i].name, targetName) == 0)
        {
            printf("Found: %s %d %s\n", students[i].name, students[i].rollNo,
                   students[i].department);
            found = 1;
        }
    }
    if (!found)
    {
        printf("No records found for student name: %s\n", targetName);
    }
}
```

```c
int main()
{
    Student students[MAX_STUDENTS];
    int studentCount = readFromFile(students, "students.txt");

    if (studentCount == 0)
    {
        return 1;
    }

    int choice;
    do
    {
        printf("Choose an option:\n1. Bubble Sort\n2. Binary Search\n3.Quit\n");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            bubbleSort(students, studentCount);
            printf("Records after sorting:\n");
            for (int i = 0; i < studentCount; i++)
            {
                printf("%s %d %s\n", students[i].name, students[i].rollNo,
                       students[i].department);
            }
            break;
        case 2:
        {
            char name[NAME_SIZE];
            printf("Enter the name to search: ");
            scanf("%s", name);
            binarySearch(students, studentCount, name);
            break;
        }
        case 3:
            printf("Exiting program...\n");
            break;
        default:
            printf("Invalid option. Try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

```
1
2    JOHN 101 CSE
3    MARY 102 ECE
4    PETER 103 ME
5    ALICE 104 IT
6    JOHN 105 EEE
7    KATE 106 CSE
```

OUTPUT :-

```
Choose an option:
1. Bubble Sort
2. Binary Search
3.Quit
1
Records after sorting:
ALICE 104 IT
JOHN 101 CSE
JOHN 105 EEE
KATE 106 CSE
MARY 102 ECE
PETER 103 ME
Choose an option:
1. Bubble Sort
2. Binary Search
3.Quit
2
Enter the name to search: 101
No records found for student name: 101
Choose an option:
1. Bubble Sort
2. Binary Search
3.Quit
2
Enter the name to search: JOHN
Found: JOHN 101 CSE
Found: JOHN 105 EEE
Choose an option:
1. Bubble Sort
2. Binary Search
3.Quit
3
Exiting program...
```

2.Let A[n] be an array of n distinct integers. If I < j and A[i] > A[j], then the pair (I, j) Is called an inversion of A. Write a C/C++ program that determines the number of Inversions in any permutation on n elements in O(n lg n) worst-case time. (Hint: Modify merge sort)

Code:

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

int count = 0;
void merge(int arr[], int low, int mid, int high);

void mergesort(int arr[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergesort(arr, low, mid);
        mergesort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

void merge(int arr[], int low, int mid, int high) {
    int temp[high - low + 1];
    int left = low, right = mid + 1, k = 0;
    // int count=0;

    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp[k] = arr[left];
            left++;
        } else {
            temp[k] = arr[right];
            right++;
            count+=(mid-left+1) ;

        }
        k++;
    }

    while (left <= mid) {
```

```
        temp[k] = arr[left];
        left++;
        k++;
    }

    while (right <= high) {

        temp[k] = arr[right];
        right++;
        k++;
    }

    for (int i = low, j = 0; i <= high; i++, j++) {
        arr[i] = temp[j];
    }
    // printf(" count variable: %d \n",count);
}

void printarr(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter The Number Of Elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter The Elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergesort(arr, 0, n - 1);
    printf("Sorted Array: ");
    printarr(arr, n);

    printf("Total Number Of Inversion: %d ",count);
    return 0;
}
```

OUTPUT :

```
Enter The Number Of Elements:
4
Enter The Elements:
4
1
3
2
Sorted Array: 1 2 3 4
Total Number Of Inversion: 4
```