Experiment 13

Aim:

To implement a menu-driven program to perform sorting using Selection Sort, Bubble Sort, and Insertion Sort.

Theory:

- **Selection Sort:**
  - This algorithm divides the array into a sorted and unsorted part. It repeatedly selects the smallest (or largest) element from the unsorted part and moves it to the sorted part.
  - **Time Complexity:** $O(n^2)$
- **Bubble Sort:**
  - This algorithm repeatedly swaps adjacent elements if they are in the wrong order. After each pass through the array, the largest element "bubbles" to its correct position.
  - **Time Complexity:** $O(n^2)$
- **Insertion Sort:**
  - Insertion sort builds the sorted array one item at a time. It picks each element from the unsorted array and inserts it into its correct position in the sorted part.
  - **Time Complexity:** $O(n^2)$

Algorithm Steps:

1. **Selection Sort:**
   - For each position in the array, find the smallest element from the unsorted portion and swap it with the current position.
2. **Bubble Sort:**
   - Compare adjacent elements and swap them if they are in the wrong order. Repeat the process until the array is sorted.
3. **Insertion Sort:**
   - Take each element and compare it with elements in the sorted portion. Insert the element at the correct position.

C Code for Menu-driven Sorting

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

```
// Function for selection sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the first element
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

// Function for bubble sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap the adjacent elements if they are in wrong
order
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function for insertion sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
```

```
        // Move elements that are greater than key to one position
ahead
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Driver code: Menu-driven program for sorting
int main() {
    int n, choice;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    while (1) {
        printf("\nSorting Menu:\n");
        printf("1. Selection Sort\n");
        printf("2. Bubble Sort\n");
        printf("3. Insertion Sort\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);

        // Create a copy of the original array to avoid modifying it
after sorting
        int tempArr[n];
        for (int i = 0; i < n; i++) {
            tempArr[i] = arr[i];
        }

        switch (choice) {
            case 1:
                selectionSort(tempArr, n);
                printf("Array after Selection Sort:\n");
                printArray(tempArr, n);
                break;
            case 2:
                bubbleSort(tempArr, n);
                printf("Array after Bubble Sort:\n");
                printArray(tempArr, n);
                break;
            case 3:
                insertionSort(tempArr, n);
                printf("Array after Insertion Sort:\n");
                printArray(tempArr, n);
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Explanation of Code:

1. **Selection Sort:**
   The `selectionSort()` function selects the smallest element from the unsorted portion and swaps it with the current element.
2. **Bubble Sort:**
   The `bubbleSort()` function repeatedly swaps adjacent elements that are in the wrong order until the array is sorted.
3. **Insertion Sort:**
   The `insertionSort()` function builds the sorted portion of the array by inserting each element into its correct position.
4. **Menu:**
   The program provides a menu to the user to choose between selection sort, bubble sort, and insertion sort. The user can input their choice and the program performs the corresponding sort and prints the sorted array.

Sample Input and Output:

```
Enter the number of elements in the array: 5
Enter the elements of the array:
64 25 12 22 11

Sorting Menu:
1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Exit
Enter your choice: 1
Array after Selection Sort:
11 12 22 25 64

Sorting Menu:
1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Exit
Enter your choice: 2
Array after Bubble Sort:
11 12 22 25 64

Sorting Menu:
1. Selection Sort
```

```
2. Bubble Sort
3. Insertion Sort
4. Exit
Enter your choice: 3
Array after Insertion Sort:
11 12 22 25 64

Sorting Menu:
1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Exit
Enter your choice: 4
Exiting...
```

Conclusion:

In this practical, we successfully implemented a menu-driven program to sort an array using Selection Sort, Bubble Sort, and Insertion Sort. Each of these algorithms has its own use case and time complexity, and the program allows the user to choose which sorting method they want to apply. The use of a menu simplifies the interaction with the sorting algorithms.