Data Structures : Algorithms and Applications(Lab)

EXPERIMENT No: 13

AIM: Implementation of Menu driven Selection sort, Bubble sort, Insertion sort.

THEORY:

Bubble Sort, Selection Sort, and Insertion Sort are simple sorting algorithms that are commonly used to sort small datasets or as building blocks for more complex sorting algorithms. Here's a comparison of the three algorithms:

Bubble Sort:
Time complexity: O(n^2) in the worst and average cases, O(n) in the best case (when the input array is already sorted)
Space complexity: O(1)
Basic idea: Iterate through the array repeatedly, comparing adjacent pairs of elements and swapping them if they are in the wrong order. Repeat until the array is fully sorted.

Selection Sort:
Time complexity: O(n^2) in all cases (worst, average, and best)
Space complexity: O(1)
Basic idea: Find the minimum element in the unsorted portion of the array and swap it with the first unsorted element. Repeat until the array is fully sorted.

Insertion Sort:
Time complexity: O(n^2) in the worst and average cases, O(n) in the best case (when the input array is already sorted)
Space complexity: O(1)
Basic idea: Build up a sorted subarray from left to right by inserting each new element into its correct position in the subarray. Repeat until the array is fully sorted.

Comparison:

- Bubble Sort and Selection Sort have the same worst-case time complexity of O(n^2), while Insertion Sort is slightly better with an average-case time complexity of O(n^2).
- Insertion Sort has the best-case time complexity of O(n) when the input array is already sorted, which is not possible for Bubble Sort and Selection Sort.
- Selection Sort and Insertion Sort both have the same space complexity of O(1), while Bubble Sort also has a space complexity of O(1).
- Bubble Sort and Insertion Sort are stable sorting algorithms, meaning that they preserve the relative order of equal elements in the sorted array, while Selection Sort is not stable.

- In terms of performance, Insertion Sort tends to perform better than Bubble Sort and Selection Sort for small datasets, while Bubble Sort and Selection Sort may perform better than Insertion Sort for larger datasets or datasets that are partially sorted.

Overall, each algorithm has its own advantages and disadvantages, and the choice of which algorithm to use depends on the specific requirements of the problem at hand.

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

void display(int a[], int n);
void bubble_sort(int a[], int n);
void selection_sort(int a[], int n);
void insertion_sort(int a[], int n);

int main() {
  int n, choice, i;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  int arr[n];
  for (i = 0; i < n; i++) {
    printf("Enter %d Element: ", i + 1);
    scanf("%d", &arr[i]);
  }

  while (1) {
    printf("\nSelect option for Sorting:\n");
    printf("1. Bubble Sort\n");
    printf("2. Selection Sort\n");
    printf("3. Insertion Sort\n");
    printf("4. Display Array\n");
    printf("5. Exit the Program\n");
    printf("Enter your Choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        bubble_sort(arr, n);
        break;
      case 2:
        selection_sort(arr, n);
        break;
```

```
            case 3:
                insertion_sort(arr, n);
                break;
            case 4:
                display(arr, n);
                break;
            case 5:
                return 0;
            default:
                printf("\nSelect option 1-5 ---- \n");
        }
    }
    return 0;
}

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

void bubble_sort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    printf("After Bubble Sort: ");
    display(arr, n);
}

void selection_sort(int arr[], int n) {
    int i, j, min_idx, temp;
    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Swap the found minimum element with the first element
        if (min_idx != i) {
            temp = arr[i];
            arr[i] = arr[min_idx];
            arr[min_idx] = temp;
        }
    }
    printf("After Selection Sort: ");
    display(arr, n);
}

void insertion_sort(int arr[], int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
    printf("After Insertion Sort: ");
    display(arr, n);
}
```

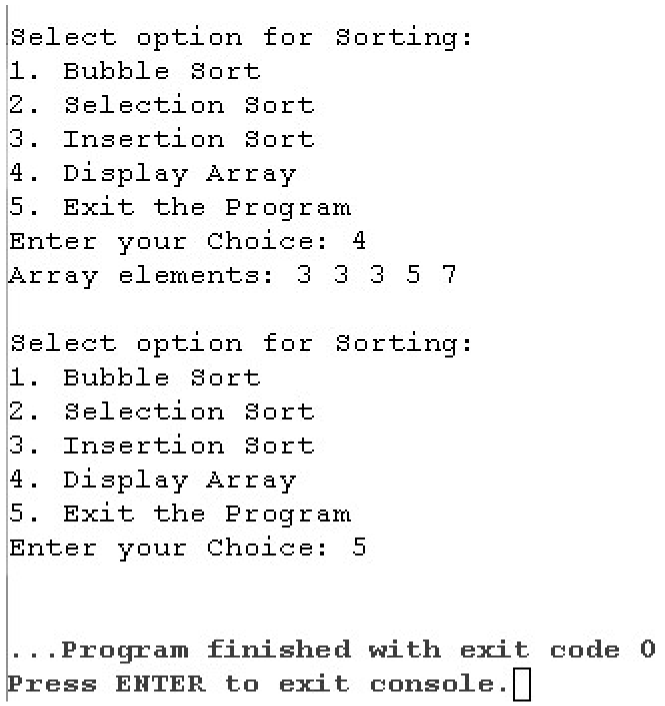
OUTPUT:

```
Enter number of elements: 5
Enter 1 Element: 3
Enter 2 Element: 5
Enter 3 Element: 7
Enter 4 Element: 3
Enter 5 Element: 3

Select option for Sorting:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array
5. Exit the Program
Enter your Choice: 1
After Bubble Sort: Array elements: 3 3 3 5 7

Select option for Sorting:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array
5. Exit the Program
Enter your Choice: 2
After Selection Sort: Array elements: 3 3 3 5 7

Select option for Sorting:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array
5. Exit the Program
Enter your Choice: 3
After Insertion Sort: Array elements: 3 3 3 5 7
```



```
Select option for Sorting:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array
5. Exit the Program
Enter your Choice: 4
Array elements: 3 3 3 5 7

Select option for Sorting:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array
5. Exit the Program
Enter your Choice: 5


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:

The program implements Bubble Sort, Selection Sort, and Insertion Sort, allowing users to sort an array of integers interactively. Each algorithm demonstrates unique sorting approaches, emphasizing their varying efficiency, particularly for small datasets.