Symbiosis Institute of Technology

Faculty of Engineering
CSE - Academic Year 2023-24
Data Structures Lab Batch 2022-26

Lab Assignment 2	
Name	Manan Bhimjiyani
PRN no	22070122107
Batch	2022-26
Class	CS B1
Academic year & semester	2023-24
Date of submission	31/08/2023
Title of Assignment	Implement following sorting techniques and find the time complexity:: i. Bubble ii. Selection iii. Insertion
Theory	A table comparing the best case, average case, and worst case time complexities of the given sorting algorithms: Bubble Sort, Insertion Sort, and Selection Sort.

Algorithm	Best Case	Avg Case	Worst Case
Bubble sort	O(n)	O(n^2)	O(n^2)
Insertion sort	O(n)	O(n^2)	O(n^2)
Selection sort	O(n^2)	O(n^2)	O(n^2)

Best case and Worst case time complexities of bubble sort, Insertion sort and selection sort.

Bubble Sort:

Best Case: O(n) - The array is already sorted, and no swaps are needed.

Worst Case: O(n^2) - The array is sorted in reverse order, and each element needs to be swapped with every other element.

Insertion Sort:

Best Case: O(n) - The array is already sorted, and each new element can be inserted in constant time.

Worst Case: O(n^2) - The array is sorted in reverse order, and each new element requires shifting all previous elements.

	<u>Selection Sort:</u> **Best Case:** O(n^2) - The algorithm doesn't have any optimization for the initial ordering of the array. **Worst Case**: O(n^2) - Regardless of the input ordering, the algorithm requires n iterations for finding the minimum element in each pass.
Source Code:	Bubble Sort: Selection Sort:

```c
#include <stdio.h>

void bs(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]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {69,26,41,34,96,40};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    bs(arr, n);

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

    return 0;
}
```

```c
#include <stdio.h>
void ss(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int mini = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[mini]) {
                mini = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[mini];
        arr[mini] = temp;
    }
}
int main() {
    int arr[] = {69,11,96,54,88};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    ss(arr, n);
    printf("\nSorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Insertion Sort:

```c
#include <stdio.h>
void is(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main() {
    int arr[] = {69,67,54,31,69};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    is(arr, n);
    printf("\nSorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output :	Bubble Sort: Original array: 69 26 41 34 96 40 Sorted array: 26 34 40 41 69 96 ...Program finished with exit code 0 Press ENTER to exit console. Selection Sort: Original array: 69 11 96 54 88 Sorted array: 11 54 69 88 96 ...Program finished with exit code 0 Press ENTER to exit console. Insertion Sort: Original array: 69 67 54 31 69 Sorted array: 31 54 67 69 69 ...Program finished with exit code 0 Press ENTER to exit console.
Conclusion:	Thus, we have studied different sorting algorithms and their time complexities