Symbiosis Institute of Technology

Faculty of Engineering

CSE- Academic Year 2024-25

Data Structures – Lab Batch 2023-27

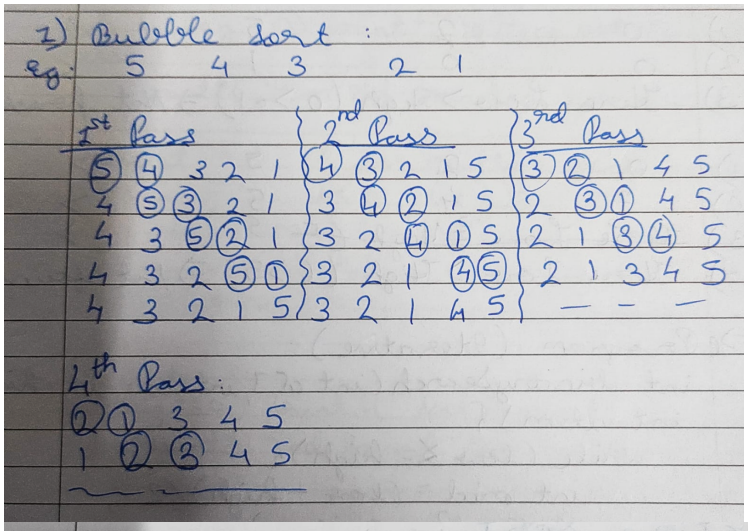
Lab Assignment No:- 1

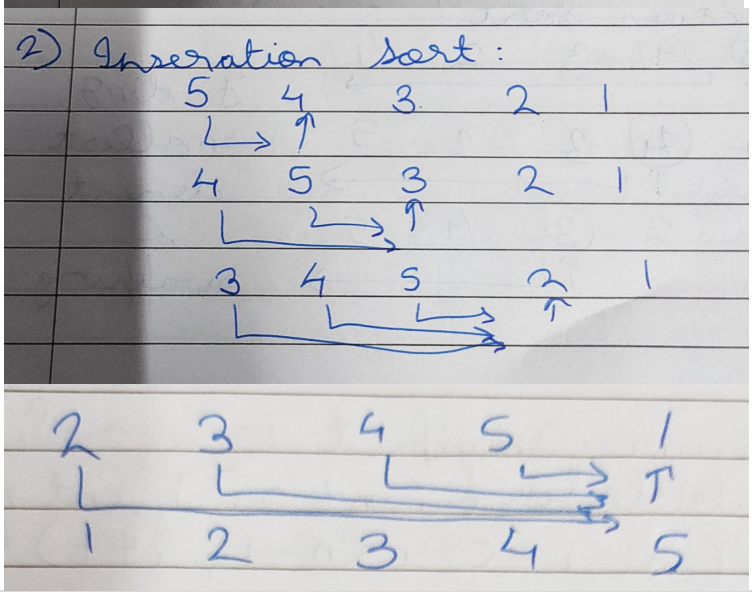
Name of Student	Faheemuddin Sayyed
PRN No.	23070122196
Batch	23-27
Class	CSE C-1
Academic Year & Semester	SY 24-25
Date of Performance	1/08/24
Title of Assignment:	A. Implement following sorting techniques and find the time complexity:: i. Bubble ii. Selection iii. Insertion
Theory Questions:	1. Prepare table for following 10 different sorting algorithms for their best case, average case and worst case time complexities. 2. Solve examples of bubble sort, insertion sort and selection sort. Show all passes. 3. Write real world applications of bubble sort, insertion sort and selection sort. 4. How we can optimize bubble sort. Answer 1:

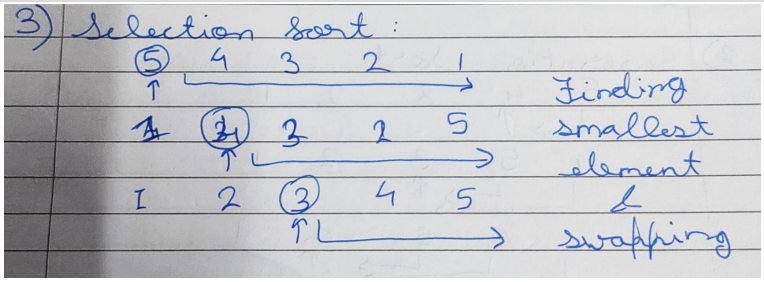
Sorting Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Merge Sort	O(n log n)	O(n log n)	O(n log n)

Quick Sort	O(n log n)	O(n log n)	O(n^2)
Heap Sort	O(n log n)	O(n log n)	O(n log n)
Count Sort	O(n+k)	O(n+k)	O(n+k)
Radix Sort	O(nk)	O(nk)	O(nk)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)
Tim Sort	O(n)	O(n log n)	O(n log n)

Answer 2:







Answer 3:

Bubble Sort:

1. **Teaching**: Simplifies the concept of sorting for beginners.
2. **Small Data Sets**: Works well with small, mostly sorted datasets.
3. **Detecting Sorted Data**: Efficient for nearly sorted data.

Insertion Sort:

1. **Online Sorting**: Sorts data as it arrives.
2. **Small Arrays**: Used in hybrid algorithms for small arrays.
3. **Adaptive Sorting**: Effective for nearly sorted data.

Selection Sort:

1. **Limited Writes**: Fewer memory writes, useful where writes are costly.
2. **Small Lists**: Simple implementation for small datasets.
3. **Fixed Size Sorting**: Selecting top elements from small lists.

Answer 4:
Bubble sort can be optimized by using a flag to detect if no swaps were made during an iteration. If no swaps occur, the array is already sorted, and the algorithm can terminate early, reducing unnecessary comparisons.

Source Code/Algorithm/Flow Chart:

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

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void bubbleSortAdaptive(int a[], int n){
    int flag;

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

```
            if(a[j] > a[j+1]){
                swap(&a[j], &a[j+1]);
                flag = 1;
            }
        }

        if(flag == 0) break;
    }
}

void selectionSort(int a[], int n){
    int minIndex;

    for(int i = 0; i < n - 1; i++){
        minIndex = i;
        for(int j = i; j < n; j++){
            if(a[j] < a[minIndex])
                minIndex = j;
        }

        if(minIndex != i)
            swap(&a[i], &a[minIndex]);
    }
}

void insertionSort(int a[], int n){
    int i, j, x;

    for(i = 1; i < n; i++){
        j = i - 1;
        x = a[i];
        while(j > -1 && a[j] > x){
            a[j+1] = a[j];
            j--;
        }

        a[j+1] = x;
    }
}

int main(){
    int *a;
    int n;

    printf("\nEnter size of array: ");
    scanf("%d", &n);
    a = (int *)malloc(n * sizeof(int));
```

```c
    printf("\nEnter elements of the array:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    // bubbleSortAdaptive(a, n);
    // selectionSort(a, n);
    insertionSort(a, n);

    printf("\nSorted Array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

Output Screenshots

```
Enter size of array: 5

Enter elements of the array:
5 4 3 2 1

Sorted Array:
1 2 3 4 5
fahee@Faheems-MacBook-Pro Data Structures %
```

Practice questions

1. Implement Optimized bubble sort
2. o/p screenshot

Conclusion

Thus we have studied different sorting algorithms and their time complexities.