22BCP469D – Aryan Randeriya

20CP209P – Design & Analysis of Algorithms Lab

Practical 1

a) Insertion Sort

Implementation:

```c
#include <stdio.h>

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{

    int n; // The number of elements

    int arr[100], j, pivot;

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

    // Loop for user input for elements of the array
    for (int i = 0; i < n; i++)
    {
        printf("Enter element at index %d: ", i);
        scanf("%d", &arr[i]);
    }

    //   Insertion Sort
    for (int i = 1; i < n; i++)
    {
        pivot = arr[i]; // Set pivot as the 2nd element and consider the first
element sorted
        j = i - 1;
```

```
        while (j >= 0 && arr[j] > pivot) // loop from j=i-1 until an element
larger than the pivot is found
        {
            arr[j + 1] = arr[j]; // Next element which is the larger (j+1) is
assigned the value of the previous
            j = j - 1;           // Decrement the index
        }
        arr[j + 1] = pivot; // The
    }

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

    return 0;
}
```

Output:

```
Enter the number of elements: 5
Enter element at index 0: 5
Enter element at index 1: 4
Enter element at index 2: 3
Enter element at index 3: 2
Enter element at index 4: 1
Elements of the array: 1 2 3 4 5
```

Time Complexity Analysis:

- **Best Case –**
  - **O(n)**

- In the best-case scenario, the array is sorted, and the outer loop will always run n-1 times regardless if any shifting is done or not. Everytime we try to enter the inner loop the condition of arr[j] > pivot is always false, meaning the pivot is always smaller.
- Total comparison operations = n-1 = O(n)
- Total shifting operations = 0 = O(1)
- **Total Best Case Time Complexity = O(n) + O(1) = O(n)**

- **Worst Case –**
  - **$O(n^2)$**

- In the worst-case scenario, the array is reversely sorted, and the outer loop will run n-1 times and at each iteration, the inner loop will run I times because arr[j]> temp will be always true. To insert ith value at the correct position, we shift all the values from j=i-1 to 0 to one
- So at the ith iteration of the outer loop: Count of comparison operations=i and count of shifting operation=i.
- Total comparison operations =1+2+3+...+n-2+n-1=n(n-1)/2=$O(n^2)$.
- Total shifting operations = 1+2+3+...+n-2+n-1=n(n-1)/2=$O(n^2)$.
- **Total Worst Case Time Complexity = $O(n^2)+O(n^2)=O(n^2)$**

- **Average Case –**
  - **$O(n^2)$**

- When the input array is randomly ordered and each element must be moved to its correct position by shifting all the larger elements to the right. So, the average case time complexity is considered same as the worst case time complexity that is O(n^2)

Selection Sort

Implementation:

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

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{

    int n; // The number of elements
    int arr[100], min;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Loop for user input for elements of the array
    for (int i = 0; i < n; i++)
    {
        printf("Enter element at index %d: ", i);
        scanf("%d", &arr[i]);
    }

    for (int i = 0; i < n; i++) // Iterate over all the elements once to set the minimum value
    {
        min = i;                    // Set the first element as the minimum
        for (int j = i; j < n; j++) // Loop until j = i -> n
        {
            if (arr[j] < arr[min]) // Compare if any element is smaller than the minimum
            {
                swap(&arr[j], &arr[min]); // If any element is smaller than minimum then swap them
            }
        }
    }
```

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

    return 0;
}
```

Output:

```
Enter the number of elements: 7
Enter element at index 0: 7
Enter element at index 1: 6
Enter element at index 2: 5
Enter element at index 3: 4
Enter element at index 4: 3
Enter element at index 5: 2
Enter element at index 6: 1
Elements of the array: 1 2 3 4 5 6 7
```

Time Complexity Analysis:

- **Best Case –**
  - **$O(n^2)$**

- When the input is already sorted, the comparison arr[j]<arr[min] becomes false every time, and the value of min will not get updated. Therefore, the sorted array is the best-case input for the selection sort.
- Total comparison operations = Count of nested loop iterations = O(n^2)
- The total count of swap operations = O(n).
- The total update operation of minindex = O(1).
- The time complexity of selection sort in the best case = O(n^2) +
- O(n) + O(1) = O(n^2).
- **Total Best Case Time Complexity = O(n^2) + O(n) + O(1) = O(n)**

- **Worst Case –**
  - **$O(n^2)$**

- When the input is sorted in decreasing order, the comparison arr[j]<arr[min] becomes true every time, and the value of min will get updated every time. So the reverse-sorted array is the worst-case input for selection sort.
- The count of comparison operations = Total count of nested loop iterations = O(n^2).
- We perform one swap on each iteration of the outer loop. The total count of swapping operations = n* O(1) =O(n).
- The total update operation of min= Total count of loop operations = O(n^2).
- **Total Worst Case Time Complexity O(n^2) +O(n) + O(n^2) = O(n^2).**

- **Average Case –**
  - **$O(n^2)$**

- As both best case and worst-case complexity analysis are the same then the average case complexity will also be the same that is O(n^2)