Experiment - 1

AIM To implement following algorithm using array as data structure and analyse time complexity

1) Insertion Sort
2) Bubble Sort
3) Selection Sort

Theory

Bubble Sort

→ Algorithm

```
Bubble Sort (a, n)
for i = 0 to n-1
for j = i to n
 if a [i] > a [j]
      temp = a [j]
      a [j] = a [i]
      a [i] = temp
```

→ Time Complexity Analysis

In bubble sort, n-1 comparison will be done in the 1st pass, n-1 in the second pass, n-2 in third pass and so on

In total number of comparison will be

$(n-1) + (n-2) + (n-3) + \ldots + 3 + 2 + 1$

Sum $= n(n-1)/2$

ie $O(n^2)$

Time complexity of Bubble sort is $O(n^2)$

Case 1 : Best Case

Best case is when array / list is already sorted

$$\text{Sum} = \sum_{i=1}^{n-1} 1 = N - 1 = O(N)$$

Case 2 : Worst Case

Worst case is when array is reverse sorted

$$\text{Sum} = \sum_{i=1}^{n-1} i = 1 + 2 + 3 + \ldots + (N-1) = \frac{(N-1)N}{2} = O(N^2)$$

Case 3 : Average Case

Each element is about half way in order

$$\text{Sum} = \sum_{i=1}^{n-1} \frac{i}{2} = \frac{1}{2}(1 + 2 + 3 + \ldots + (n-1)) = \frac{(n-1)n}{4} = O(n^2)$$

Insertion Sort

→ Algorithm

~~for~~ Insertion Sort (A, n)

for $j = 2$ to n

key = A[j]

// Insert A[j] into the sorted sequence A[1 ... j-1]

$i = j - 1$

```
while i > 0 and A[i] > key
    A[i+1] = A[i]
    i = i - 1
    A[i+1] = key
```

→ Time complexity Analysis

Insertion Sort (A, n)	Cost	times
for $j = 2$ to n	c_1	n
key = A[j]	c_2	$n-1$
i = j − 1	c_4	$n-1$
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i = i − 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_8	$n-1$

Case 1: Best Case

Cost

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 (n-1) + c_8 (n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8) n - (c_2 + c_2 + c_5 + c_8)$$

$$= an + b$$

⇒ Linear function of n

⇒ $O(n)$

Case-2: Worst Case

Cost

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5((n(n+1)/2) - 1) + C_6(n(n-1)/2) + C_7(n(n-1)/2) + C_8(n-1)$$

$$= ((C_5/2 + C_6/2 + C_7/2)n^2 + (C_1 + C_2 + C_4 + C_5/2 + C_6/2 - C_7/2 + C_8)n - (C_2 + C_4 + C_5 + C_8)$$

$$= an^2 + bn + c = O(n^2)$$

Case 3: Average Case

In average case half of the elements are in A[1...j-1] are less than A[j] and half of the elements are greater. On average half of the subarray is checked and t_j is about j/2

Here time complexity is similar to worst case is quadratic

Time complexity of Average Case = $O(n^2)$

Selection Sort

→ Algorithm

```
Selection Sort (A)
  n ← length [A]
  for j ← 1 to n-1
  do smallest ← j
    for i ← j+1 to n
      do if A[i] < A[smallest]
        then smallest ← i
      swap A[j] ↔ A[smallest]
```

→ Complexity Analysis

	Cost	Times
selection Sort (A)		
$n \leftarrow$ length [A]	c_1	1
for $j \leftarrow 1$ to $n-1$	c_2	n
do smallest $\leftarrow j$	c_3	$n-1$
for $i \leftarrow j+1$ to n	c_4	$\sum_{j=1}^{n} (n-j+1)$
do if $A[i] \leftarrow A[$ smallest$]$	c_5	$\sum_{j=1}^{n} (n-j)$
the smallest $\leftarrow i$	0	$\sum_{j=1}^{n} (n-j)$
Swap $A[j] \leftrightarrow A[$smallest$]$	c_6	$n-1$

By Cost

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4 \sum_{j=1}^{n-1} (n-j+1) + c_5 \sum_{j=1}^{n-1} (n-j) + c_6 (n-1)$$

$\Rightarrow (n-1) + (n-2) + (n-3) + \cdots 2+1$

$\Rightarrow n(n-1)/2$

$\Rightarrow O(n^2)$

Case 1: Best case time complexity = $O(n^2)$

Case 2: Worst case time complexity = $O(n^2)$

Case 3: Average case time complexity = $O(n^2)$

Result: Insertion Sort, Bubble Sort, Selection Sort Algorithm implemented successfully.

Code (Insertion Sort)

```cpp
#include <iostream>
#include <chrono>
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;
int main()
{
    int n, i, j, key;
    cout << "Enter the Number of Elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the Elements: ";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    auto start = steady_clock::now();
    for (i = 0; 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;
    }
    auto stop = steady_clock::now();
    cout << "Sorted array is: ";
    for (j = 0; j < n; j++)
    {
        cout << arr[j] << " ";
    }
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nTime taken by function: " << duration.count() << " nano
seconds" << endl;
}
```

Output (Insertion Sort)

Best Case

```
Enter the Number of Elements: 5
Enter the Elements: 1 2 3 4 5
Sorted array is: 1 2 3 4 5
Time taken by function: 1500 nanoseconds
```

Worst Case

```
Enter the Number of Elements: 5
Enter the Elements: 5 4 3 2 1
Sorted array is: 1 2 3 4 5
Time taken by function: 1900 nanoseconds
```

Code (Bubble Sort)

```cpp
#include <iostream>
#include <chrono>
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;
int main()
{
    int i, j, k, n, temp;
    cout << "Enter Number of Elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of array: ";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    auto start = steady_clock::now();
```

```
    for (i = 0; i < n; i++)
    {
        for (j = i; j < n; j++)
        {
            if (arr[i] > arr[j])
            {
                temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
    }
    auto stop = steady_clock::now();
    cout << "Sorted Array is: ";
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nTime taken by function: " << duration.count() << " nanoseconds" <<
 endl;
}
```

Output (Bubble Sort)

Best Case

```
Enter Number of Elements: 5
Enter the elements of array: 1 2 3 4 5
Sorted Array is: 1 2 3 4 5
Time taken by function: 1500 nanoseconds
```

Worst case

```
Enter Number of Elements: 5
Enter the elements of array: 5 4 3 2 1
Sorted Array is: 1 2 3 4 5
Time taken by function: 1700 nanoseconds
```

Code (Selection Sort)

```cpp
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
int main()
{
    int i, j, n, min, temp, k;
    cout << "Enter the Number of Elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements: ";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    auto start = steady_clock::now();
    for (i = 0; i < n - 1; i++)
    {
        temp = i;
        for (j = i + 1; j < n; j++)
        {
            if (arr[temp] > arr[j])
            {
                temp = j;
            }
        }
        min = arr[temp];
        arr[temp] = arr[i];
        arr[i] = min;
    }
    auto stop = steady_clock::now();
    cout << "Sorted Array" << endl;
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nTime taken by function: " << duration.count() << " nanoseconds" <<
 endl;
}
```

Output (Selection Sort)

Best Case

```
Enter the Number of elements: 5
Enter the elements: 1 2 3 4 5
Sorted array: 1 2 3 4 5
Time taken by function: 2700 nanoseconds
```

Worst Case

```
Enter the Number of elements: 5
Enter the elements: 5 4 3 2 1
Sorted array: 1 2 3 4 5
Time taken by function: 2800 nanoseconds
```