Assignment –1

Name: Bijoy Das Gupta

ID: 0112330355

Course: CSE 2116 (DSA 1 LAB)

Section: D

Answer To the Question No. 1:

Using String Selection Sort: selectionSortStr(arr, n, cmp, swaps) ,

```
#include<bits/stdc++.h>
using namespace std;

void selectionSortStr(string arr[], int n, long long &cmp, long long &swaps)
{
   cmp=swaps=0;

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

```
      for(int j=i+1; j<n; j++){
         cmp++;
         if(arr[j] < arr[minIndex])
         {
            minIndex=j;
         }
      }
      if(minIndex != i)
      {
         swap(arr[i], arr[minIndex]);
         swaps++;
      }
   }
}
int main ()
{
   int n;
   cin >> n;
   string arr[10000];

   for(int i=0; i<n; i++)
      cin >> arr[i];
```

```
    long long swaps, cmp;
    selectionSortStr(arr, n, cmp, swaps);

    cout << "Sorted Array: " << endl;
    for(int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Answer To the Question No. 2:

Stability: Both Insertion sort and Merge sort is Stable. But selection sort is not stable.

Time Complexity: **Firstly**, the time complexity of best case is Ω(n) and worst case is O (n2) for Insertion sort.

Secondly, the time complexity of best case is Ω(n2) and worst case is O(n2) for Selection sort.

Finally, the time complexity of best case is Ω (n log n) and worst case is O (n log n) for Merge Sort.

Space Complexity: **Firstly,** the space complexity of the sort is O (1) for Insertion Sort.

Secondly, the space complexity of the sort is O (1) for Selection sort.

Finally, the space complexity of the sort is O(n) for Merge Sort.

Answer To the Question No. 3:

Testing & comparing all three sorts on 1000 random numbers by using random functions,

```
#include <bits/stdc++.h>
#include <cstdlib>
#include <ctime>
using namespace std;

void selectionSortStr(int arrS[], int n, long long &cmp2, long long &swaps)
{
   cmp2 = swaps = 0;

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

```
      {
         cmp2++;
         if (arrS[j] < arrS[minIndex])
         {
            minIndex = j;
         }
      }
      if (minIndex != i)
      {
         swap(arrS[i], arrS[minIndex]);
         swaps++;
      }
   }
}

void insertionSort(int arrI[], int n, long long &cmp1, long long &shifts)
{
   cmp1 = shifts = 0;
   for (int i = 1; i < n; i++)
   {
      int key = arrI[i];
      int j = i - 1;
      while (j >= 0 && (++cmp1 && arrI[j] > key))
```

```
        {
            arrI[j + 1] = arrI[j];
            shifts++;
            j--;
        }
        arrI[j + 1] = key;
        shifts++;
    }
}
long long cmp3 = 0, copyCount = 0;

void merging(int arrM[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
    {
        L[i] = arrM[l + i];
        copyCount++;
    }
    for (int j = 0; j < n2; j++)
```

```
{
   R[j] = arrM[m + 1 + j];
   copyCount++;
}
int i = 0, j = 0, k = l;

while (i < n1 && j < n2)
{
   cmp3++;
   if (L[i] <= R[j])
   {
      arrM[k] = L[i];
      i++;
      k++;
   }
   else
   {
      arrM[k] = R[j];
      j++;
      k++;
   }
}
while (i < n1)
```

```
    {
        arrM[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arrM[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arrM[], int l, int r)
{
    if (l < r)
    {
        int m = (l + r) / 2;
        mergeSort(arrM, l, m);
        mergeSort(arrM, m + 1, r);
        merging(arrM, l, m, r);
    }
}
```

```
int main()
{
   srand(time(0));

   int n = 1000;

   int arrI[10000], arrS[10000], arrM[10000];

   for (int i = 0; i < n; i++)
   {
      int p = rand();
      arrI[i] = p;
      arrS[i] = p;
      arrM[i] = p;
   }

   long long swaps, cmp1, cmp2, shifts;

   insertionSort(arrI, n, cmp1, shifts);
   selectionSortStr(arrS, n, cmp2, swaps);
   mergeSort(arrM, 0, n - 1);

   cout << "Sorted Array: " << endl;
```

```
    for (int i = 0; i < n; i++)
        cout << arrI[i] << " ";

    cout << endl;
    cout << endl;

    cout << "Insertion Sort: Comparisons = " << cmp1 << ", shifts = " << shifts << endl;
    cout << endl;

    cout << "Selection Sort: Cmparisons = " << cmp2 << ", Swaps = " << swaps << endl;
    cout << endl;

    cout << "Merge Sort: Comparisons = " << cmp3 << ", Copy Count = " << copyCount << endl;
    cout << endl;

    return 0;
}
```