K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A3 Roll No.: 16010121051

Experiment No.1

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of selection sort/ Insertion sort

Objective: To analyse performance of sorting methods

CO to be achieved:

CO 1 Analyze the asymptotic running time and space complexity of algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Insertion_sort**
4. **http://www.sorting-algorithms.com/insertion-sort**
5. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html**
6. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**
7. **http://en.wikipedia.org/wiki/Selection_sort**
8. **http://www.sorting-algorithms.com/selection-sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**
10. **http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.html**

Pre Lab/ Prior Concepts:
Data structures, sorting techniques.

Historical Profile:
There are various methods to sort the given list. As the size of input changes, the performance of these strategies tends to differ from each other. In such case, the priori analysis can helps the engineer to choose the best algorithm.

New Concepts to be learned:
Space complexity, time complexity, size of input, order of growth.

Topic: Sorting Algorithms

Theory: Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and- conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller sub problems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

Algorithm Insertion Sort

INSERTION_SORT (*A*,*n*)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR j ← 2 TO length[A]
    DO key ← A[j]
        {Put A[j] into the sorted sequence A[1 . . j − 1]}
           i ← j − 1
           WHILE i > 0 and A[i] > key
               DO A[i +1] ← A[i]
                   i ← i − 1
           A[i + 1] ← key
```

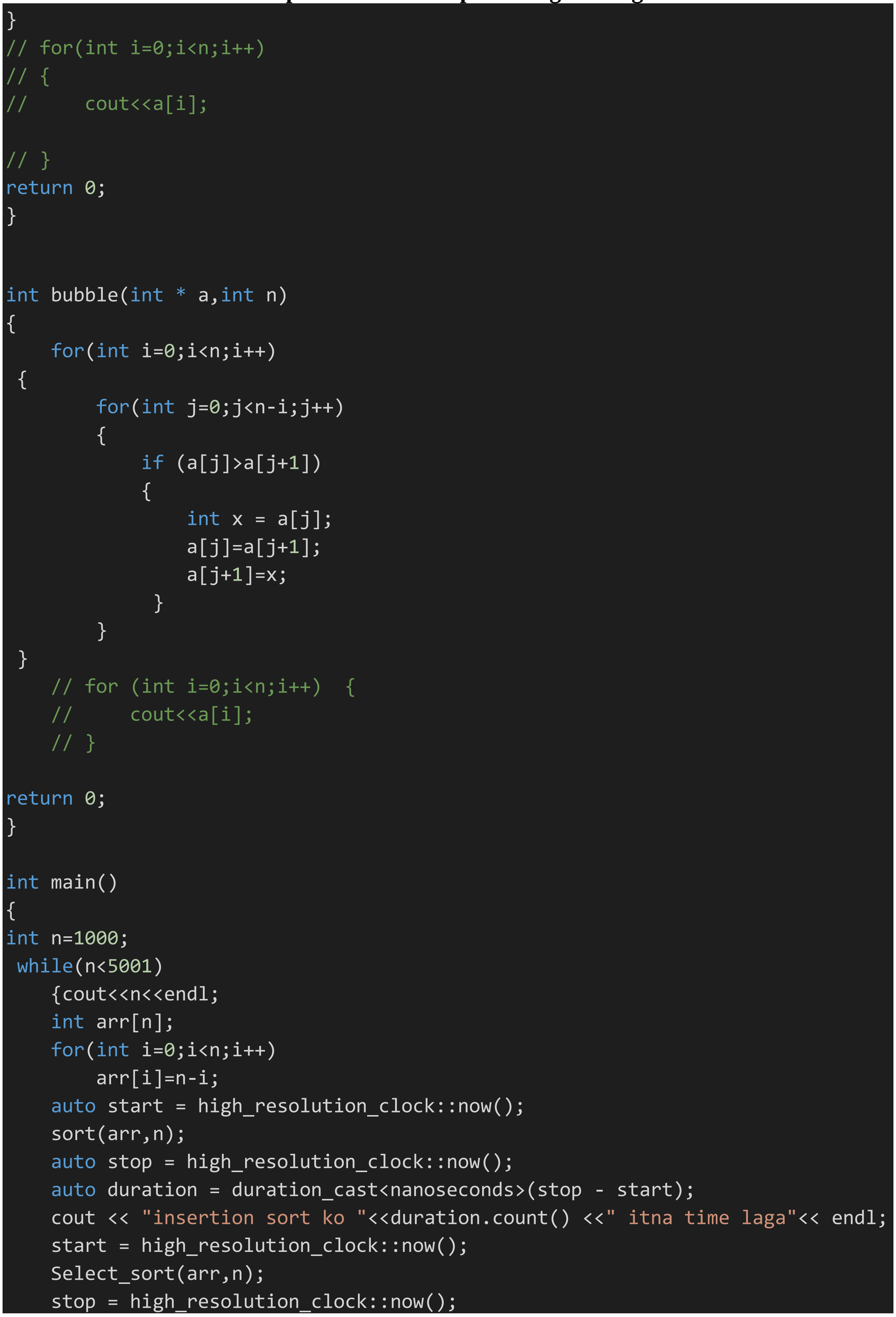
Algorithm Selection Sort

SELECTION_SORT (A,n)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR i ← 1 TO n-1 DO
   min j ← i;
   min x ← A[i]
  FOR j ← i + 1 to n do
     IF A[j] < min x then
        min j ← j
        min x ← A[j]
  A[min j] ← A [i]
   A[i] ← min x
```

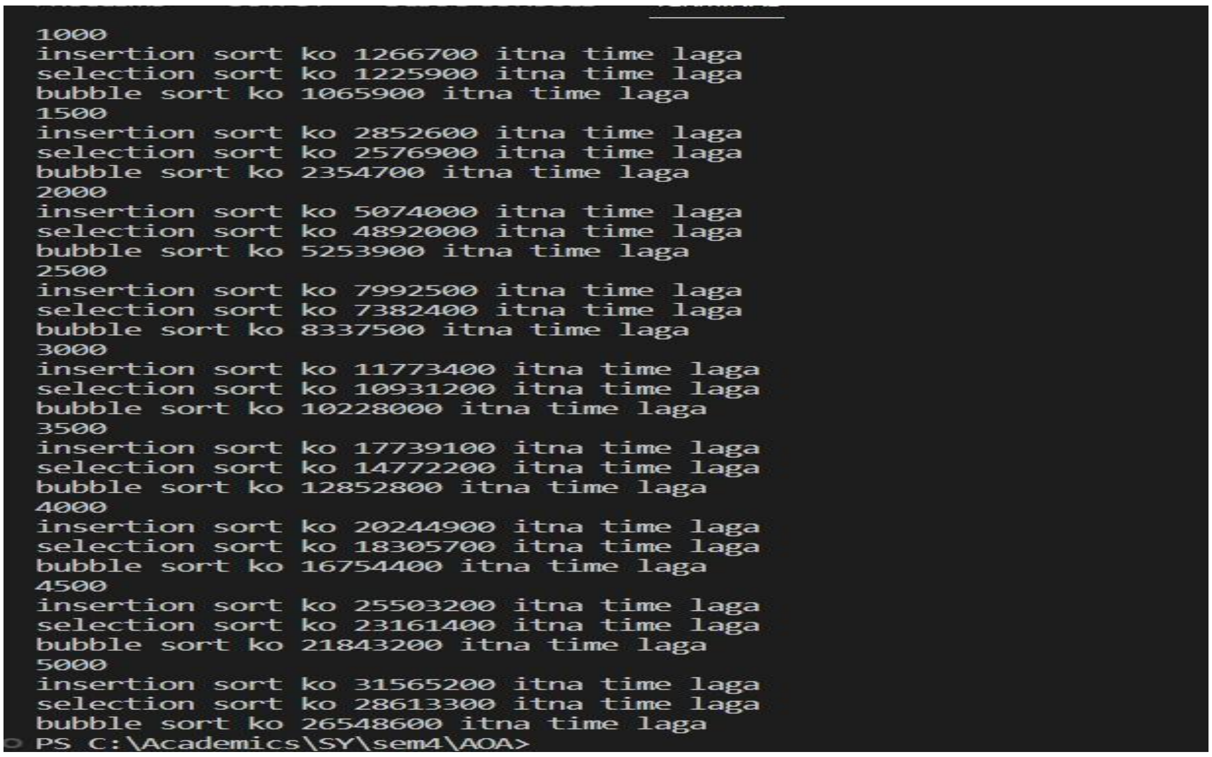
Code:

```
#include<iostream>
#include <chrono>
using namespace std::chrono;
using namespace std;

int sort(int* a,int n)
{
    for(int i=0;i<n;i++)
    {
    int x = a[i];
    int j = i - 1;

    while (x < a[j] && j >= 0) {
      a[j + 1] = a[j];
      --j;
    }
    a[j + 1] = x;
    }

    // for(int i=0;i<n;i++)
    // {
    //     cout<<a[i];
    // }

    return 0;
}

int Select_sort(int* a,int n){
for(int i=0;i<n-1;i++)
{
    int i_min =i;
    for(int j=i+1;j<n;j++)
    {
        if(a[j]<a[i_min])
        {
            i_min=j;
        }
    }
        int temp=a[i];
        a[i]=a[i_min];
        a[i_min]=temp;
```



```
}
// for(int i=0;i<n;i++)
// {
//     cout<<a[i];

// }
return 0;
}


int bubble(int * a,int n)
{
    for(int i=0;i<n;i++)
 {
        for(int j=0;j<n-i;j++)
        {
            if (a[j]>a[j+1])
            {
                int x = a[j];
                a[j]=a[j+1];
                a[j+1]=x;
             }
        }
 }
    // for (int i=0;i<n;i++)  {
    //     cout<<a[i];
    // }

return 0;
}

int main()
{
int n=1000;
 while(n<5001)
    {cout<<n<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
        arr[i]=n-i;
    auto start = high_resolution_clock::now();
    sort(arr,n);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "insertion sort ko "<<duration.count() <<" itna time laga"<< endl;
    start = high_resolution_clock::now();
    Select_sort(arr,n);
    stop = high_resolution_clock::now();
    duration = duration_cast<nanoseconds>(stop - start);
    cout << "selection sort ko "<<duration.count() <<" itna time laga"<< endl;
    start = high_resolution_clock::now();
    bubble(arr,n);
    stop = high_resolution_clock::now();
    duration = duration_cast<nanoseconds>(stop-start);
    cout<<"bubble sort ko "<<duration.count()<<" itna time laga"<<endl;
    n=n+500;
    }


    return 0;
}
```

Output:



```
1000
insertion sort ko 1266700 itna time laga
selection sort ko 1225900 itna time laga
bubble sort ko 1065900 itna time laga
1500
insertion sort ko 2852600 itna time laga
selection sort ko 2576900 itna time laga
bubble sort ko 2354700 itna time laga
2000
insertion sort ko 5074000 itna time laga
selection sort ko 4892000 itna time laga
bubble sort ko 5253900 itna time laga
2500
insertion sort ko 7992500 itna time laga
selection sort ko 7382400 itna time laga
bubble sort ko 8337500 itna time laga
3000
insertion sort ko 11773400 itna time laga
selection sort ko 10931200 itna time laga
bubble sort ko 10228000 itna time laga
3500
insertion sort ko 17739100 itna time laga
selection sort ko 14772200 itna time laga
bubble sort ko 12852800 itna time laga
4000
insertion sort ko 20244900 itna time laga
selection sort ko 18305700 itna time laga
bubble sort ko 16754400 itna time laga
4500
insertion sort ko 25503200 itna time laga
selection sort ko 23161400 itna time laga
bubble sort ko 21843200 itna time laga
5000
insertion sort ko 31565200 itna time laga
selection sort ko 28613300 itna time laga
bubble sort ko 26548600 itna time laga
PS C:\Academics\SY\sem4\AOA>
```

The space complexity of Insertion sort: O(n)

It takes in a total of n + 5 elements of space

Hence O(n)

The space complexity of Selection sort: O(n)

It takes in a total of n + 5 elements of space

Hence O(n)

Time complexity for Insertion sort: O(n^2)

(n-1) + (n(n-1))/2 = (n^2+n+2)/2

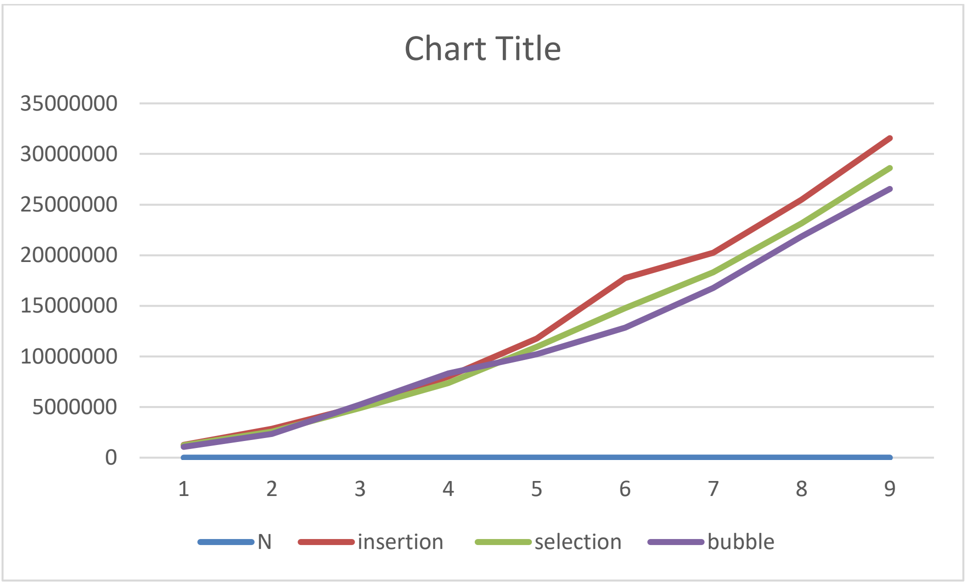
Hence O(n^2)

Time complexity for selection sort: O(n^2)

(n-1) + (n(n-1))/2 = (n^2+n+2)/2

Hence O(n^2)

Graphs for varying input sizes: (Insertion Sort & Selection sort & Bubble sort)



CONCLUSION:

Understood the logic behind insertion sort and selection sort and the analysis of their space and time complexities.