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

Batch: A3	**Roll No.: 16010121045**
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, $1< k \leq n$, 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.

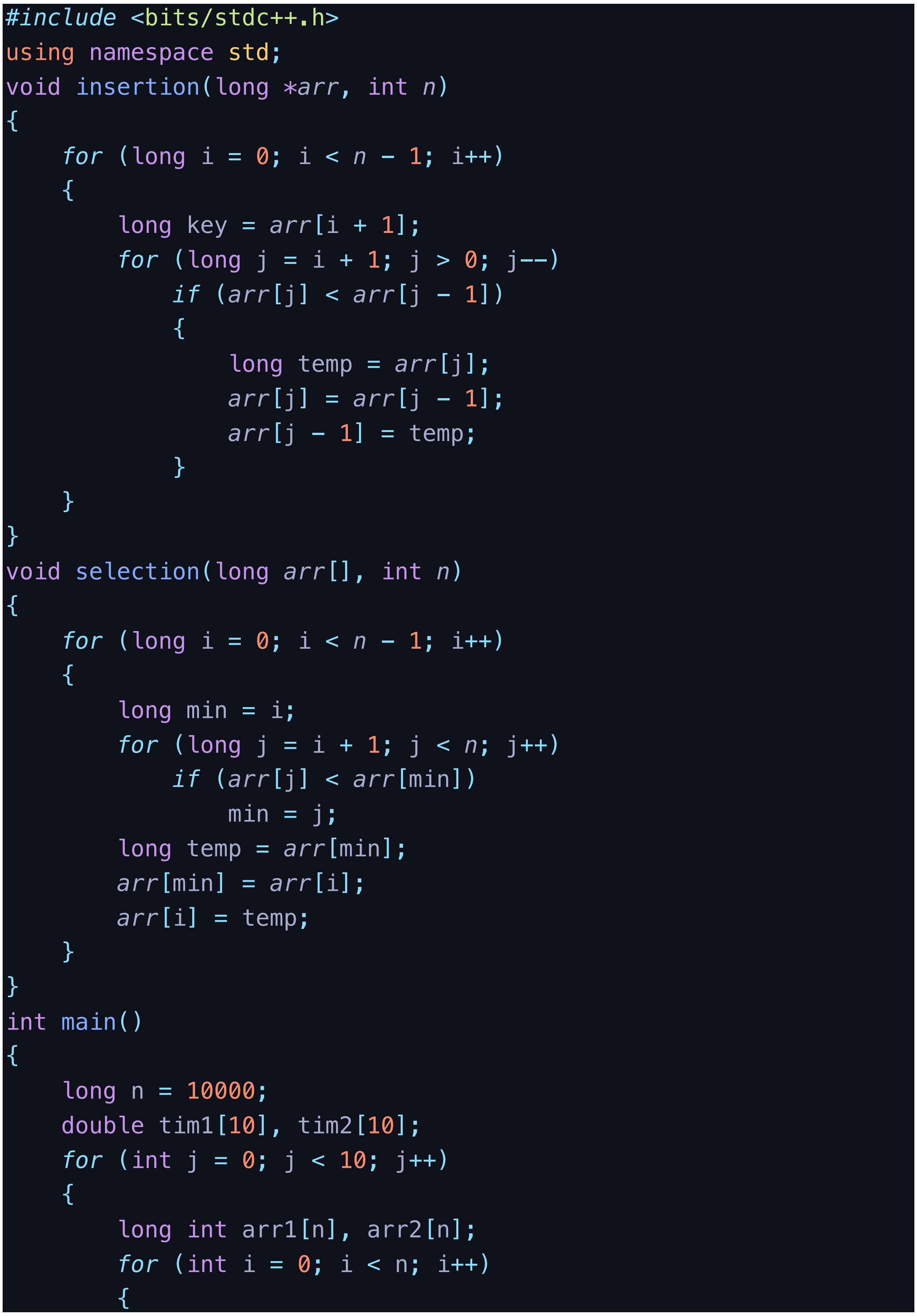
Algorithm Insertion Sort

```
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

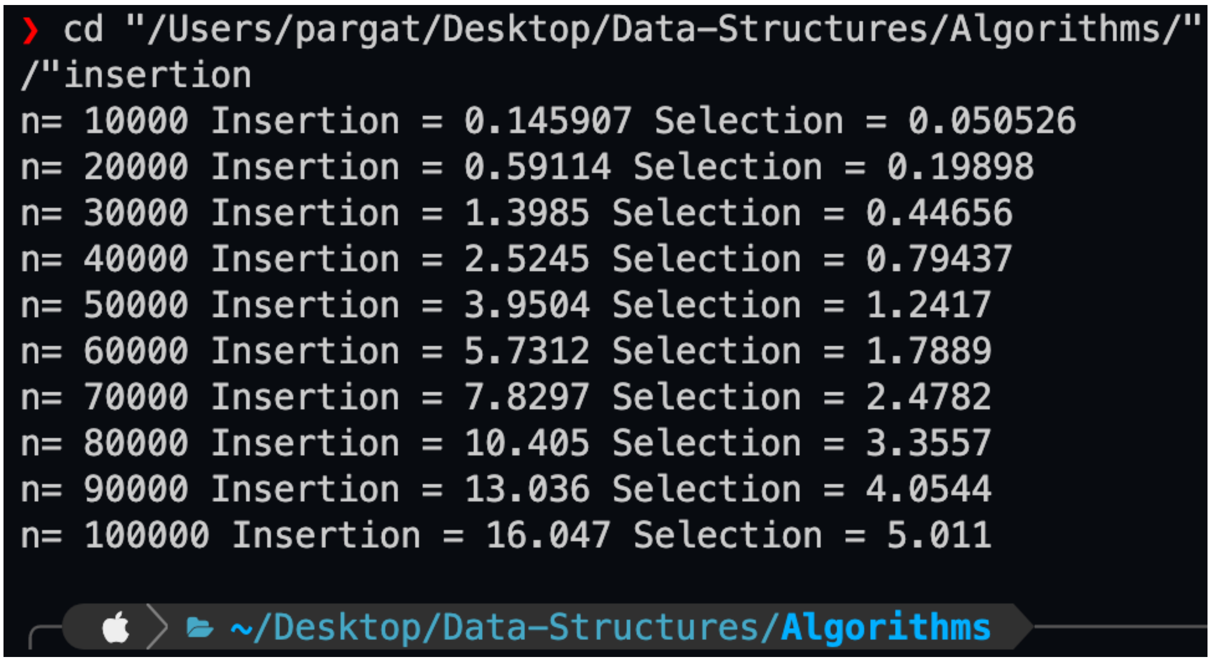
```
SELECTION_SORT (A,n)
//The algorithm takes as parameters an array A[1.. n] and the length n of the array.
//The array A is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

            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 <bits/stdc++.h>
using namespace std;
void insertion(long *arr, int n)
{
    for (long i = 0; i < n - 1; i++)
    {
        long key = arr[i + 1];
        for (long j = i + 1; j > 0; j--)
            if (arr[j] < arr[j - 1])
            {
                long temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
            }
    }
}
void selection(long arr[], int n)
{
    for (long i = 0; i < n - 1; i++)
    {
        long min = i;
        for (long j = i + 1; j < n; j++)
            if (arr[j] < arr[min])
                min = j;
        long temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}
int main()
{
    long n = 10000;
    double tim1[10], tim2[10];
    for (int j = 0; j < 10; j++)
    {
        long int arr1[n], arr2[n];
        for (int i = 0; i < n; i++)
        {
            arr1[i] = n - i;
            arr2[i] = n - i;
        }
        clock_t start, end;
        start = clock();
        ios_base::sync_with_stdio(false);
        insertion(arr1, n);
        end = clock();
        tim1[j] = ((double) (end - start)) / CLOCKS_PER_SEC;
        start = clock();
        selection(arr2, n);
        end = clock();
        tim2[j] = ((double) (end - start)) / CLOCKS_PER_SEC;
        cout << "n= " << n << " Insertion = "<< tim1[j] <<
setprecision(5)<< " Selection = " << tim2[j] << endl;
        n += 10000;
    }

    return 0;
}
```

Output:



```
❯ cd "/Users/pargat/Desktop/Data-Structures/Algorithms/"
/"insertion
n= 10000 Insertion = 0.145907 Selection = 0.050526
n= 20000 Insertion = 0.59114 Selection = 0.19898
n= 30000 Insertion = 1.3985 Selection = 0.44656
n= 40000 Insertion = 2.5245 Selection = 0.79437
n= 50000 Insertion = 3.9504 Selection = 1.2417
n= 60000 Insertion = 5.7312 Selection = 1.7889
n= 70000 Insertion = 7.8297 Selection = 2.4782
n= 80000 Insertion = 10.405 Selection = 3.3557
n= 90000 Insertion = 13.036 Selection = 4.0544
n= 100000 Insertion = 16.047 Selection = 5.011

 ~/Desktop/Data-Structures/Algorithms
```

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 $O(n^2)$

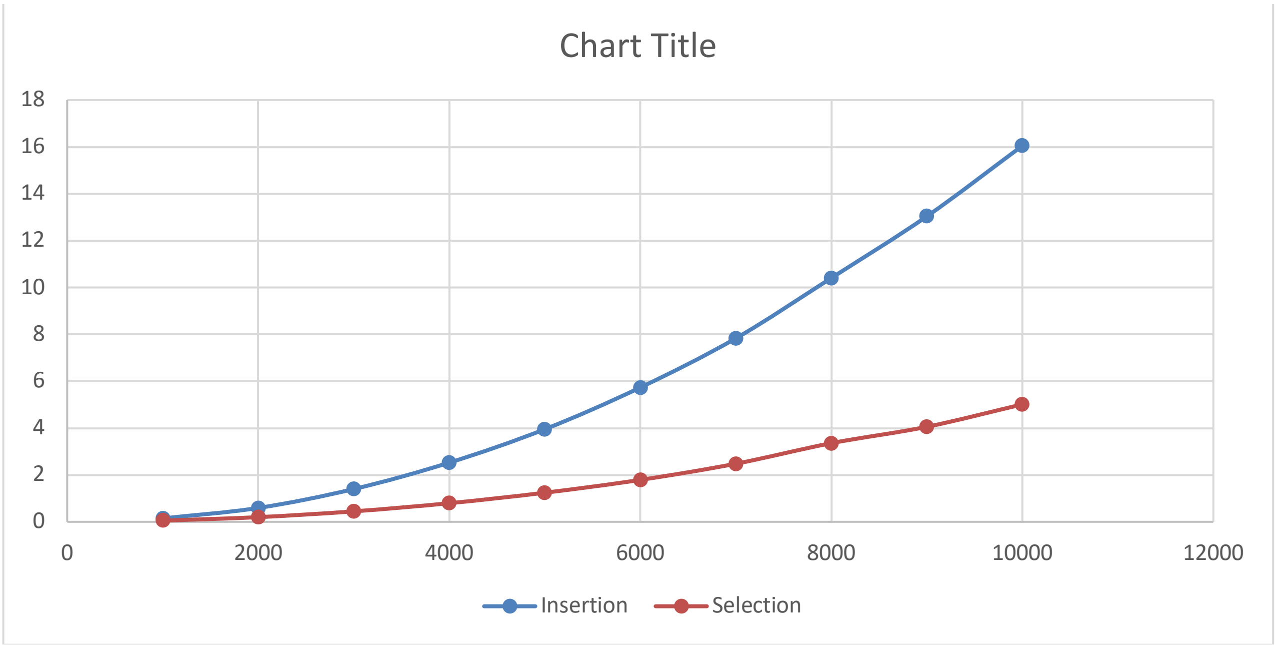
Time complexity for selection sort: $O(n^2)$

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

Hence $O(n^2)$

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

Both of them have the same time complexity O(n2), but selection sort has been proven to be worse than insertion sort for large arrays.



CONCLUSION:

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