Batch: A4 **Roll No.: 1721001**

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:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching 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.

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
```

The space complexity of Insertion sort:

Best-Case Performance	O(n) comparisons. O(1) swaps
Average-Case Performance	O(n^2) comparisons. swaps
Worst-Case Performance	O(n) total, O(1) auxlliary

The space complexity of Selection sort:

Best-Case Performance	O(n^2) comparisons. O(n) swaps
Average-Case Performance	O(n^2) comparisons. O(n) swaps
Worst-Case Performance	O(n) total, O(1) auxiliary

Time complexity for Insertion sort:

Worst-Case	O(n^2)
Best-Case	O(n)
Average-Case	O(n^2)

Time complexity for selection sort:

O(1)

Program of selection sort:

```
import java.util.*;

class Selection
{
	public static void main(String args[])
	{
		int arr[] = new int[50000];
		int i,j,n=50000;
		Random rn = new Random();
		for(i=0;i<n;i++){
			arr[i]=rn.nextInt(1000);}
		/*System.out.println("Before sorting");
		System.out.println(" ");
		for(i=0;i<n;i++){
			System.out.print(" " + arr[i]);}
		System.out.println(" ");
		System.out.println("Sorting begins:");*/
		long st,et,extm;
		st=System.currentTimeMillis();
		for(i=0;i< n-1;i++)
	{
	   int index = i;
	   for (j=i+1;j<n;j++){
```

```
            if (arr[j] < arr[index]){
                index = j;
                                }}
        int sn = arr[index];
        arr[index] = arr[i];
        arr[i] = sn;
                        /*System.out.println(" ");
                        for(int k=0;k<n;k++)
                        {
                                System.out.print(" " + arr[k]);
                        }*/
                }
                et=System.currentTimeMillis();
                /*System.out.println(" ");
                System.out.println("After Sorting");
                System.out.println("");
                for(i=0;i<n;i++)
                {
                        System.out.print(" " + arr[i]);
                }*/
                extm=et-st;
                System.out.println("Time Required: " + extm);
        }}
```

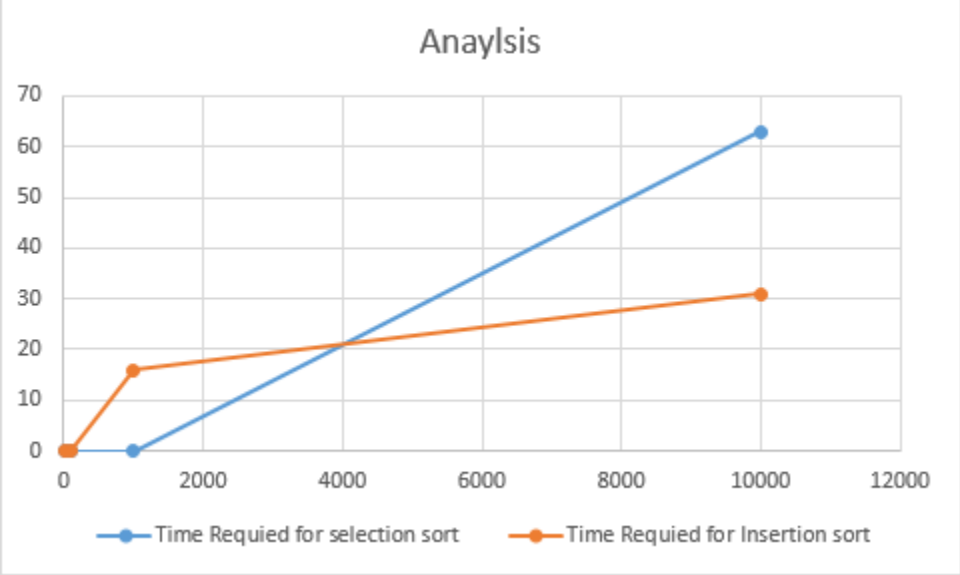
Program of insertion sort:

```
import java.util.*;

class Insertion
{
   void sort(int arr[])
   {
      int n = arr.length;
      for (int i=1; i<n; ++i)
      {
         int key = arr[i];
         int j = i-1;
         while (j>=0 && arr[j] > key)
         {
            arr[j+1] = arr[j];
            j = j-1;
         }
         arr[j+1] = key;
      }
   }
   static void printArray(int arr[])
   {
```

```
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}
public static void main(String args[])
{
                long st,et,extm;
                int arr[]=new int[10000];
                Random rn = new Random();
                for(int i=0;i<10000;i++)
                {
                        arr[i]=rn.nextInt(1000);
                }
    Insertion ob = new Insertion();
                st=System.currentTimeMillis();
    ob.sort(arr);
                et=System.currentTimeMillis();
                extm=et-st;
    //printArray(arr);
                System.out.println("Time Required: " + extm);
}}
```

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



Conclusion: Thus we have analysed and studied insertion and selection sort with their worst, average and best case.