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

Batch: A1 **Roll No.: 16010123012**

Experiment No.: 1

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

Signature of the Staff In-charge with date

Title: Study, Implementation, and Performance Comparison of Selection Sort, Insertion Sort to analyze sorting strategies.

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 $x \leftarrow A[i]$
FOR $j \leftarrow i + 1$ to n do
  IF A[j] < min x then
    min $j \leftarrow j$
    min $x \leftarrow$ A[j]
A[min j] $\leftarrow$ A [i]
A[i] $\leftarrow$ min x

Code:

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

void insertionSort(long arr[], int n)
{
  for (int i = 1; i < n; i++)
  {
    int select = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > select)
    {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = select;
  }
}

int main()
{
  long n = 100;
  long arr[n];
  srand(time(0));
  int i;
  for (i = 0; i < n; i++)
  {
    arr[i] = rand() % 10000;
  }
  auto start = high_resolution_clock::now();
  insertionSort(arr, n);
  auto stop = high_resolution_clock::now();
  auto duration = duration_cast<microseconds>(stop - start);
  cout << "\n";
  cout << "Time taken by function: " << duration.count() << " microseconds" << endl;
}
```

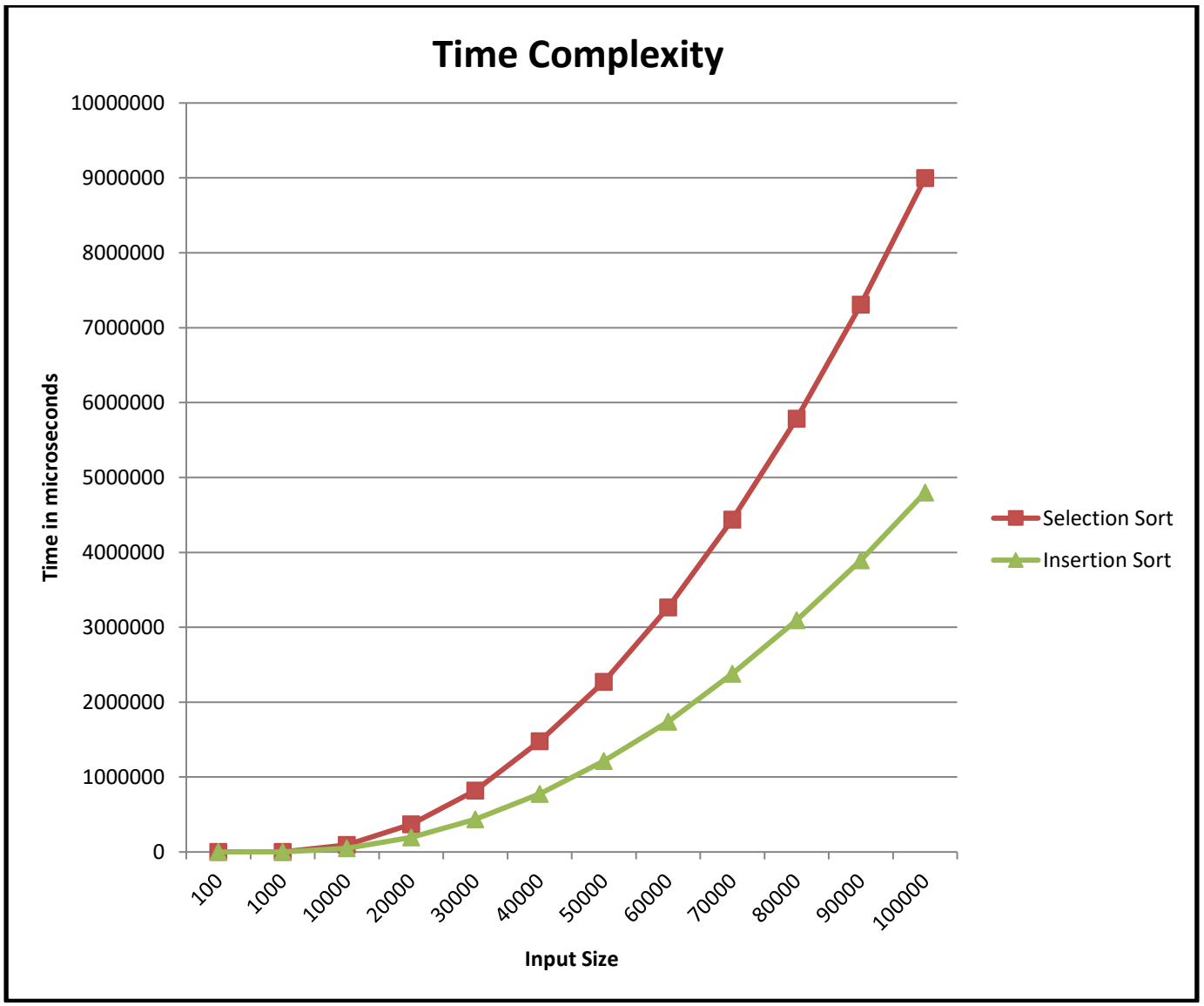
```cpp
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;

void selectionSort(long arr[], int n)
{
  for (int i = 0; i < n - 1; i++)
  {
    int min = i;
    for (int j = i + 1; j < n; j++)
    {
      if (arr[j] < arr[min])
      {
        min = j;
      }
    }
    swap(arr[i], arr[min]);
  }
}

int main()
{
  long n = 10000;
  long arr[n];
  srand(time(0));
  int i;
  for (i = 0; i < n; i++)
  {
    arr[i] = rand() % 10000;
  }

  auto start = high_resolution_clock::now();
  selectionSort(arr, n);
  auto stop = high_resolution_clock::now();
  auto duration = duration_cast<microseconds>(stop - start);

  cout << "\n";
  cout << "Time taken by function: " << duration.count() << " microseconds" << endl;
}
```

The space complexity of Insertion sort: O(n)
The space complexity of Selection sort: O(n)

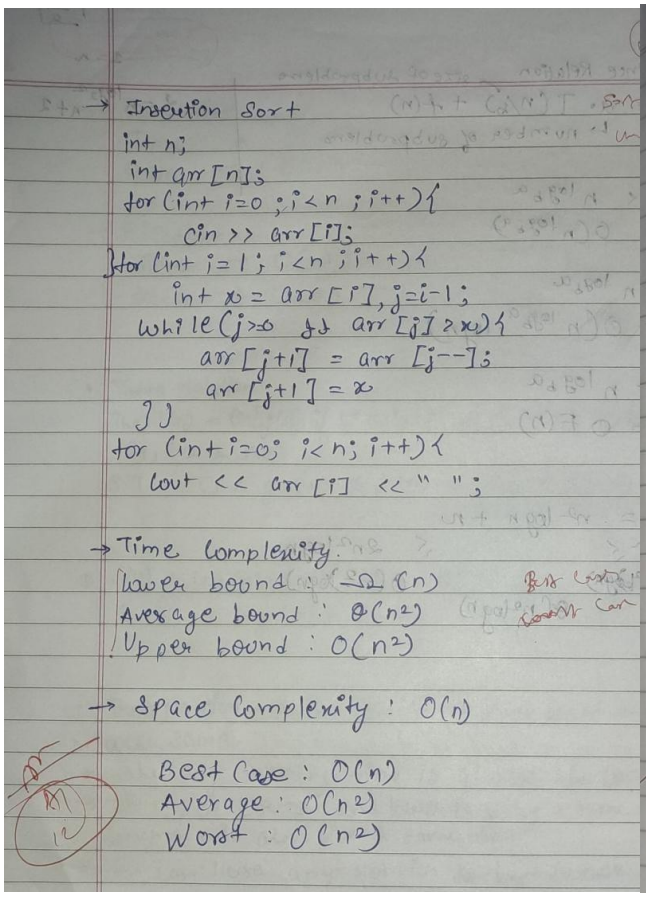
Time complexity for Insertion sort: $O(n^2)$
Time complexity for selection sort: $O(n^2)$

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

Input Size	Selection Sort	Insertion Sort
100	0	0
1000	997	985
10000	93775	48895
20000	369041	195539
30000	818808	435840
40000	1480039	775922
50000	2271948	1214776
60000	3263268	1739347
70000	4439121	2380590
80000	5784523	3093693
90000	7306486	3892614
100000	8998922	4798160



Lab Work:



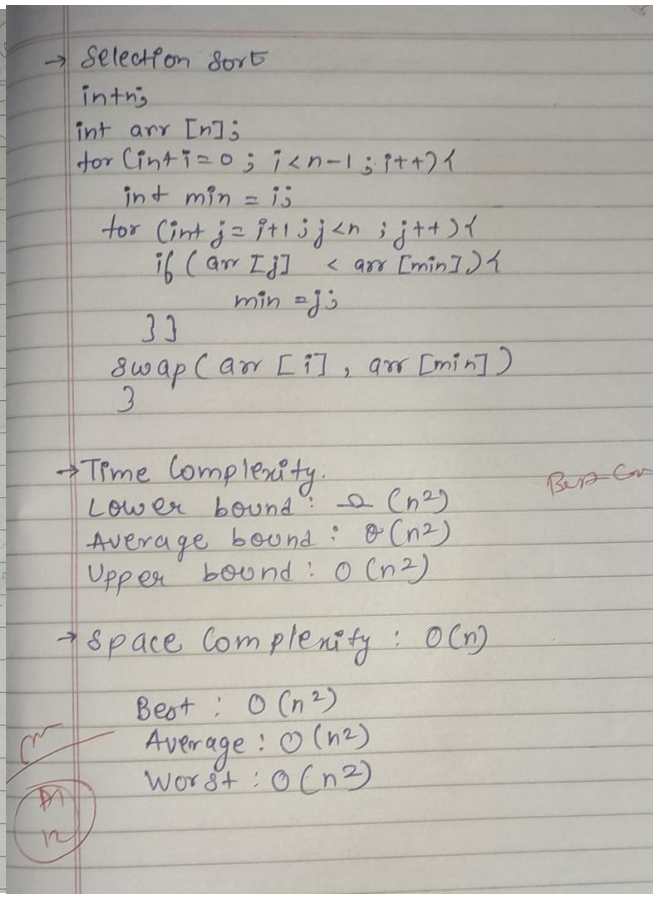
→ Insertion Sort

```
int n;
int arr[n];
for(int i=0 ; i<n ; i++){
    cin >> arr[i];
}for(int i=1; i<n ; i++){
    int x = arr[i], j=i-1;
  while(j>=0 && arr[j]>x){
    arr[j+1] = arr[j--];
    arr[j+1] = x
  }}
for(int i=0; i<n; i++){
  cout << arr[i] << " ";
```

→ Time Complexity:
Lower bound : Ω(n)
Average bound : Θ(n²)
Upper bound : O(n²)

→ Space Complexity : O(n)

Best Case : O(n)
Average : O(n²)
Worst : O(n²)



→ Selection Sort

```
int n;
int arr[n];
for(int i=0; i<n-1; i++){
   int min = i;
  for(int j=i+1; j<n ; j++){
    if(arr[j] < arr[min]){
        min = j;
  }}
swap(arr[i], arr[min])
}
```

→ Time Complexity.
Lower bound : Ω(n²)
Average bound : Θ(n²)
Upper bound : O(n²)

→ Space Complexity : O(n)

Best : O(n²)
Average : O(n²)
Worst : O(n²)

CONCLUSION:

I on analyzed and compared the performance of Insertion Sort and Selection Sort. I implemented both algorithms and observed their time complexities to be $O(n^2)$ for the average and worst cases. The space complexity for both algorithms is O(n). Through this experiment, I recognized the importance of selecting appropriate algorithms based on the size and characteristics of the input data, a crucial aspect of algorithm analysis and practical application.