COMP 314: Algorithms and Complexity
Lab work 1: Sorting

Purpose

Implementation, testing and performance measurement of sorting algorithms.

Methodology:

Some of the methodologies and the code snippet is given below. For complete code please refer to https://github.com/ranjanlamsal/COMP314Lab.git .

Input Generation:

Random arrays of integers are generated using the generate_array function, with the lengths of the arrays varying from 10 to 100,000.

```
def generate_array(n):
    A = []
    for i in range(n):
        A.append(randint(0, 100000000000))

    return A
```

Sorting Algorithms:

Two sorting algorithms, insertion sort and selection sort, are applied to the generated arrays to measure their execution times.

```
#selection_sort.py
def selection_sort(A):
    n = len(A)

    for  i in range(n-1):
        m = i
        for j in range(i+1, n):
            if A[j] < A[m]:
                m = j
```

```
        temp = A[i]
        A[i] = A[m]
        A[m] = temp

    return A
```

```
#insertion_sort.py
def insertion_sort(A):
    n = len(A)
    for  i in range(1, n):
        value = A[i]
        j = i-1

        while(j >=0 and value < A[j]):
            A[j+1] = A[j]
            j = j-1

        A[j+1] = value


    return A
```

Execution Time Measurement:

The execution times of both sorting algorithms are recorded using the time module in Python.

```
def calculate_insertion_time(n):
    testArray1 = generate_array(n)

    start = time()

    resultArray1_insertion = insertion_sort(testArray1)

    end = time()
```

```
    return end - start


def calculate_selection_time(n):
    testArray1 = generate_array(n)

    start = time()

    resultArray1_selection = selection_sort(testArray1)

    end = time()

    return end - start
```

Visualization

The execution times of insertion sort and selection sort for different input sizes are compared and graph is visualized using matplotlib

```
sizes = [10, 100, 1000, 10000, 100000]
insertion_time= []
selection_time = []

for size in sizes:
    insertion_time.append(calculate_insertion_time(size))
    selection_time.append(calculate_selection_time(size))


# Graph
x = sizes
y1 = insertion_time
y2 = selection_time
print(f"Insertion sort time:{insertion_time}")
print(f"selection sort time:{selection_time}")
# Plotting
plt.plot(x, y1, marker='o', color='b', label='Insertion sort')
plt.plot(x, y2, marker='s', color='r', label='Selection Sort')
```

```
# Adding labels and title
plt.xlabel('N')
plt.ylabel('Time')
plt.title('Insertion and Selection Sort')

# Adding legend
plt.legend()

# Displaying the plot
plt.grid(True)
plt.show()
```

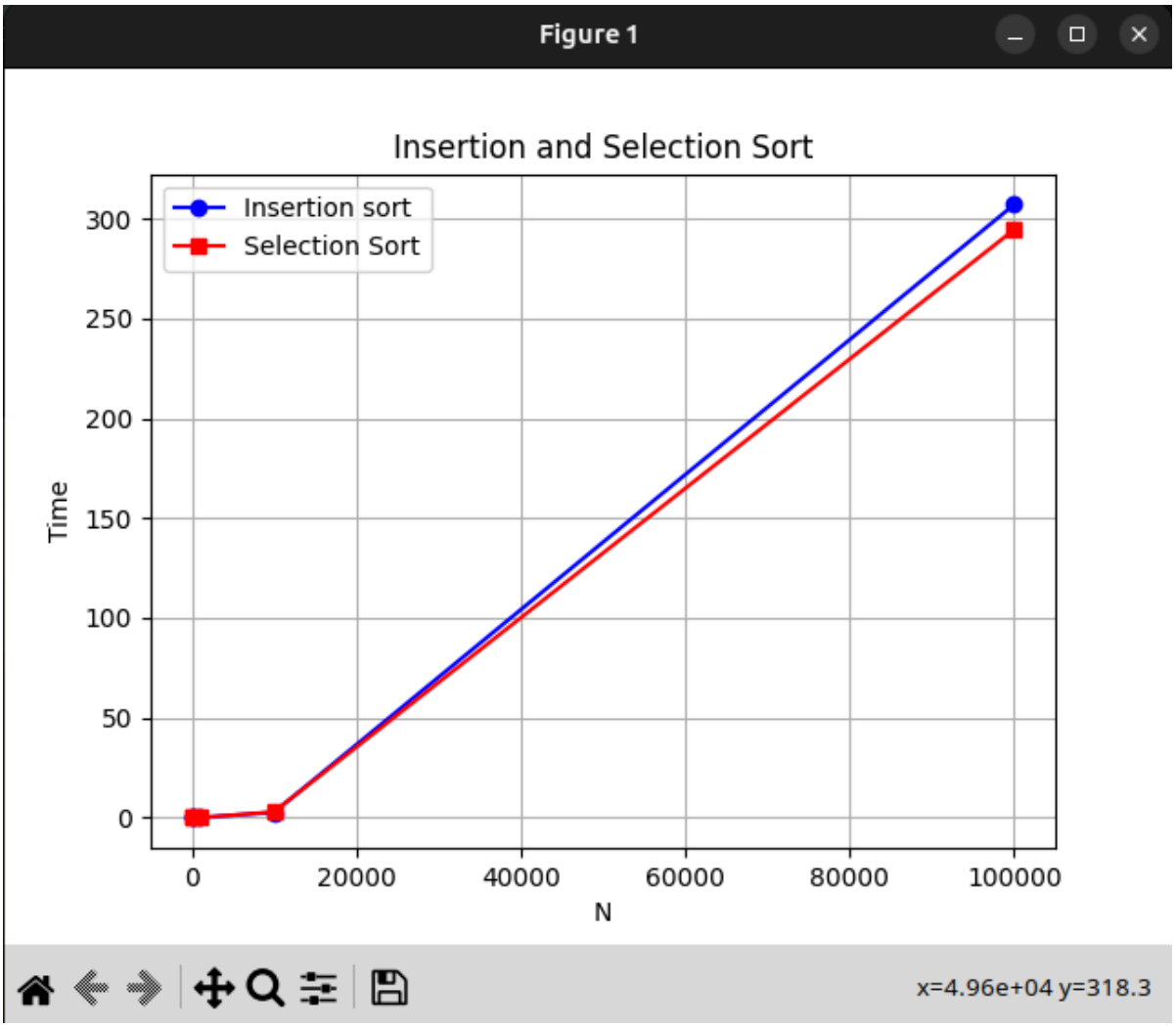
Observations:

The execution times of insertion sort and selection sort increase as the size of the input arrays grows. However, selection sort performed better than insertion sort for larger input sizes. This observation aligns with the expected time complexity of O(n^2) for both algorithms, with selection sort having a slightly better average-case performance compared to insertion sort.

For input_sizes = [10, 100, 1000, 10000, 100000]

Time lapses for Insertion sort = [1.430511474609375e-05, 0.0004062652587890625, 0.02154684066772461, 2.634857654571533, 306.99769043922424]

Time lapses for selection sort = [1.2636184692382812e-05, 0.0003943443298339844, 0.026239871978759766, 2.8456101417541504, 294.4455142021179]



Submitted By:

Ranjan Lamsal
Roll call: 26
Group: CE (3rd year 2nd Semester)