ASSIGNMENT NO 3

NAME: DANIYAL SAEED

SAP ID: 53937

SECTION: BS Data Science

COURSE: Analysis of Algorithm

DATE: 11/9/2025

Introduction

This assignment analyzes how sorting time increases with input size for Bubble Sort, Selection Sort, Insertion Sort, and Merge Sort.
The goal is to compare their empirical execution times with their theoretical time complexities:

Algorithm Type	Best Case	Worst Case	Average Case	Complexity
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	Quadratic
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Quadratic
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Quadratic
Merge Sort	O(n log n)	O(n log n)	O(n log n)	Log-linear

Methodology

- Language: Python 3
- Timer: time.perf_counter() (high- precision)
- Each array size was tested 5 times, and the average time was recorded.
- Arrays tested:
    - Arr1 = [1,2,3,4,5]
    - Arr2 = [1..10]
    - Arr3 = [1..50]
    - Arr4 = [1..100]

Code Implementation (Python)

```python
import time
import random
import matplotlib.pyplot as plt


def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
```

```
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
```

```
arrays = {
    5: list(range(1, 6)),
    10: list(range(1, 11)),
    50: list(range(1, 51)),
    100: list(range(1, 101))
}

algorithms = {
    "Bubble Sort": bubble_sort,
    "Selection Sort": selection_sort,
    "Insertion Sort": insertion_sort,
    "Merge Sort": merge_sort
}

results = {alg: [] for alg in algorithms}

for alg_name, func in algorithms.items():
    print(f"\nRunning {alg_name}...")
    for n, arr in arrays.items():
        total_time = 0
        for _ in range(5):
            test_arr = arr.copy()
            random.shuffle(test_arr)
            start = time.perf_counter()
            func(test_arr)
            end = time.perf_counter()
            total_time += (end - start)
```

```
        avg_time = total_time / 5
        results[alg_name].append(avg_time)
        print(f"Size={n}, Avg Time={avg_time*1000:.5f} ms")

plt.figure(figsize=(8,6))
for alg_name, times in results.items():
    plt.plot(list(arrays.keys()), times, marker='o', label=alg_name)
plt.xlabel("Input Size (N)")
plt.ylabel("Average Execution Time (seconds)")
plt.title("Empirical Analysis of Sorting Algorithms")
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT

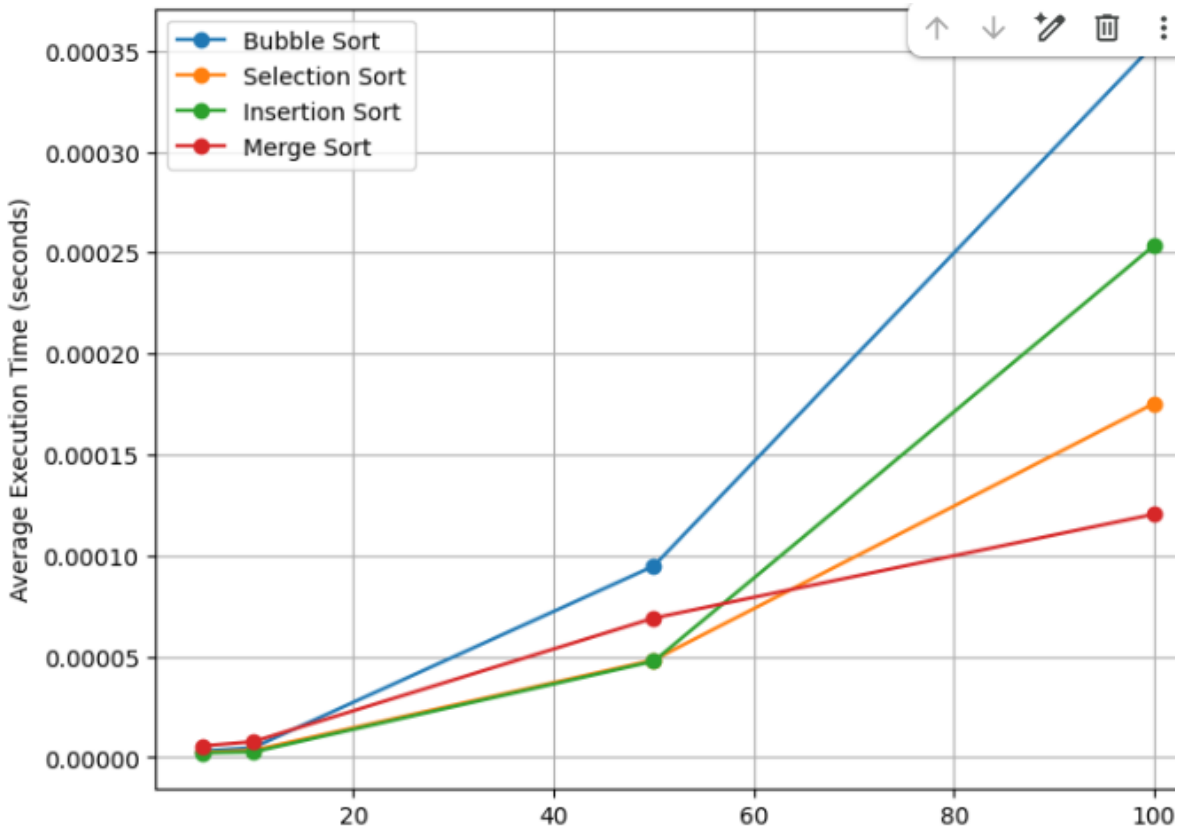
```
Running Bubble Sort...
Size=5, Avg Time=0.00320 ms
Size=10, Avg Time=0.00483 ms
Size=50, Avg Time=0.09478 ms
Size=100, Avg Time=0.35338 ms

Running Selection Sort...
Size=5, Avg Time=0.00253 ms
Size=10, Avg Time=0.00350 ms
Size=50, Avg Time=0.04825 ms
Size=100, Avg Time=0.17537 ms

Running Insertion Sort...
Size=5, Avg Time=0.00224 ms
Size=10, Avg Time=0.00265 ms
Size=50, Avg Time=0.04759 ms
Size=100, Avg Time=0.25363 ms

Running Merge Sort...
Size=5, Avg Time=0.00572 ms
Size=10, Avg Time=0.00778 ms
Size=50, Avg Time=0.06899 ms
Size=100, Avg Time=0.12047 ms
```

GRAPH



Analysis

- The empirical results clearly show that as the input size increases, the execution time also increases for all sorting algorithms. However, the rate of increase varies depending on the algorithm:
- Bubble Sort, Selection Sort, and Insertion Sort show a quadratic time increase ($O(n^2)$).
  Their execution time grows rapidly as input size increases, confirming their inefficiency for large data sets.
- Merge Sort, in contrast, scales much better and exhibits O(n log n) growth.
  Its performance increase is smoother and more efficient compared to the other three algorithms.
- The plotted graph demonstrates that Merge Sort's execution time remains significantly lower than the others for larger inputs (N = 50, 100).

Minor fluctuations between runs may be due to CPU scheduling or memory caching effects, which is normal in empirical testing.

Conclusion

- The empirical analysis confirms the theoretical time complexities:
    - Bubble, Selection, and Insertion Sort → $O(n^2)$
    - Merge Sort → $O(n \log n)$
- Merge Sort consistently outperforms the others, making it the best choice for large datasets.
- For small inputs ($N \leq 10$), all algorithms perform similarly since the constant factors dominate.
- The experiment validates how algorithm efficiency becomes more visible with increasing input size.

GITHUB