

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

Institute of Computer Technology

B. Tech Computer Science and Engineering

Sub: Algorithm Analysis and Design

Practical 3

NextMid Technology is an American food company that manufactures, markets, and distributes spices, seasoning mixes, condiments, and other flavoring products for the industrial, restaurant, institutional, and home markets, they are having some number quantity of different categories item food, kindly help them to sort data using any three sorting methods and determine the time required to sort the elements. Repeat the experiment for different values of n , the number of elements in the list to be sorted and plot a graph of the comparison between them.

Design the algorithm for the same and implement using the programming language of your choice. Make comparative analysis for various use cases & input size.

CODE:

```
import streamlit as st
```

```
import numpy as np
```

```
import time
```

```
import matplotlib.pyplot as plt
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

Sorting algorithms

```
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
            k += 1
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

k += 1

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
```

```
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]
```

Function to time sorting algorithms

```
def time_algorithm(algorithm, arr):
    start_time = time.time()
    algorithm(arr.copy())
    return time.time() - start_time
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

Streamlit app

```
st.title("Sorting Algorithm Performance Comparison")
```

```
st.write("This app compares the performance of Merge Sort, Quick Sort, and  
Bubble Sort on different input sizes.")
```

User input for array sizes

```
n_values = st.slider("Select the range for the number of elements (n):", 100, 5000,  
(100, 1000), step=100)
```

```
n_steps = st.slider("Step size for increasing n:", 100, 1000, 100)
```

```
n_list = list(range(n_values[0], n_values[1] + 1, n_steps))
```

Arrays to store timings

```
merge_times = []
```

```
quick_times = []
```

```
bubble_times = []
```

```
st.write("### Running Comparisons...")
```

```
for n in n_list:
```

```
    arr = np.random.randint(0, 10000, n)
```

Time each sorting algorithm

```
merge_times.append(time_algorithm(merge_sort, arr))
```

```
quick_times.append(time_algorithm(quick_sort, arr))
```

```
bubble_times.append(time_algorithm(bubble_sort, arr))
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

```
st.write(f"Completed for n={n} ")
```

```
# Plotting results
```

```
st.write("### Time Complexity Comparison")
```

```
plt.figure(figsize=(10, 6))
```

```
plt.plot(n_list, merge_times, label="Merge Sort", color="blue", marker="o")
```

```
plt.plot(n_list, quick_times, label="Quick Sort", color="green", marker="x")
```

```
plt.plot(n_list, bubble_times, label="Bubble Sort", color="red", marker="s")
```

```
plt.xlabel("Number of Elements (n)")
```

```
plt.ylabel("Time (seconds)")
```

```
plt.title("Sorting Algorithm Performance Comparison")
```

```
plt.legend()
```

```
st.pyplot(plt)
```

```
st.write("""
```

```
    ### Analysis:
```

```
    - **Merge Sort**: Expected  $O(n \log n)$  complexity, performs well on large datasets.
```

```
    - **Quick Sort**: Also  $O(n \log n)$  on average, but can degrade to  $O(n^2)$  in the worst case.
```

```
    - **Bubble Sort**: Expected  $O(n^2)$  complexity, is inefficient on large datasets.
```

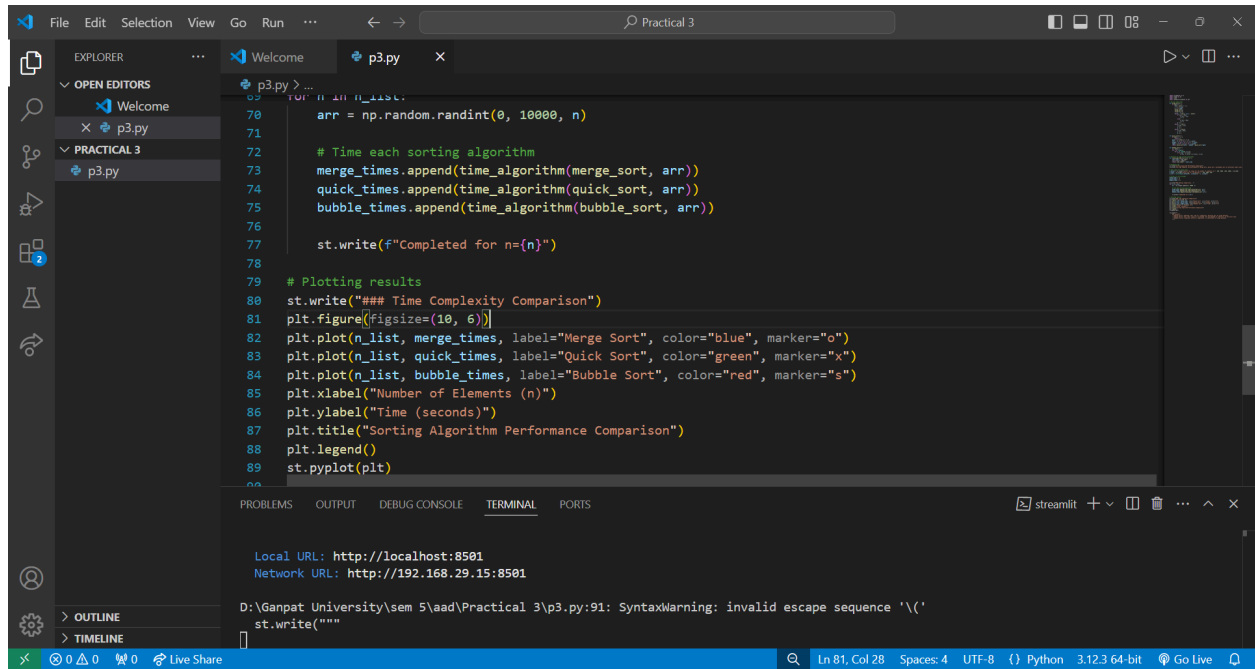
```
    """)
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)



The image shows a Visual Studio Code editor window with a Python file named `p3.py` open. The script generates random data and measures the execution time of Merge Sort, Quick Sort, and Bubble Sort for different array sizes. It then uses Matplotlib to create a line plot titled "Sorting Algorithm Performance Comparison" showing time in seconds versus the number of elements. The plot uses different colors and markers for each algorithm: Merge Sort (blue circles), Quick Sort (green crosses), and Bubble Sort (red squares).

```
70 arr = np.random.randint(0, 10000, n)
71
72 # Time each sorting algorithm
73 merge_times.append(time_algorithm(merge_sort, arr))
74 quick_times.append(time_algorithm(quick_sort, arr))
75 bubble_times.append(time_algorithm(bubble_sort, arr))
76
77 st.write(f"Completed for n={n}")
78
79 # Plotting results
80 st.write("### Time Complexity Comparison")
81 plt.figure(figsize=(10, 6))
82 plt.plot(n_list, merge_times, label="Merge Sort", color="blue", marker="o")
83 plt.plot(n_list, quick_times, label="Quick Sort", color="green", marker="x")
84 plt.plot(n_list, bubble_times, label="Bubble Sort", color="red", marker="s")
85 plt.xlabel("Number of Elements (n)")
86 plt.ylabel("Time (seconds)")
87 plt.title("Sorting Algorithm Performance Comparison")
88 plt.legend()
89 st.pyplot(plt)
```

The terminal window at the bottom shows the Streamlit application running successfully. It displays the local and network URLs and a syntax warning for an invalid escape sequence in the `st.write` command.

```
Local URL: http://localhost:8501
Network URL: http://192.168.29.15:8501

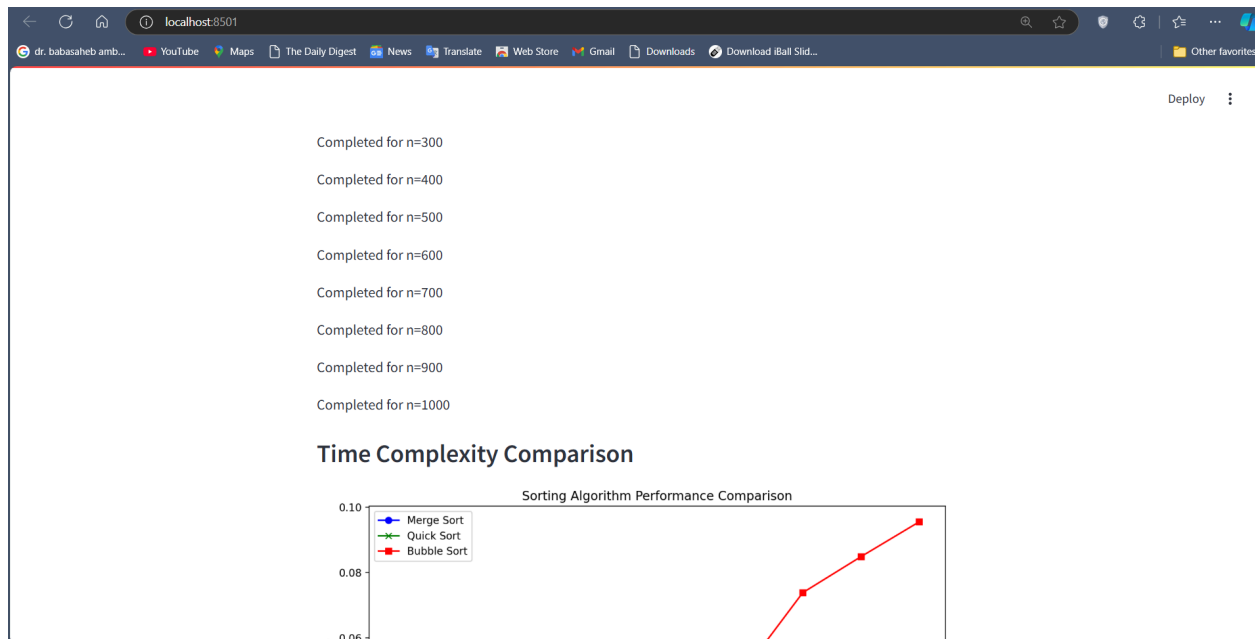
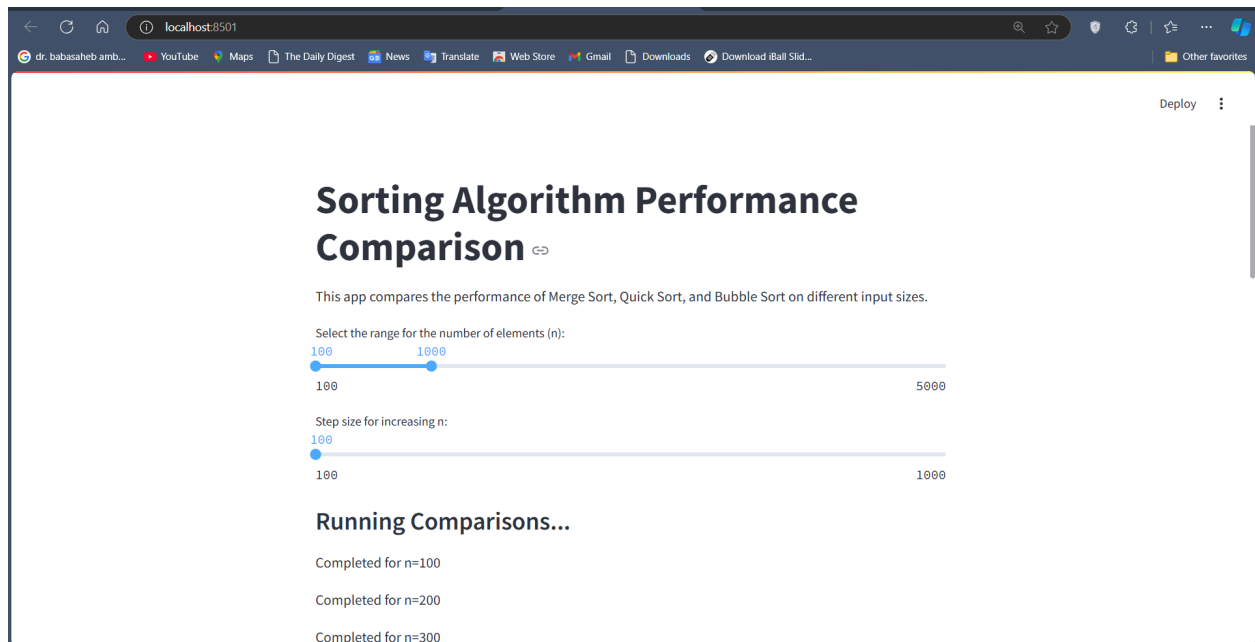
D:\Ganpat University\sem 5\aad\Practical 3\p3.py:91: SyntaxWarning: invalid escape sequence '\('
  st.write("""
```

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

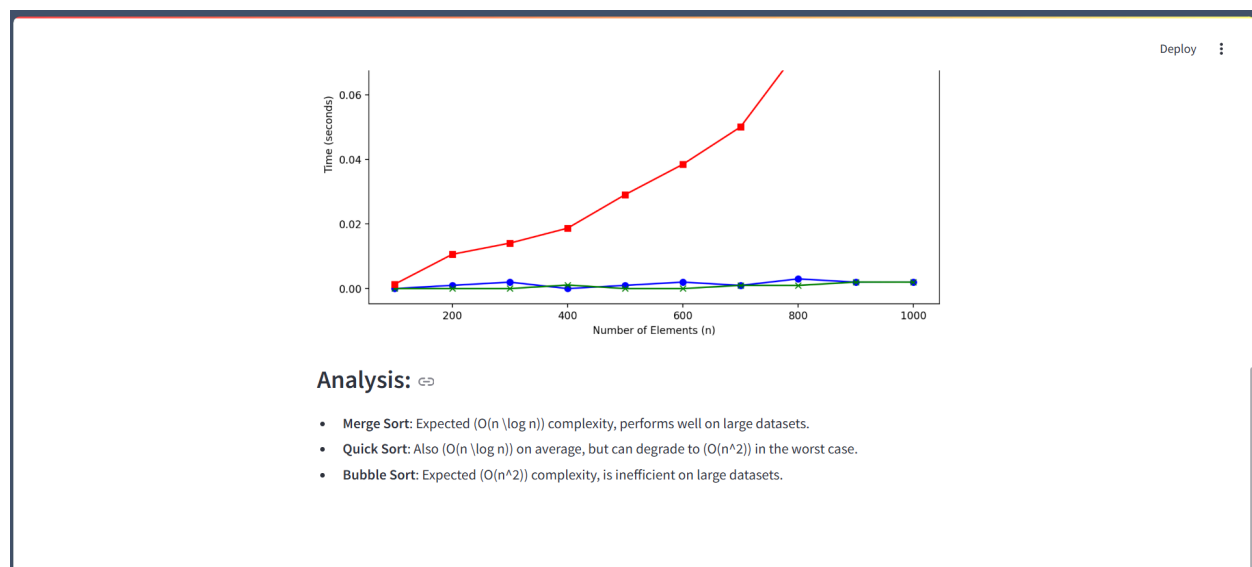
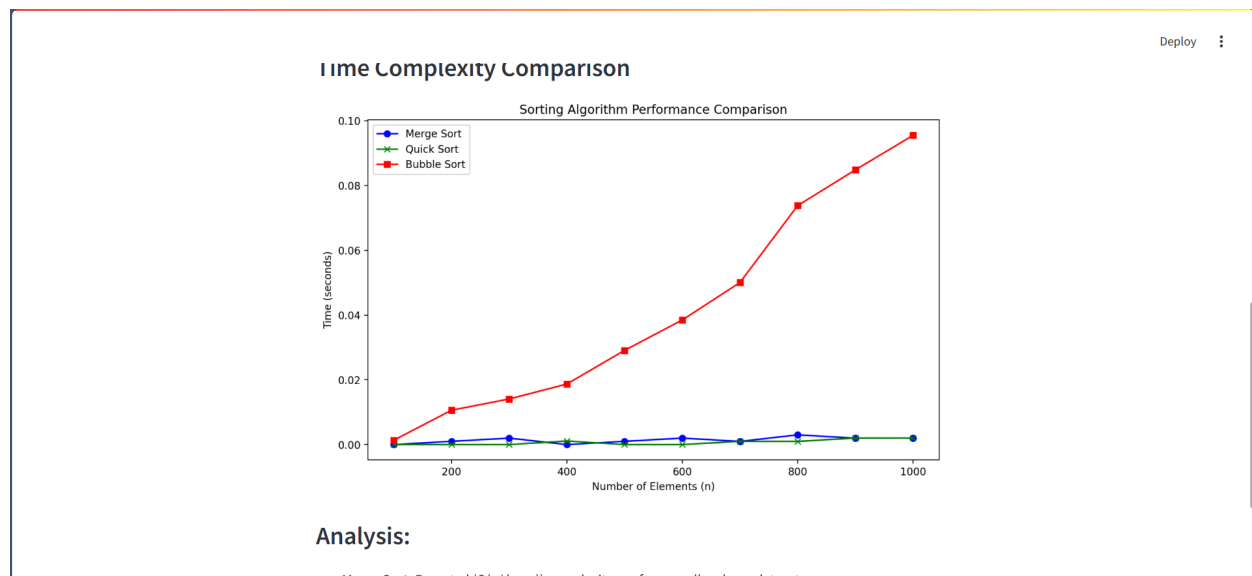


Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)



Questions:

What is the best, average and worst case analysis of algorithms?

In algorithm analysis, the best case refers to the minimum time an algorithm takes to complete a task, given the most favorable input conditions. This case often provides an ideal but less practical view since real-world data doesn't always follow this ideal pattern. The average case time complexity represents the

expected runtime over typical inputs, showing how the algorithm generally performs. This is usually the most valuable measure, as it approximates the performance on random data. The worst case time complexity describes the maximum time an algorithm might need to complete its task, given the least favorable input conditions. This upper bound is crucial to understand because it shows the slowest performance one can expect, which is important for ensuring reliability in critical applications.

Which are different asymptotic notations? What is their use?

Big O Notation

Big O notation provides an **upper bound** on the runtime or space complexity of an algorithm, describing the **worst-case scenario**. It shows how the runtime grows at most as the input size increases. For example, an algorithm with $O(n \log n)$ complexity will never take more than $n \log n$ time steps for sufficiently large inputs. Big O is the most commonly used notation because it ensures the algorithm won't exceed a certain time limit, which is crucial for designing reliable systems.

Big Omega Notation Big Omega notation offers a **lower bound** on the runtime or space complexity, describing the **best-case scenario** for the algorithm. It shows the minimum time the algorithm will take under ideal input conditions. For instance, if an algorithm has $\Omega(n)$ complexity, it will take at least n time steps for large inputs. This notation is useful when assessing the minimum performance guarantee of an algorithm.

Big Theta Notation

Big Theta notation describes a **tight bound** for an algorithm's complexity, providing both an upper and lower bound. It applies when the algorithm's growth

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)

rate in the best, average, and worst cases are all similar. For example, if an algorithm has $\Theta(n \log n)$ complexity, its runtime will grow exactly at the rate of $n \log n$ for large n . Big Theta is used when an algorithm's performance is consistent and predictable across different inputs.

What is the time complexity of above 3 sorting algorithms in all cases?

1. Merge Sort

- Best Case: $O(n \log n)$
- Average Case: $O(n \log n)$
- Worst Case: $O(n \log n)$

2. Quick Sort

- Best Case: $O(n \log n)$
- Average Case: $O(n \log n)$
- Worst Case: $O(n^2)$

3. Bubble Sort

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

Darshan Singh Chauhan

22162581001

BTech CSE

Batch 52 (CSE)