NAME: DEVANSHI JAIN
ER-NO: 22162101006
BATCH: 51 [CBA]

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:

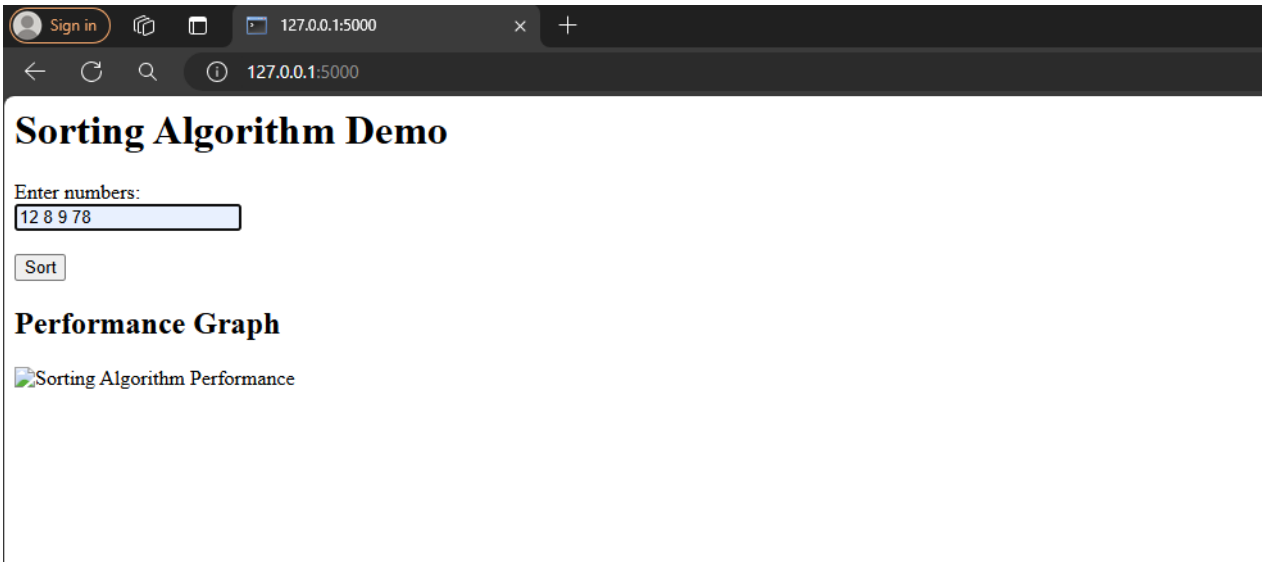
```
from flask import Flask, request, render_template_string, send_file
import time
import matplotlib.pyplot as plt
import io
import random

app = Flask(__name__)

def bubble_sort(arr):
    n = len(arr)
    start_time = time.time()
    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]
    end_time = time.time()
    return (end_time - start_time) * 1000
def quick_sort(arr):
    start_time = time.time()

    def partition(low, high):
        i = low - 1
        pivot = arr[high]
        for j in range(low, high):
            if arr[j] < pivot:
                i += 1
                arr[i], arr[j] = arr[j], arr[i]
        arr[i+1], arr[high] = arr[high], arr[i+1]
        return i + 1

    def quick_sort_recursive(low, high):
        if low < high:
            pi = partition(low, high)
            quick_sort_recursive(low, pi - 1)
            quick_sort_recursive(pi + 1, high)

    quick_sort_recursive(0, len(arr) - 1)
    end_time = time.time()
    return (end_time - start_time) * 1000

def merge_sort(arr):
    start_time = time.time()
```

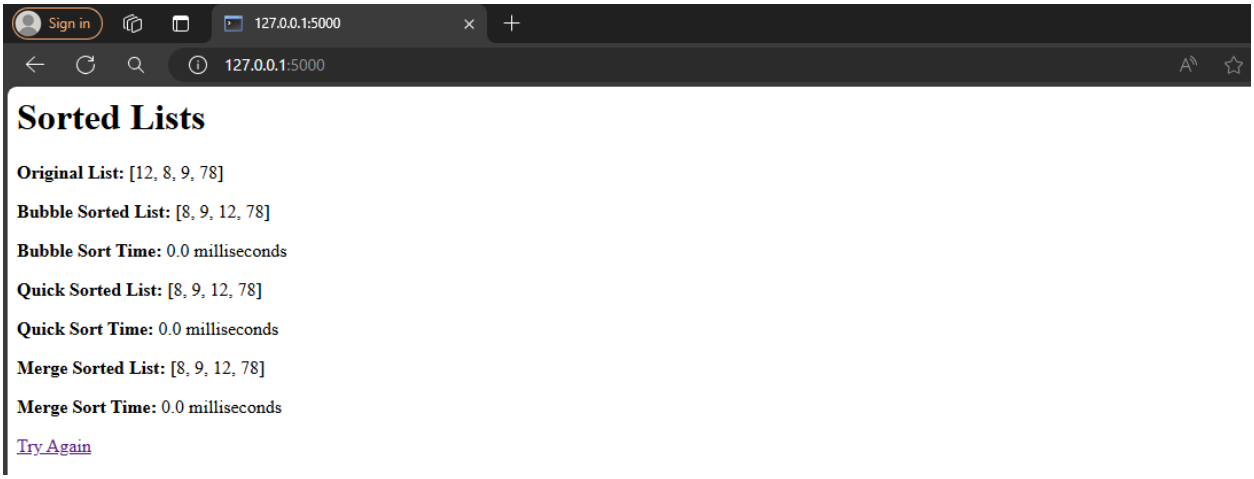
```
    def merge(left, right):
        result = []
        i = j = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                result.append(left[i])
                i += 1
            else:
                result.append(right[j])
                j += 1
        result.extend(left[i:])
        result.extend(right[j:])
        return result

    def merge_sort_recursive(arr):
        if len(arr) <= 1:
            return arr
        mid = len(arr) // 2
        left = merge_sort_recursive(arr[:mid])
        right = merge_sort_recursive(arr[mid:])
        return merge(left, right)

    arr[:] = merge_sort_recursive(arr)
    end_time = time.time()
    return (end_time - start_time) * 1000  # Convert to milliseconds

def measure_sort_times(size):
    arr = [random.randint(0, 10000) for _ in range(size)]
    bubble_time = bubble_sort(arr.copy())
    quick_time = quick_sort(arr.copy())
    merge_time = merge_sort(arr.copy())
    return bubble_time, quick_time, merge_time

def plot_sorting_times():
    sizes = [10, 50, 100, 200, 300, 400, 500]
    bubble_times = []
    quick_times = []
    merge_times = []

    for size in sizes:
        bubble_time, quick_time, merge_time = measure_sort_times(size)
        bubble_times.append(bubble_time)
        quick_times.append(quick_time)
        merge_times.append(merge_time)
```

```
    plt.figure(figsize=(10, 6))
    plt.plot(sizes, bubble_times, label='Bubble Sort', marker='o')
    plt.plot(sizes, quick_times, label='Quick Sort', marker='o')
    plt.plot(sizes, merge_times, label='Merge Sort', marker='o')
    plt.xlabel('Number of Elements')
    plt.ylabel('Time (milliseconds)')
    plt.title('Sorting Algorithm Performance')
    plt.legend()
    plt.grid(True)

    img = io.BytesIO()
    plt.savefig(img, format='png')
    img.seek(0)
    plt.close()
    return img

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        try:
            elements = request.form["elements"]
            arr = list(map(int, elements.split()))

            bubble_arr = arr.copy()
            bubble_time = bubble_sort(bubble_arr)

            quick_arr = arr.copy()
            quick_time = quick_sort(quick_arr)

            merge_arr = arr.copy()
            merge_time = merge_sort(merge_arr)

            return render_template_string("""
                <h1>Sorted Lists</h1>
                <p><strong>Original List:</strong> {{ original }}</p>
                <p><strong>Bubble Sorted List:</strong> {{ bubble }}</p>
                <p><strong>Bubble Sort Time:</strong> {{ bubble_time }}
milliseconds</p>
                <p><strong>Quick Sorted List:</strong> {{ quick }}</p>
                <p><strong>Quick Sort Time:</strong> {{ quick_time }}
milliseconds</p>
                <p><strong>Merge Sorted List:</strong> {{ merge }}</p>
                <p><strong>Merge Sort Time:</strong> {{ merge_time }}
milliseconds</p>
                <a href="/">Try Again</a>
```

```
                <h2>Performance Graph</h2>
                <img src="{{ url_for('plot') }}" alt="Sorting Algorithm
Performance">
            """, original=arr, bubble=bubble_arr, bubble_time=bubble_time,
               quick=quick_arr, quick_time=quick_time,
               merge=merge_arr, merge_time=merge_time)

        except ValueError:
            return "Invalid input. Please enter a space-separated list of
integers."

    return """
        <h1>Sorting Algorithm Demo</h1>
        <form method="post">
            <label for="elements">Enter numbers:</label><br>
            <input type="text" id="elements" name="elements" required><br><br>
            <input type="submit" value="Sort">
        </form>
        <h2>Performance Graph</h2>
        <img src="{{ url_for('plot') }}" alt="Sorting Algorithm Performance">
    """

@app.route("/plot")
def plot():
    img = plot_sorting_times()
    return send_file(img, mimetype='image/png')

if __name__ == "__main__":
    app.run(debug=True)
```

OUTPUT:

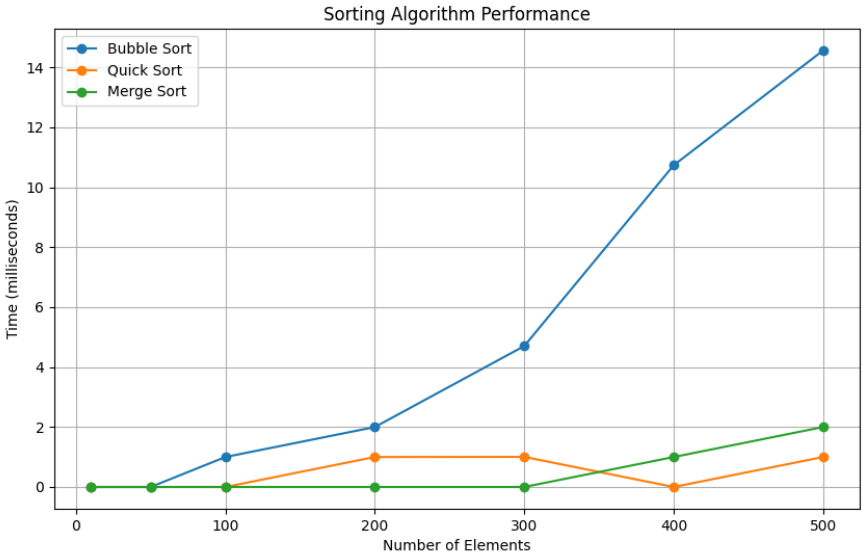

Sign in
127.0.0.1:5000
127.0.0.1:5000
Sorting Algorithm Demo
Enter numbers:
12 8 9 78
Sort
Performance Graph
Sorting Algorithm Performance


Sign in
127.0.0.1:5000
127.0.0.1:5000
Sorted Lists
Original List: [12, 8, 9, 78]
Bubble Sorted List: [8, 9, 12, 78]
Bubble Sort Time: 0.0 milliseconds
Quick Sorted List: [8, 9, 12, 78]
Quick Sort Time: 0.0 milliseconds
Merge Sorted List: [8, 9, 12, 78]
Merge Sort Time: 0.0 milliseconds
Try Again

Sign in 127.0.0.1:5000

127.0.0.1:5000

Performance Graph



Sign in 127.0.0.1:5000

127.0.0.1:5000

Sorted Lists

Original List: [1, 45, 23, 69, 78, 65, 42, 30, 98, 44]

Bubble Sorted List: [1, 23, 30, 42, 44, 45, 65, 69, 78, 98]

Bubble Sort Time: 0.0 milliseconds

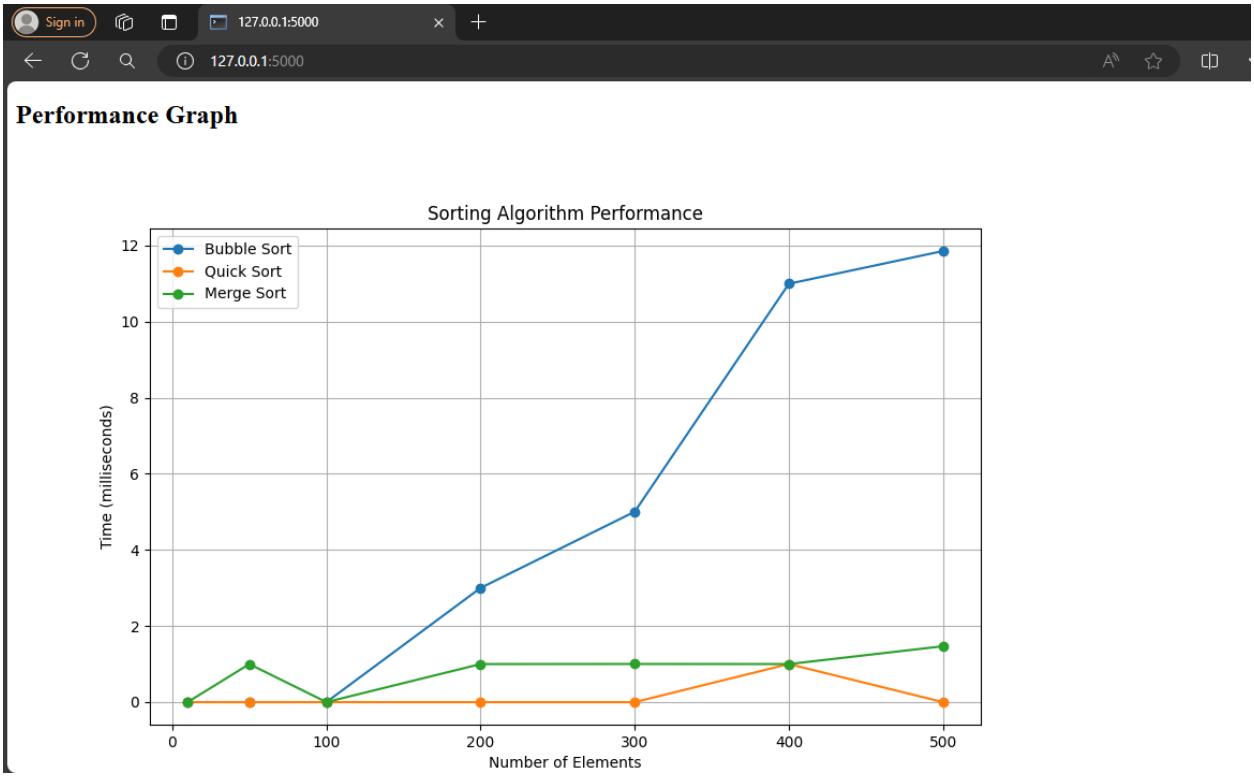
Quick Sorted List: [1, 23, 30, 42, 44, 45, 65, 69, 78, 98]

Quick Sort Time: 0.0 milliseconds

Merge Sorted List: [1, 23, 30, 42, 44, 45, 65, 69, 78, 98]

Merge Sort Time: 0.0 milliseconds

Try Again



Questions:

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

The best case analysis of an algorithm provides the minimum time or space required for the algorithm to complete, given the most favorable input scenario.

The average case analysis provides the expected time or space complexity of an algorithm over all possible inputs, assuming a certain distribution of inputs.

The worst case analysis provides the maximum time or space required for the algorithm to complete, given the most unfavorable input scenario.

2. Which are different asymptotic notations? What is their use?

Big-O Notation (O):

Use: To express the maximum time complexity and to ensure the algorithm's performance does not exceed a certain bound.

Omega Notation (Ω):

Use: To express the minimum time complexity and to ensure the algorithm's performance is not better than a certain bound.

Theta Notation (Θ):

Use: To express the exact growth rate of the algorithm’s time complexity.

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

Bubble Sort:

Best Case:
O(n)

Average Case:
O(n^2)

Worst Case:
O(n^2)

Quick Sort:

Best Case:
O(nlogn)

Average Case:
O(nlogn)

Worst Case:
O(n^2)

Merge Sort:

Best Case:
O(nlogn)

Average Case:
O(nlogn)

Worst Case:
O(nlogn)