NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4

Trigent is an early pioneer in IT outsourcing and offshore software development business.

Thousands of employees working in this company kindly help to find out the employee's details (i.e employee ID, employee salary etc) to implement Recursive Binary search and Linear search (or Sequential Search) and determine the time taken to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

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

Using the algorithm search for the following

1. The designation which has highest salary package
2. The Name of the Employee who has the lowest salary
3. The Mobile number who is youngest employee
4. Salary of the employee who is oldest in age

Code:-

```
from flask import Flask, render_template, request
import random
import time
import matplotlib.pyplot as plt
import os

app = Flask(__name__)

# Step 1: Generate Random Employee Data
def generate_employee_data(n):
    employees = []
    for i in range(n):
        employee = {
            "EmployeeID": i + 1,
            "Name": f"Employee_{i + 1}",
            "Salary": random.randint(30000, 150000),
```

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4

```
            "Age": random.randint(22, 60),
            "Mobile": f"9{random.randint(100000000,
999999999)}",
            "Designation": random.choice(["Developer",
"Manager", "Analyst", "HR", "Consultant"])
        }
        employees.append(employee)
    return employees

# Step 2: Implement Recursive Binary Search
def recursive_binary_search(arr, low, high, key, key_field):
    if high >= low:
        mid = (high + low) // 2
        if arr[mid][key_field] == key:
            return arr[mid]
        elif arr[mid][key_field] > key:
            return recursive_binary_search(arr, low, mid -
1, key, key_field)
        else:
            return recursive_binary_search(arr, mid + 1,
high, key, key_field)
    else:
        return None

# Step 3: Implement Linear Search
def linear_search(arr, key, key_field):
    for item in arr:
        if item[key_field] == key:
            return item
    return None

# Step 4: Measure Time Complexity and Plot Graphs
def measure_time_complexity(employees):
    sizes = [100, 1000, 5000, 10000, 20000, 50000]
    binary_search_times = []
    linear_search_times = []
```

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4

```
    for size in sizes:
        data = employees[:size]
        data.sort(key=lambda x: x['Salary'])  # Sort data
for binary search

        # Measure Recursive Binary Search Time
        start_time = time.time()
        recursive_binary_search(data, 0, len(data) - 1,
data[size // 2]['Salary'], 'Salary')
        end_time = time.time()
        binary_search_times.append(end_time - start_time)

        # Measure Linear Search Time
        start_time = time.time()
        linear_search(data, data[size // 2]['Salary'],
'Salary')
        end_time = time.time()
        linear_search_times.append(end_time - start_time)

    # Plot the time complexity graph

    plt.figure()
    plt.plot(sizes, binary_search_times, label='Recursive
Binary Search')
    plt.plot(sizes, linear_search_times, label='Linear
Search')
    plt.xlabel('Number of Elements (n)')
    plt.ylabel('Time Taken (seconds)')
    plt.title('Time Complexity of Search Algorithms')
    plt.legend()
    plt.grid(True)
    plt.savefig('static/search_time_complexity.png')  # Save
the plot to the static folder
    plt.close()

# Step 5: Perform Specific Searches
def perform_specific_searches(employees):
```

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4

```
    highest_salary_employee = max(employees, key=lambda x:
x['Salary'])
    lowest_salary_employee = min(employees, key=lambda x:
x['Salary'])
    youngest_employee = min(employees, key=lambda x:
x['Age'])
    oldest_employee = max(employees, key=lambda x: x['Age'])

    return highest_salary_employee, lowest_salary_employee,
youngest_employee, oldest_employee

@app.route('/', methods=['GET', 'POST'])
def index():
    result = None
    if request.method == 'POST':
        try:
            num_employees =
int(request.form.get('num_employees', 50000))
            employees =
generate_employee_data(num_employees)
            measure_time_complexity(employees)  # Generate
and save the plot

            # Perform Specific Searches
            highest_salary_employee, lowest_salary_employee,
youngest_employee, oldest_employee =
perform_specific_searches(employees)

            result = {
                "highest_salary_employee":
highest_salary_employee,
                "lowest_salary_employee":
lowest_salary_employee,
                "youngest_employee": youngest_employee,
                "oldest_employee": oldest_employee
            }
        except Exception as e:
```

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4

```
            result = {"error": str(e)}

    return render_template('index.html', result=result)

if __name__ == "__main__":
    # Ensure 'static' directory exists for serving the plot
image
    os.makedirs('static', exist_ok=True)
    app.run(debug=True)
```

html code:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Employee Search Results</title>
</head>
<body>
    <h1>Employee Search Results</h1>
    <form method="POST">
        <label for="num_employees">Number of Employees to
Generate:</label>
        <input type="number" id="num_employees"
name="num_employees" value="50000" min="100" max="50000">
        <button type="submit">Submit</button>
    </form>

    {% if result %}
        {% if result.error %}
            <p style="color:red;">Error: {{ result.error
}}</p>
        {% else %}
            <h2>Highest Salary Employee</h2>
```

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

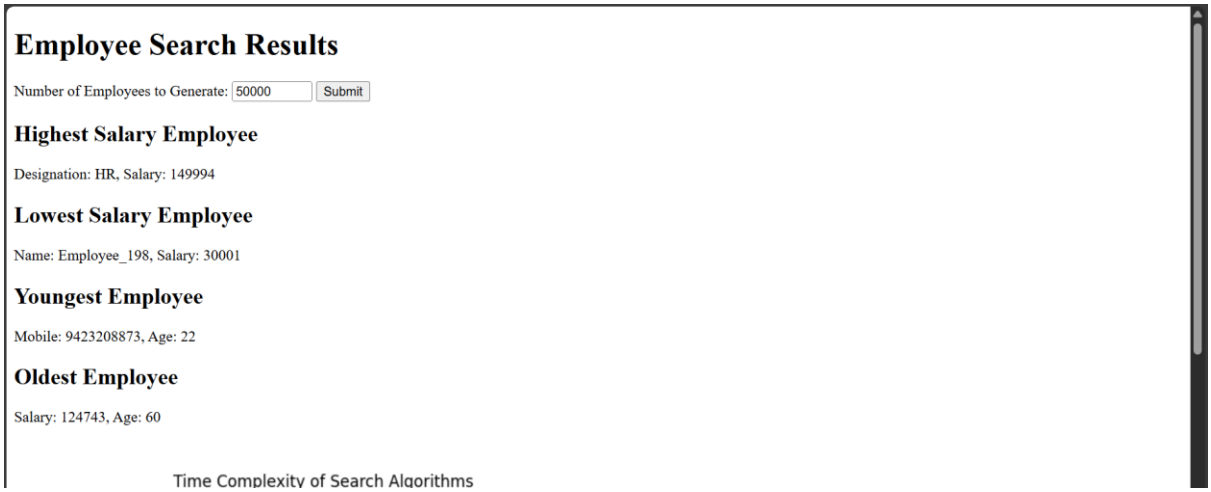
PRACTICAL-4

```
            <p>Designation: {{
result.highest_salary_employee.Designation }}, Salary: {{
result.highest_salary_employee.Salary }}</p>

            <h2>Lowest Salary Employee</h2>
            <p>Name: {{ result.lowest_salary_employee.Name
}}, Salary: {{ result.lowest_salary_employee.Salary }}</p>

            <h2>Youngest Employee</h2>
            <p>Mobile: {{ result.youngest_employee.Mobile
}}, Age: {{ result.youngest_employee.Age }}</p>

            <h2>Oldest Employee</h2>
            <p>Salary: {{ result.oldest_employee.Salary }},
Age: {{ result.oldest_employee.Age }}</p>

            <img src="{{ url_for('static',
filename='search_time_complexity.png') }}" alt='Time
Complexity Graph'>
        {% endif %}
    {% endif %}
</body>
</html>
```

Output:-

NAME:- MARVIN PATEL
ER NO:- 22162101013
BATCH:- 51
SUB :- AAD

PRACTICAL-4



Oldest Employee

Salary: 124743, Age: 60