Prathamsinh **Er no. 22162171004** **Batch: 54**

Institute of Computer Technology
B. Tech Computer Science and Engineering

Sub: Algorithm Analysis and Design

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

Prathamsinh **Er no. 22162171004** **Batch: 54**

CODE:

Python file

```
from flask import Flask, render_template
import time
import random
import matplotlib.pyplot as plt
import io
import baseG4

app = Flask(__name__)

class Employee:
    def __init__(self, emp_id, name, age, salary, designation, mobile):
        self.emp_id = emp_id
        self.name = name
        self.age = age
        self.salary = salary
        self.designation = designation
        self.mobile = mobile

employees = [
    Employee(1, "Hinal", 45, 6000, "Manager", "587G543210"),
    Employee(2, "Navya", 25, 3000, "Developer", "587G543211"),
    Employee(3, "Diya", 22, 2000, "Intern", "587G543212"),
     Employee(4, "Pratham", G0, 1500000, "CEO", "587G543213"),
```

```
    Employee(5, "Khusboo", 18, 70000, "Designer", "587G543214"),
]

def linear_search(employees, target, key):
    for i, emp in enumerate(employees):
        if getattr(emp, key) == target:
            return i
    return -1

def binary_search_recursive(employees, target, key, low, high):
    if high >= low:
        mid = (high + low) // 2
        if getattr(employees[mid], key) == target:
            return mid
        elif getattr(employees[mid], key) > target:
            return binary_search_recursive(employees, target, key, low, mid - 1)
        else:
            return binary_search_recursive(employees, target, key, mid + 1, high)
    else:
        return -1

def measure_time(search_func, employees, target, key, *args):
    start_time = time.time()
    result = search_func(employees, target, key, *args)
    end_time = time.time()
    return result, end_time - start_time
```

```
@app.route('/')
def index():
    highest_salary = max(employees, key=lambda x: x.salary).salary
    index_high, linear_time_high = measure_time(linear_search, employees, highest_salary, "salary")
    highest_salary_designation = employees[index_high].designation

    lowest_salary = min(employees, key=lambda x: x.salary).salary
    index_low, linear_time_low = measure_time(linear_search, employees, lowest_salary, "salary")
    lowest_salary_name = employees[index_low].name

    youngest_age = min(employees, key=lambda x: x.age).age
    index_young, linear_time_young = measure_time(linear_search, employees, youngest_age, "age")
    youngest_mobile = employees[index_young].mobile

    oldest_age = max(employees, key=lambda x: x.age).age
    index_old, linear_time_old = measure_time(linear_search, employees, oldest_age, "age")
    oldest_salary = employees[index_old].salary

    sizes = [10, 100, 500, 1000, 5000, 10000]
    linear_times = []
    binary_times = []

    for size in sizes:
        sample_employees = random.sample(employees * (size // len(employees)), size)
        sample_employees.sort(key=lambda x: x.salary)

        target = sample_employees[-1].salary
```

```
        _, linear_time = measure_time(linear_search, sample_employees, target, "salary")
        _, binary_time = measure_time(binary_search_recursive, sample_employees, target, "salary", 0,
len(sample_employees) - 1)

        linear_times.append(linear_time)
        binary_times.append(binary_time)

    plt.plot(sizes, linear_times, label="Linear Search")
    plt.plot(sizes, binary_times, label="Binary Search")
    plt.xlabel('Number of Elements')
    plt.ylabel('Time Taken (seconds)')
    plt.title('Time Taken vs Number of Elements')
    plt.legend()

    img = io.BytesIO()
    plt.savefig(img, format='png')
    img.seek(0)
    plot_url = baseG4.bG4encode(img.getvalue()).decode('utf8')

    return render_template('index.html',
                           plot_url=plot_url,
                           highest_salary_designation=highest_salary_designation,
                           lowest_salary_name=lowest_salary_name,
                           youngest_mobile=youngest_mobile,
                           oldest_salary=oldest_salary)
```

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

Index.html file:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Employee Search Analysis</title>
</head>

<body>
    <h1>Employee Search Analysis</h1>

    <h2>Results</h2>
    <ul>
        <li><strong>Designation with the Highest Salary:</strong> {{ highest_salary_designation }}</li>
        <li><strong>Employee with the Lowest Salary:</strong> {{ lowest_salary_name }}</li>
        <li><strong>Mobile Number of the Youngest Employee:</strong> {{ youngest_mobile }}</li>
        <li><strong>Salary of the Oldest Employee:</strong> {{ oldest_salary }}</li>
    </ul>

    <h2>Search Time Analysis</h2>
    <img src="data:image/png;baseG4,{{ plot_url }}" alt="Time Taken vs Number of Elements">
```

```
</body>


</html>
```

OUTPUT: