Name : Ayush Patel   Enrolment: 22162171038 Class B Batch 55

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

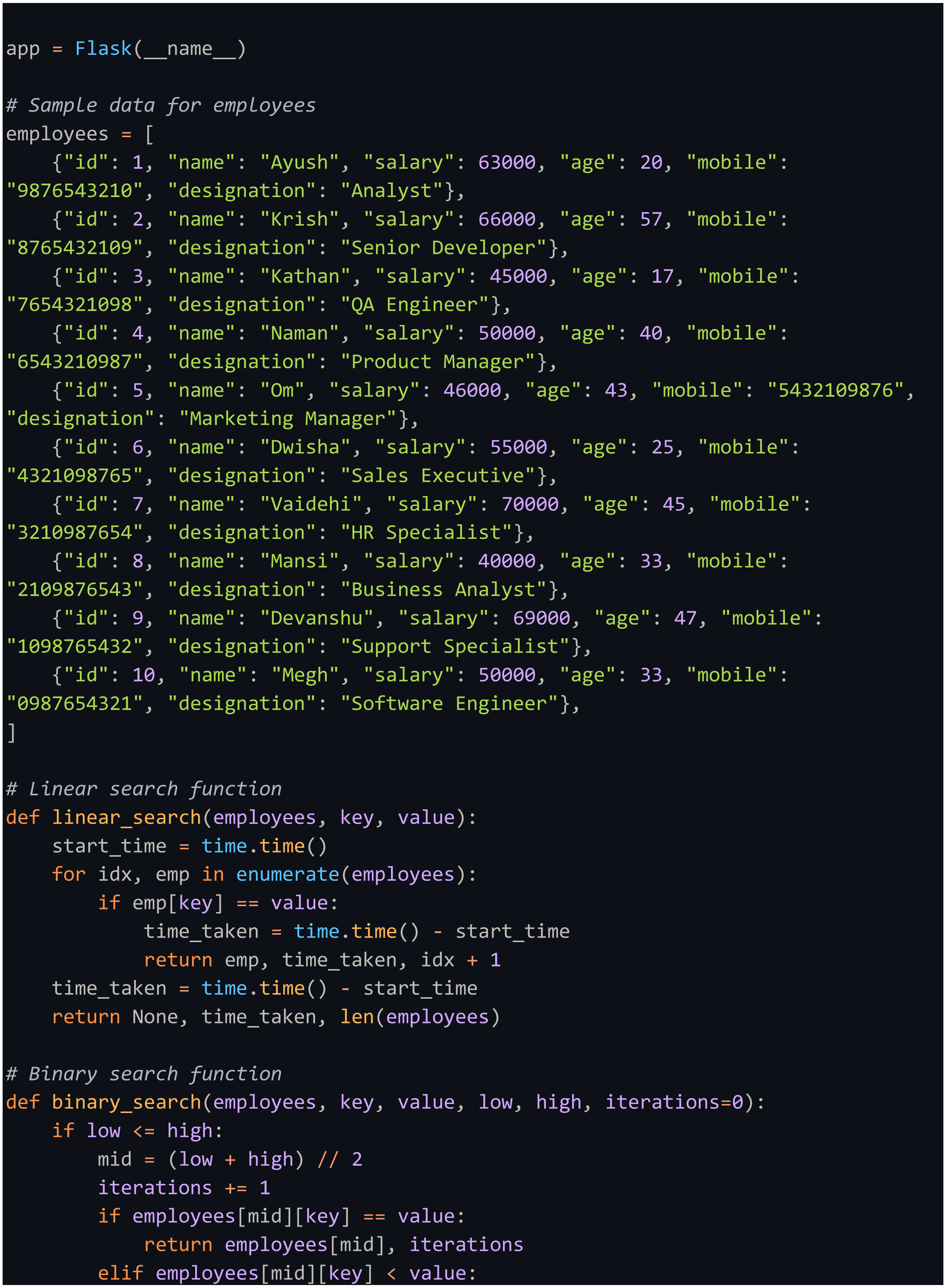
1. The designation which has highest salary package
2. The Name of the Employee who has the lowest salary
3. The Mobile number who is youngest employee
4. Salary of the employee who is oldest in age

Code:

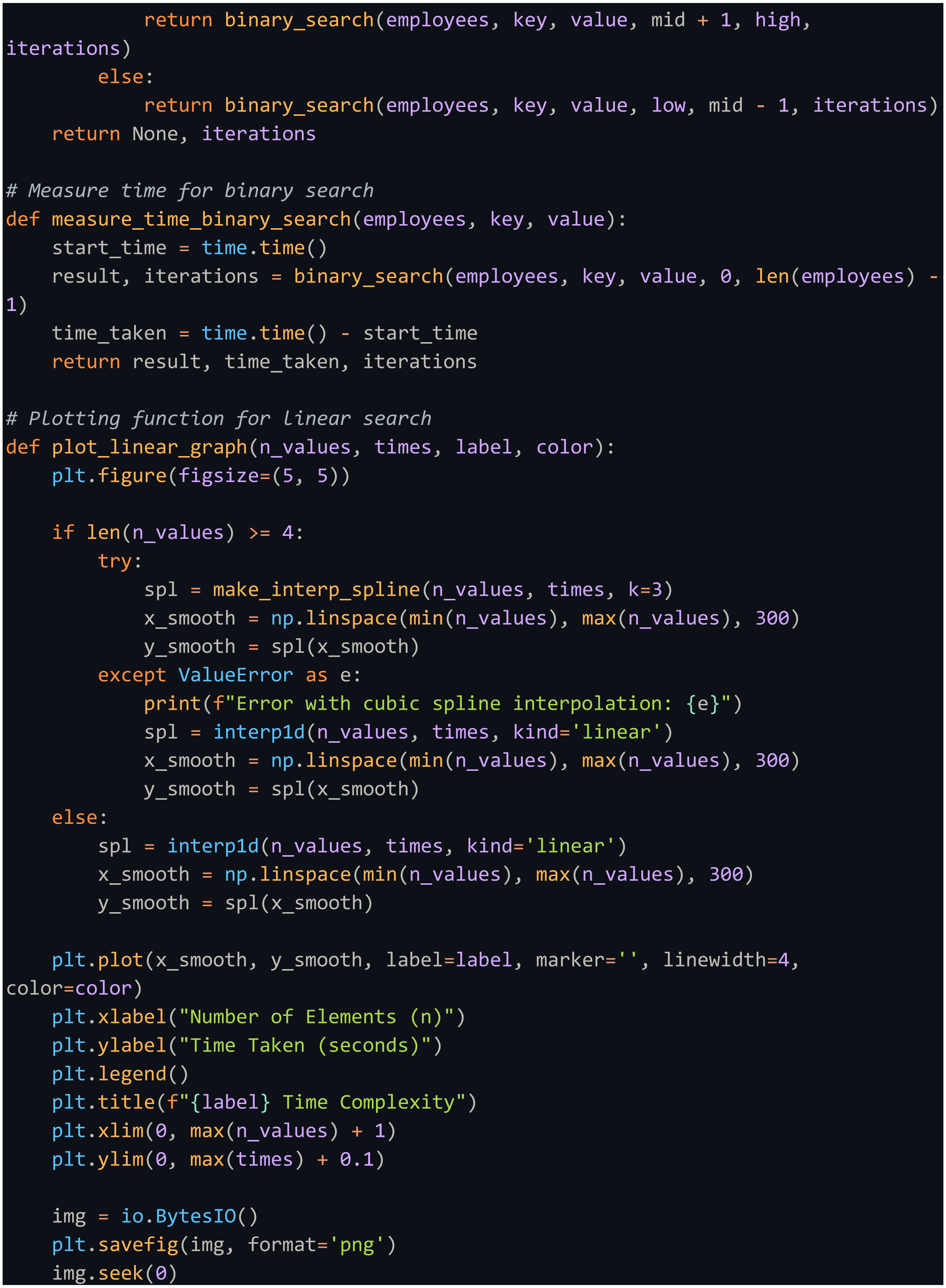
App.py

```
from flask import Flask, render_template, request, redirect, url_for
import time
import matplotlib.pyplot as plt
import io
import base64
import numpy as np
from scipy.interpolate import interp1d
from scipy.interpolate import make_interp_spline
```

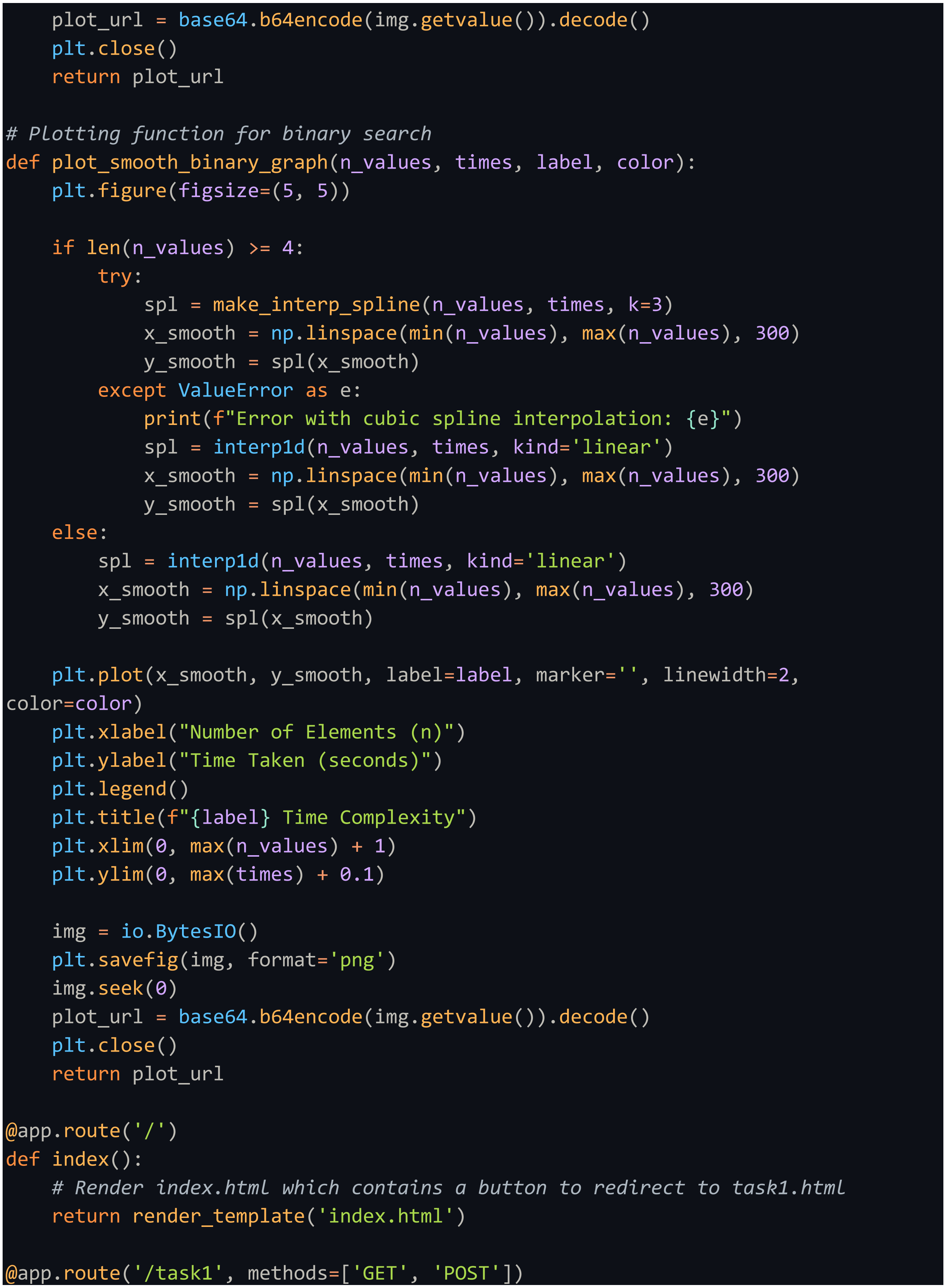
Name : Ayush Patel   Enrolment: 22162171038 Class B Batch 55



```
app = Flask(__name__)

# Sample data for employees
employees = [
    {"id": 1, "name": "Ayush", "salary": 63000, "age": 20, "mobile":
"9876543210", "designation": "Analyst"},
    {"id": 2, "name": "Krish", "salary": 66000, "age": 57, "mobile":
"8765432109", "designation": "Senior Developer"},
    {"id": 3, "name": "Kathan", "salary": 45000, "age": 17, "mobile":
"7654321098", "designation": "QA Engineer"},
    {"id": 4, "name": "Naman", "salary": 50000, "age": 40, "mobile":
"6543210987", "designation": "Product Manager"},
    {"id": 5, "name": "Om", "salary": 46000, "age": 43, "mobile": "5432109876",
"designation": "Marketing Manager"},
    {"id": 6, "name": "Dwisha", "salary": 55000, "age": 25, "mobile":
"4321098765", "designation": "Sales Executive"},
    {"id": 7, "name": "Vaidehi", "salary": 70000, "age": 45, "mobile":
"3210987654", "designation": "HR Specialist"},
    {"id": 8, "name": "Mansi", "salary": 40000, "age": 33, "mobile":
"2109876543", "designation": "Business Analyst"},
    {"id": 9, "name": "Devanshu", "salary": 69000, "age": 47, "mobile":
"1098765432", "designation": "Support Specialist"},
    {"id": 10, "name": "Megh", "salary": 50000, "age": 33, "mobile":
"0987654321", "designation": "Software Engineer"},
]

# Linear search function
def linear_search(employees, key, value):
    start_time = time.time()
    for idx, emp in enumerate(employees):
        if emp[key] == value:
            time_taken = time.time() - start_time
            return emp, time_taken, idx + 1
    time_taken = time.time() - start_time
    return None, time_taken, len(employees)

# Binary search function
def binary_search(employees, key, value, low, high, iterations=0):
    if low <= high:
        mid = (low + high) // 2
        iterations += 1
        if employees[mid][key] == value:
            return employees[mid], iterations
        elif employees[mid][key] < value:
```



```
            return binary_search(employees, key, value, mid + 1, high,
iterations)
        else:
            return binary_search(employees, key, value, low, mid - 1, iterations)
    return None, iterations

# Measure time for binary search
def measure_time_binary_search(employees, key, value):
    start_time = time.time()
    result, iterations = binary_search(employees, key, value, 0, len(employees) -
1)
    time_taken = time.time() - start_time
    return result, time_taken, iterations

# Plotting function for linear search
def plot_linear_graph(n_values, times, label, color):
    plt.figure(figsize=(5, 5))

    if len(n_values) >= 4:
        try:
            spl = make_interp_spline(n_values, times, k=3)
            x_smooth = np.linspace(min(n_values), max(n_values), 300)
            y_smooth = spl(x_smooth)
        except ValueError as e:
            print(f"Error with cubic spline interpolation: {e}")
            spl = interp1d(n_values, times, kind='linear')
            x_smooth = np.linspace(min(n_values), max(n_values), 300)
            y_smooth = spl(x_smooth)
    else:
        spl = interp1d(n_values, times, kind='linear')
        x_smooth = np.linspace(min(n_values), max(n_values), 300)
        y_smooth = spl(x_smooth)

    plt.plot(x_smooth, y_smooth, label=label, marker='', linewidth=4,
color=color)
    plt.xlabel("Number of Elements (n)")
    plt.ylabel("Time Taken (seconds)")
    plt.legend()
    plt.title(f"{label} Time Complexity")
    plt.xlim(0, max(n_values) + 1)
    plt.ylim(0, max(times) + 0.1)

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

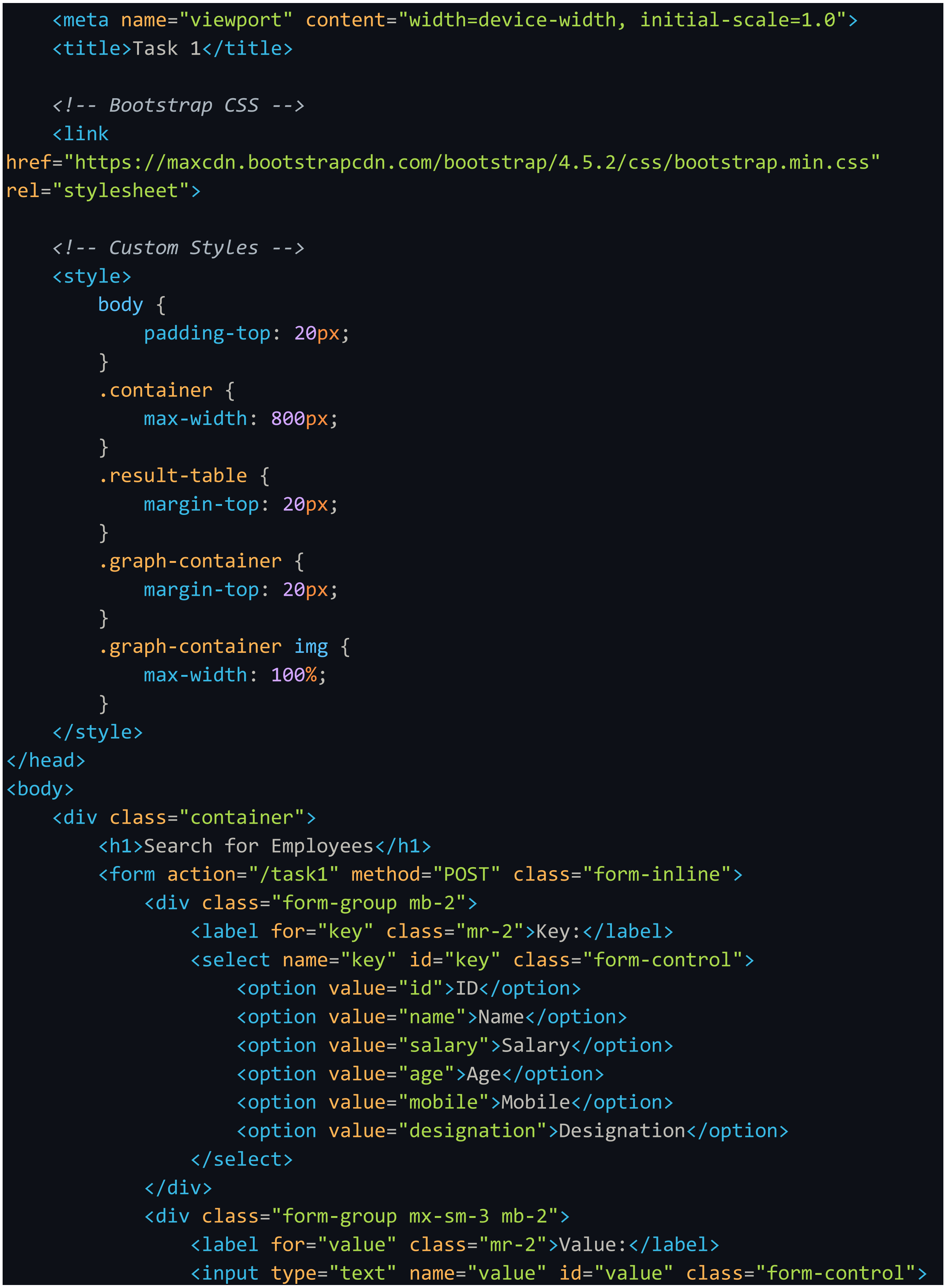


```
    plot_url = base64.b64encode(img.getvalue()).decode()
    plt.close()
    return plot_url

# Plotting function for binary search
def plot_smooth_binary_graph(n_values, times, label, color):
    plt.figure(figsize=(5, 5))

    if len(n_values) >= 4:
        try:
            spl = make_interp_spline(n_values, times, k=3)
            x_smooth = np.linspace(min(n_values), max(n_values), 300)
            y_smooth = spl(x_smooth)
        except ValueError as e:
            print(f"Error with cubic spline interpolation: {e}")
            spl = interp1d(n_values, times, kind='linear')
            x_smooth = np.linspace(min(n_values), max(n_values), 300)
            y_smooth = spl(x_smooth)
    else:
        spl = interp1d(n_values, times, kind='linear')
        x_smooth = np.linspace(min(n_values), max(n_values), 300)
        y_smooth = spl(x_smooth)

    plt.plot(x_smooth, y_smooth, label=label, marker='', linewidth=2,
color=color)
    plt.xlabel("Number of Elements (n)")
    plt.ylabel("Time Taken (seconds)")
    plt.legend()
    plt.title(f"{label} Time Complexity")
    plt.xlim(0, max(n_values) + 1)
    plt.ylim(0, max(times) + 0.1)

    img = io.BytesIO()
    plt.savefig(img, format='png')
    img.seek(0)
    plot_url = base64.b64encode(img.getvalue()).decode()
    plt.close()
    return plot_url

@app.route('/')
def index():
    # Render index.html which contains a button to redirect to task1.html
    return render_template('index.html')

@app.route('/task1', methods=['GET', 'POST'])
```

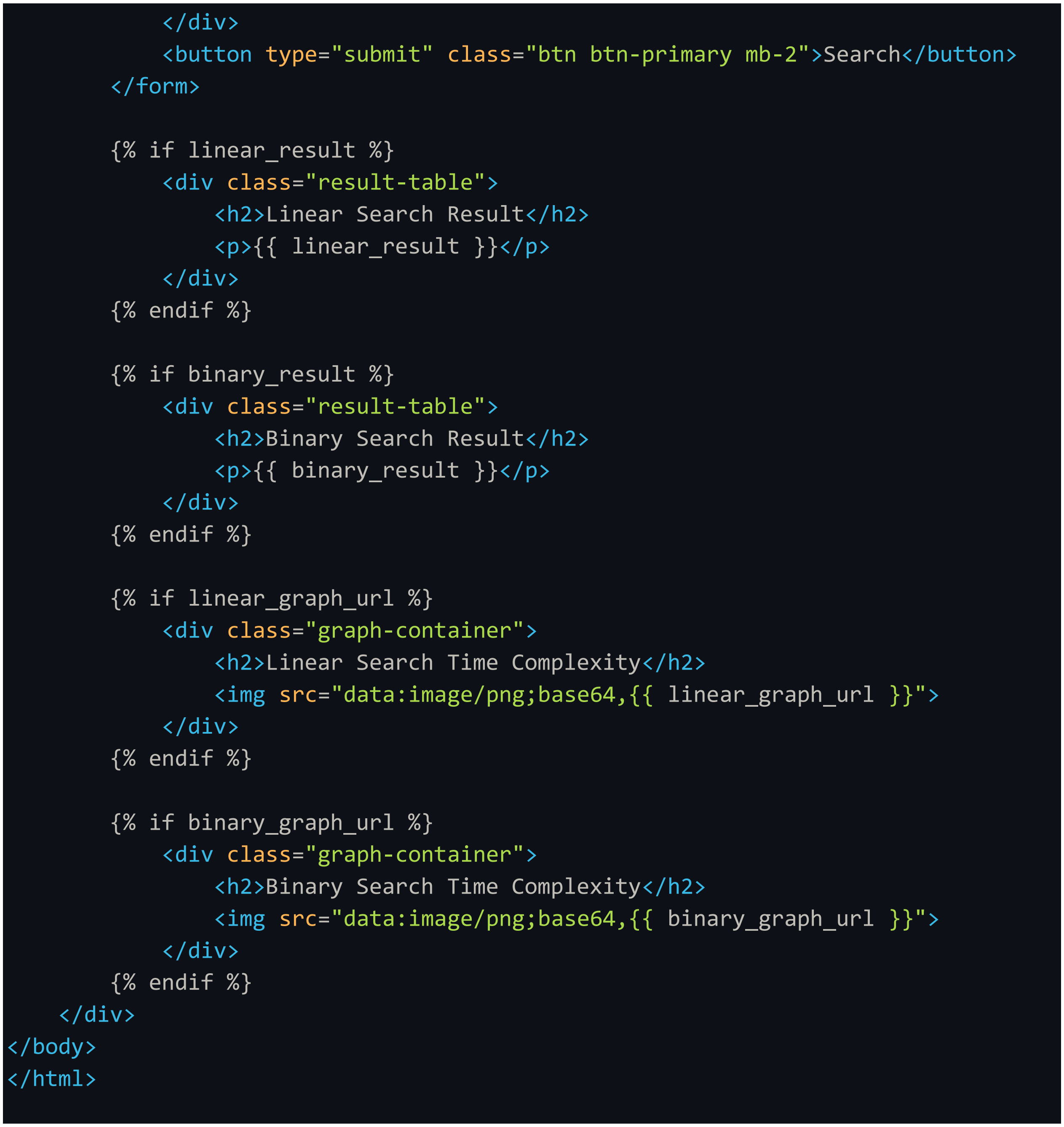
```
def task1():
    if request.method == 'POST':
        key = request.form['key']
        value = request.form['value']

        if key in ['id', 'salary', 'age']:
            value = int(value)

        linear_result, linear_time, linear_iterations = linear_search(employees,
key, value)

        sorted_employees = sorted(employees, key=lambda x: x[key])
        binary_result, binary_time, binary_iterations =
measure_time_binary_search(sorted_employees, key, value)

        n_values = list(range(0, len(employees) + 1))

        linear_times = [(linear_iterations / len(employees)) * linear_time for _
in n_values]

        binary_times = [(binary_iterations / len(employees)) * np.log2(n) if n >
0 else 0 for n in n_values]

        linear_graph_url = plot_linear_graph(n_values, linear_times, "Linear
Search (O(n))", 'red')
        binary_graph_url = plot_smooth_binary_graph(n_values, binary_times,
"Binary Search (O(log n))", 'blue')

        return render_template('task1.html', linear_result=linear_result,
binary_result=binary_result,
                               linear_graph_url=linear_graph_url,
binary_graph_url=binary_graph_url)
    return render_template('task1.html')

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

Task1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```



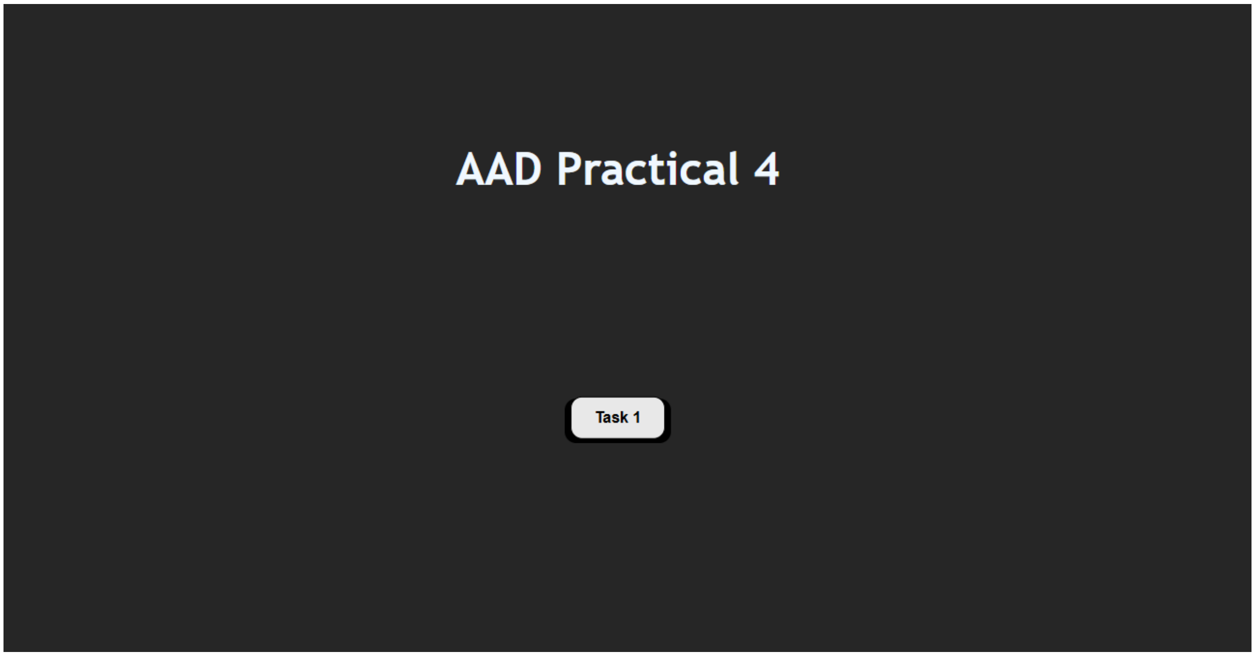
```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task 1</title>

    <!-- Bootstrap CSS -->
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">

    <!-- Custom Styles -->
    <style>
        body {
            padding-top: 20px;
        }
        .container {
            max-width: 800px;
        }
        .result-table {
            margin-top: 20px;
        }
        .graph-container {
            margin-top: 20px;
        }
        .graph-container img {
            max-width: 100%;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Search for Employees</h1>
        <form action="/task1" method="POST" class="form-inline">
            <div class="form-group mb-2">
                <label for="key" class="mr-2">Key:</label>
                <select name="key" id="key" class="form-control">
                    <option value="id">ID</option>
                    <option value="name">Name</option>
                    <option value="salary">Salary</option>
                    <option value="age">Age</option>
                    <option value="mobile">Mobile</option>
                    <option value="designation">Designation</option>
                </select>
            </div>
            <div class="form-group mx-sm-3 mb-2">
                <label for="value" class="mr-2">Value:</label>
                <input type="text" name="value" id="value" class="form-control">
```



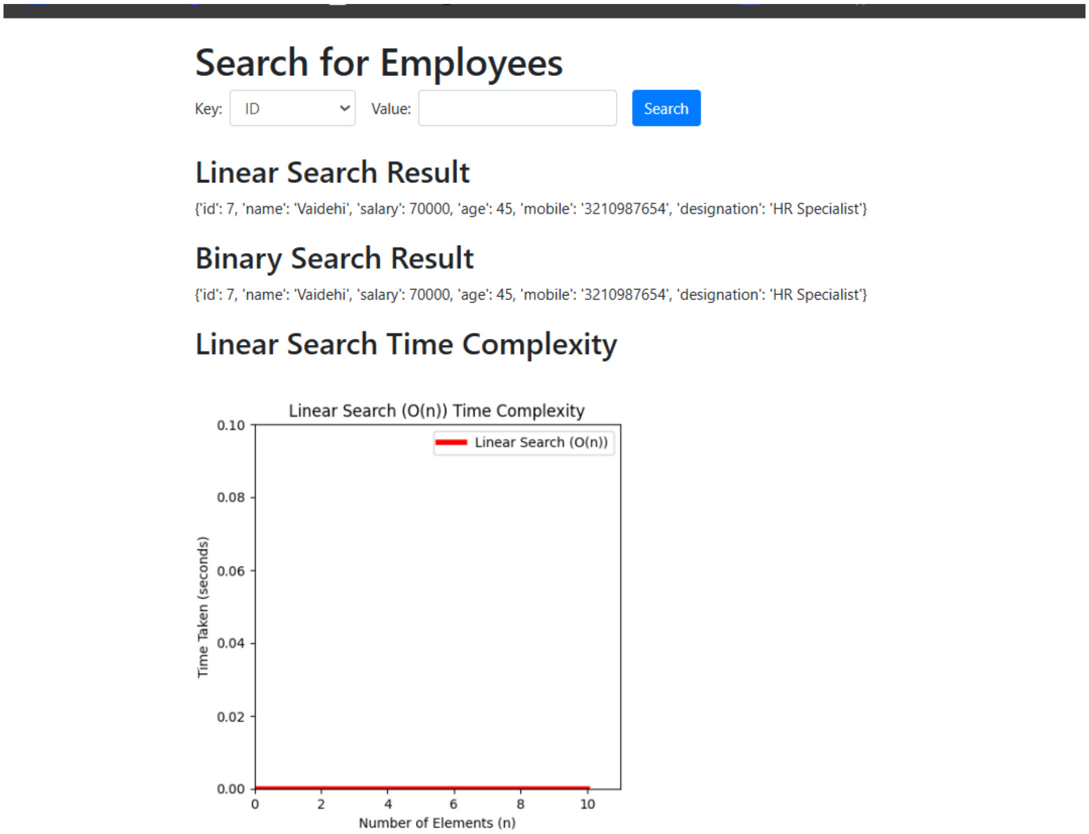
```
            </div>
            <button type="submit" class="btn btn-primary mb-2">Search</button>
        </form>

        {% if linear_result %}
            <div class="result-table">
                <h2>Linear Search Result</h2>
                <p>{{ linear_result }}</p>
            </div>
        {% endif %}

        {% if binary_result %}
            <div class="result-table">
                <h2>Binary Search Result</h2>
                <p>{{ binary_result }}</p>
            </div>
        {% endif %}

        {% if linear_graph_url %}
            <div class="graph-container">
                <h2>Linear Search Time Complexity</h2>
                <img src="data:image/png;base64,{{ linear_graph_url }}">
            </div>
        {% endif %}

        {% if binary_graph_url %}
            <div class="graph-container">
                <h2>Binary Search Time Complexity</h2>
                <img src="data:image/png;base64,{{ binary_graph_url }}">
            </div>
        {% endif %}
    </div>
</body>
</html>
```

Name : Ayush Patel   Enrolment: 22162171038 Class B Batch 55

Output:



AAD Practical 4



Search for Employees

Key: ID Value:

Linear Search Result

{'id': 7, 'name': 'Vaidehi', 'salary': 70000, 'age': 45, 'mobile': '3210987654', 'designation': 'HR Specialist'}

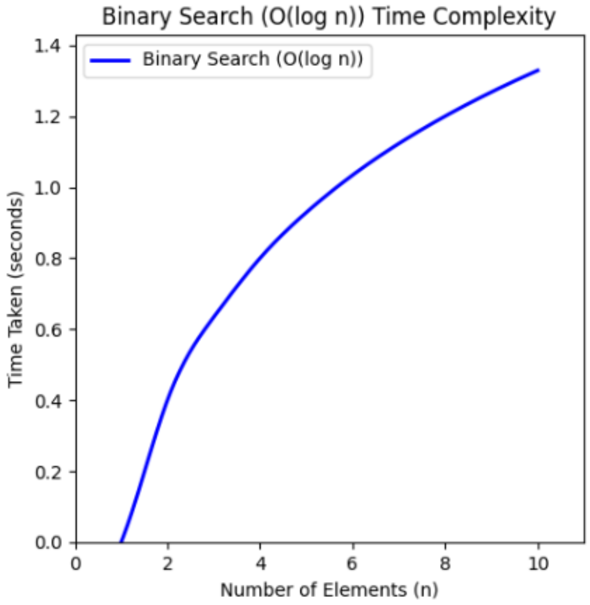
Binary Search Result

{'id': 7, 'name': 'Vaidehi', 'salary': 70000, 'age': 45, 'mobile': '3210987654', 'designation': 'HR Specialist'}

Linear Search Time Complexity

Name : Ayush Patel Enrolment: 22162171038 Class B Batch 55

Binary Search Time Complexity


Binary Search (O(log n)) Time Complexity
Binary Search (O(log n))
Time Taken (seconds)
Number of Elements (n)