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**

App.py

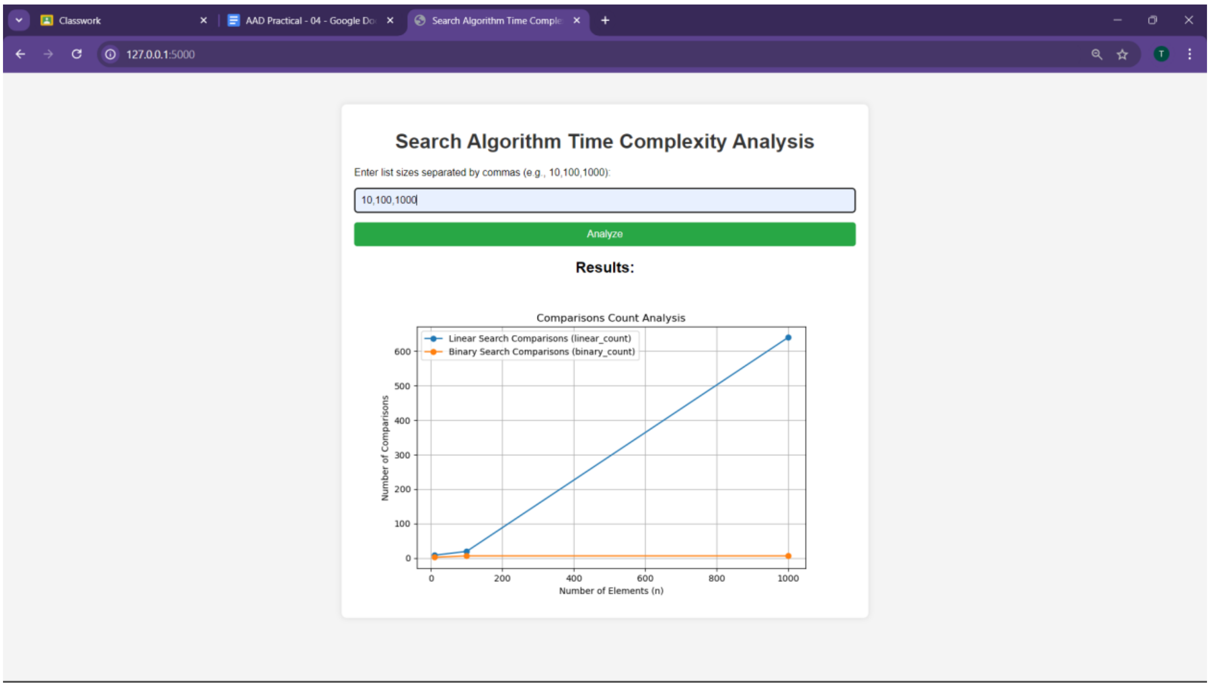
```
from flask import Flask, render_template, request

import matplotlib.pyplot as plt

import random

import io

import base64
```

```
plt.switch_backend('Agg')


app = Flask(__name__)


def linear_search(arr, x):
    linear_count = 0
    for i in range(len(arr)):
        linear_count += 1
        if arr[i] == x:
            return i, linear_count
    return -1, linear_count


def binary_search(arr, x, low, high, binary_count=0):
    if high >= low:
        mid = (high + low) // 2
        binary_count += 1

        # If element is present at the middle
        if arr[mid] == x:
            return mid, binary_count

        # If element is smaller than mid, search in left subarray
        elif arr[mid] > x:
            return binary_search(arr, x, low, mid - 1, binary_count)

        # If element is larger than mid, search in right subarray
        else:
            return binary_search(arr, x, mid + 1, high, binary_count)
    else:
        # Element is not present in the array
        return -1, binary_count

# Measure comparisons for each search algorithm
def measure_comparisons(search_function, arr, x):
    _, comparisons = search_function(arr, x)
    return comparisons

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        sizes = request.form.get('sizes')

        if not sizes:
            return render_template('index.html', error="Please enter valid sizes separated by 
commas.")

        sizes = list(map(int, sizes.split(',')))  # Ensure sizes is not None before splitting
        linear_comparisons = []
        binary_comparisons = []

        for size in sizes:
            arr = sorted(random.sample(range(size * 2), size))
            target = random.choice(arr)

            # Measure comparisons for Linear Search
            linear_comparison = measure_comparisons(lambda arr, target: linear_search(arr, 
target), arr, target)
```

```
            linear_comparisons.append(linear_comparison)

            # Measure comparisons for Binary Search
            binary_comparison = measure_comparisons(lambda arr, target: binary_search(arr, target, 0, len(arr) - 1), arr, target)
            binary_comparisons.append(binary_comparison)

        # Plot the comparison results
        plt.figure(figsize=(8, 5))
        plt.plot(sizes, linear_comparisons, label="Linear Search Comparisons (linear_count)", marker='o')
        plt.plot(sizes, binary_comparisons, label="Binary Search Comparisons (binary_count)", marker='o')
        plt.xlabel("Number of Elements (n)")
        plt.ylabel("Number of Comparisons")
        plt.title("Comparisons Count Analysis")
        plt.legend()
        plt.grid(True)

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

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

    return render_template('index.html')

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

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Search Algorithm Time Complexity Analysis</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f4;
    }
    .container {
      max-width: 800px;
      margin: 50px auto;
      padding: 20px;
      background-color: #fff;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    h1 {
      text-align: center;
      color: #333;
    }
    form {
      display: flex;
      flex-direction: column;
      gap: 15px;
    }
    input[type="text"] {
      padding: 10px;
      font-size: 16px;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    input[type="submit"] {
      padding: 10px;
      font-size: 16px;
      background-color: #28a745;
      color: #fff;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }
    input[type="submit"]:hover {
      background-color: #218838;
    }
    .plot-container {
      text-align: center;
      margin-top: 20px;
    }
    img {
      max-width: 100%;
      height: auto;
      }
    </style>
</head>
<body>
    <div class="container">
      <h1>Search Algorithm Time Complexity Analysis</h1>
      <form method="post">
          <label for="sizes">Enter list sizes separated by commas (e.g., 10,100,1000):</label>
          <input type="text" id="sizes" name="sizes" placeholder="Enter sizes" required>
          <input type="submit" value="Analyze">
      </form>

      {% if plot_url %}
      <div class="plot-container">
          <h2>Results:</h2>
          <img src="data:image/png;base64,{{ plot_url }}" alt="Time Complexity Graph">
      </div>
      {% endif %}
    </div>
</body>
</html>
```


Classwork
AAD Practical - 04 - Google Do
Search Algorithm Time Comple
127.0.0.1:5000

Search Algorithm Time Complexity Analysis

Enter list sizes separated by commas (e.g., 10,100,1000):

10,100,1000

Analyze

Results:

Comparisons Count Analysis
Linear Search Comparisons (linear_count)
Binary Search Comparisons (binary_count)
Number of Comparisons
0
100
200
300
400
500
600
0
200
400
600
800
1000
Number of Elements (n)