Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavrepalanchowk

Lab report on **Lab 1**

IMPLEMENTATION, TESTING AND PERFORMANCE MEASUREMENT OF LINEAR SEARCH AND BINARY SEARCH ALGORITHMS

Sub Code: COMP 314

Submitted by:

Saskar Khadka, CE

Roll No: 27

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 12 April, 2023

Linear Search:

A linear search, also known as sequential search is the simplest approach employed to search for an element in a data set. It is a Brute Force algorithm meaning it examines each element in the data set until it finds a match starting at the beginning of the dataset.

Binary Search:

Binary search is used to find an item in a sorted dataset. It works by repeatedly dividing the portion of the dataset that could contain the item in half until the possible location is narrowed down to just one.

1. Implementation of Linear and Binary search.

[Source Code]()

1.1 Linear Search:

```python
# Searches for the target in an unsorted array
# Returns index of the target if it exists in the data else returns -1
def linear_search(data, target):
    for index, each in enumerate(data):
        if each == target:
            return index
    return -1
```

1.2 Binary Search:

```python
# Searches for the target in a binary search tree (sorted array)
# Returns index of the target if it exists in the data else returns -1
def binary_search(data, target):
    low = 0
    high = len(data) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2
        if data[mid] < target:
            low = mid + 1
        elif data[mid] > target:
            high = mid - 1
        else:
            return mid

    return -1
```

1.3 Main Driver:

```python
from graph import plot_graph
from search import linear_search, binary_search

if __name__ == "__main__":
    # 1
    print(f"Index of 2 in given list is {linear_search([1, 5, 3, 2, 7], 2)}")
    print(
        f"Index of 5 in given list is {binary_search([1, 2, 3, 4, 5, 6, 7], 5)}")

    # 3
    plot_graph()
```

1.4 Outputs:

```
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 1> py .\main.py
Index of 2 in given list is 3
Index of 5 in given list is 4
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 1>
```

2. **Write some test cases for your program.**

2.1 Test Cases:

```python
import unittest
from search import linear_search, binary_search


class TestSearch(unittest.TestCase):

    # Test for linear search
    def test_linear_search(self):
        self.assertEqual(linear_search([4, 2, 6, 1, 7, 3], 1), 3)

    # Test for binary search
    def test_binary_search(self):
        data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        self.assertEqual(binary_search(data, 6), 5)


if __name__ == "__main__":
    unittest.main()
```

2.2 Test Output:

```
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 1> py test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
PS C:\Users\Saskarkhadka\OneDrive\Desktop\Algorithms-Labs\Lab 1>
```

3. Plot an input-size vs. execution-time graph for best case and worst case of both of the algorithms

3.1 main.py

```python
from graph import plot_graph
from search import linear_search, binary_search

if __name__ == "__main__":
    # 1
    print(f"Index of 2 in given list is {linear_search([1, 5, 3, 2, 7], 2)}")
    print(
        f"Index of 5 in given list is {binary_search([1, 2, 3, 4, 5, 6, 7], 5)}")

    # 3
    plot_graph()
```

3.2 graph.py

```
import matplotlib.pyplot as plt
import time
from search import linear_search, binary_search
import numpy as np


# Generates random input and calculates the corresponding execution
# time for linear and binary search algorithms
def search_data(bestCase: bool):
    data = {
        "linear_input_size": [],
        "linear_exec_time": [],
        "binary_input_size": [],
        "binary_exec_time": [],
    }
    for size in range(10000, 100000, 10000):
        test_data = np.arange(size+1)

        # For linear search, best case = first data and worst case = invalid data
        data["linear_input_size"].append(size)
        start_time = time.time()
        result = linear_search(test_data, 0 if bestCase else size + 1)
        data["linear_exec_time"].append((time.time() - start_time) * 1000)

        # For binary search, best case = middle data and worst case = invalid data
        data["binary_input_size"].append(size)
        start_time = time.time()
        result = binary_search(
            test_data, (len(test_data) - 1) // 2 if bestCase else size + 1)
        data["binary_exec_time"].append((time.time() - start_time) * 1000)

    return data
```

```python
# Plots input size vs. execution time for best case and worst cases
def plot(ax, bestCase):
    title = "Best Case" if bestCase else "Worst Case"
    data = search_data(bestCase)

    ax.plot(data["binary_input_size"],
            data["binary_exec_time"], label="Binary Search")

    ax.plot(data["linear_input_size"],
            data["linear_exec_time"], label="Linear Search", alpha=0.7, linestyle="dashed")

    ax.set_xlabel("Input Size")
    ax.set_ylabel("Execution Time")

    ax.set_title(title)
    ax.legend()


def plot_graph():
    fig, (ax1, ax2) = plt.subplots(2)
    plot(ax1, bestCase=True)
    plot(ax2, bestCase=False)
    fig.tight_layout()
    plt.show()
```

Output:

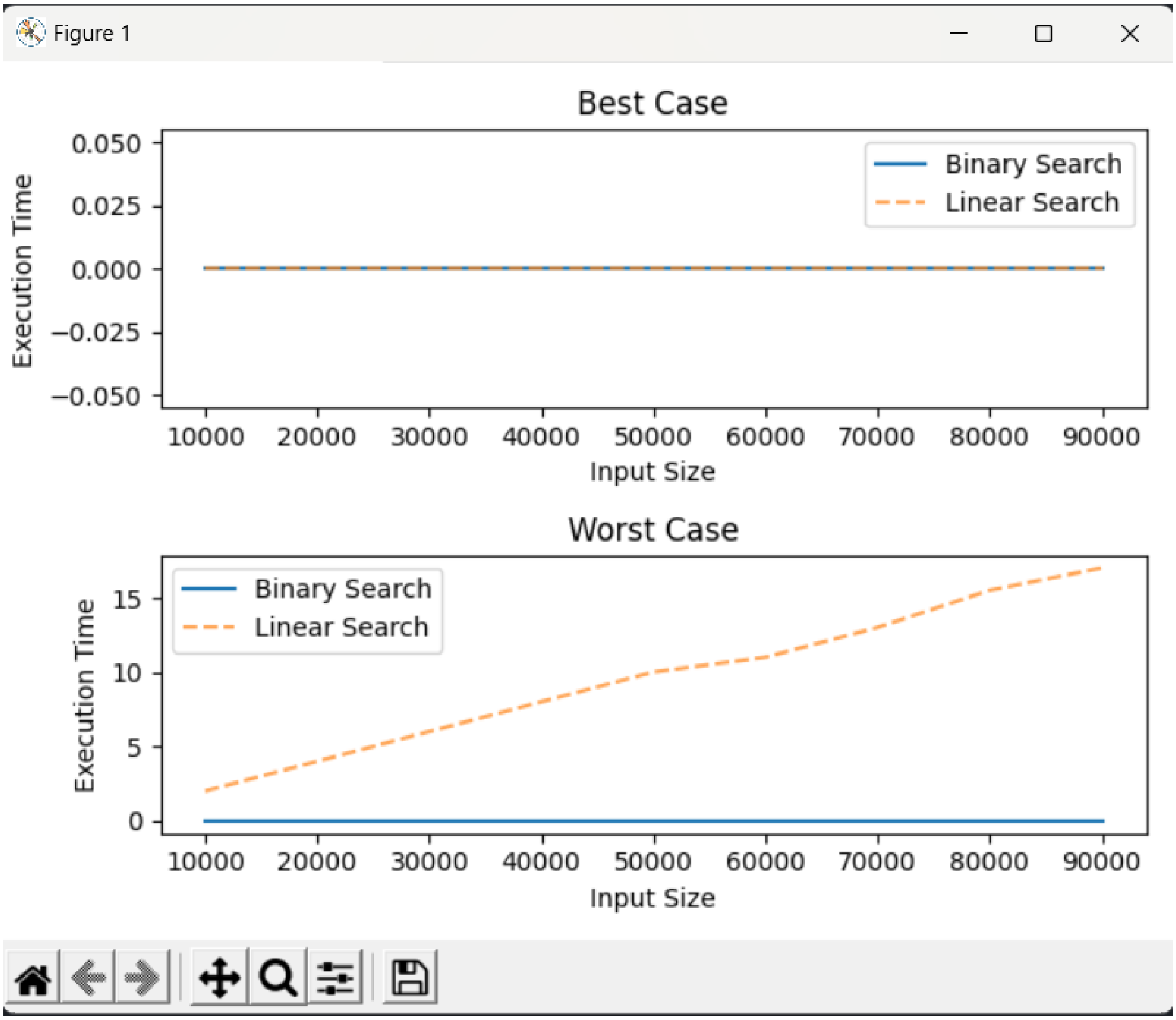

Figure 1
Best Case
Execution Time
Input Size
Binary Search
Linear Search
0.050
0.025
0.000
−0.025
−0.050
10000
20000
30000
40000
50000
60000
70000
80000
90000
Worst Case
Execution Time
Input Size
Binary Search
Linear Search
15
10
5
0