122COM

Introduction to Algorithms

Task 2 - Profiling

Student Name: **Chalumuri Venkata Nag Rithin Naidu**

Student ID: **7030330**

Table of Contents

Introduction 3

Linear Search 3

Binary Search 4

Test 1 5

Test 2 6

Test 3 7

Test 4 8

Discussion of Results 9

INTRODUCTION

Profiling is used to measure the performance of a program. It allows us to understand the behavior of the different parts of program. The simplest way to profile is to time how long the program takes to run, this is also known as Crude Profiling. For this task, I will be using another profiling method (more powerful) known as Deterministic Profiling. Deterministic profiling uses software to externally monitor precisely the time each portion of program takes to run. I will also be using **Gprof2dot and Graphviz** for converting raw profiling information into diagrams. The main purpose of this task is to highlight the comparisons between linear search and binary search algorithms in terms of performance. In order to do so, I will be profiling the results of few test programs written in python, which run both linear and binary search functions.

LINEAR SEARCH

Linear search is straight forward. The algorithm simply iterates over all the elements in the sequence until the value being searched for is found, in other words, it compares the target value with every element in the sequence. It is the easiest searching algorithm to implement. C++ Implementation of linear search algorithm:

```
bool linear_search(vector<int> sequence, int value)
{
  for( int i : sequence )
  {
    if( i == value )
    {
      return true;
    }
  }
  return false;
}
```

Where sequence is the list of elements and value is target value being searched for. This function returns true if the target value if found in the sequence otherwise false.

BINARY SEARCH

Binary search is one of the quickest and efficient algorithms used for searching. It is a divide and conquer algorithm. However, it only works with sorted sequences. The algorithm works by picking the element in the middle of the sequence and compares with target value. If the middle value is equal to the target value, then the element is successfully found. If the picked element is greater than target value, then ignore the bottom half of the sequence. Otherwise, if the picked element is lesser than the target value, then ignore the top half of the sequence. Binary search algorithm repeats this process until length of sequence becomes 0.

C++ Implementation of binary search algorithm:

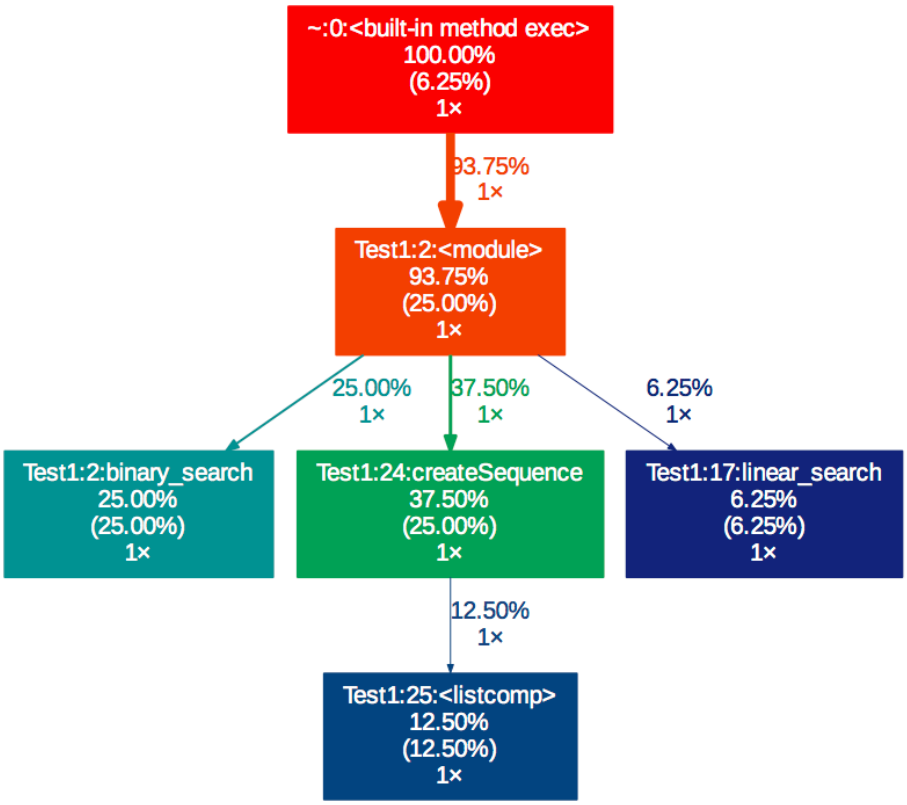
```
bool binary_search(vector<int> sequence, int value)
{
  int start,seqLen,end;
  start = 0;
  seqLen = sequence.size();
  end = seqLen - 1;

  while (start <= end)
  {
    int position;
    position = (start+end)/2;
    if(sequence[position] == value)
    {
      return true;
    }
    else if (sequence[position] < value)
    {
      start = position + 1;
    }
    else if (sequence[position] > value)
    {
      end = position - 1;
    }
  }

  return false;
}
```

TEST 1

In this test the program is going to do both linear and binary search for '33' in a sequence containing multiples of 3 between 0 and 100 (including 0).

Number of elements in sequence: 34

The results are:



Key information to note from the above diagram (Test 1 profiling results):

- The function to create a sequence of 34 elements was called once, and it took 37.50% of the *total time* (including its children).
- The linear search function to search value '33' in created sequence was called once, and it took 6.25% of the *total time.*
- The binary search function to search value '33' in created sequence was called once, and it took 25.00% of the *total time*.

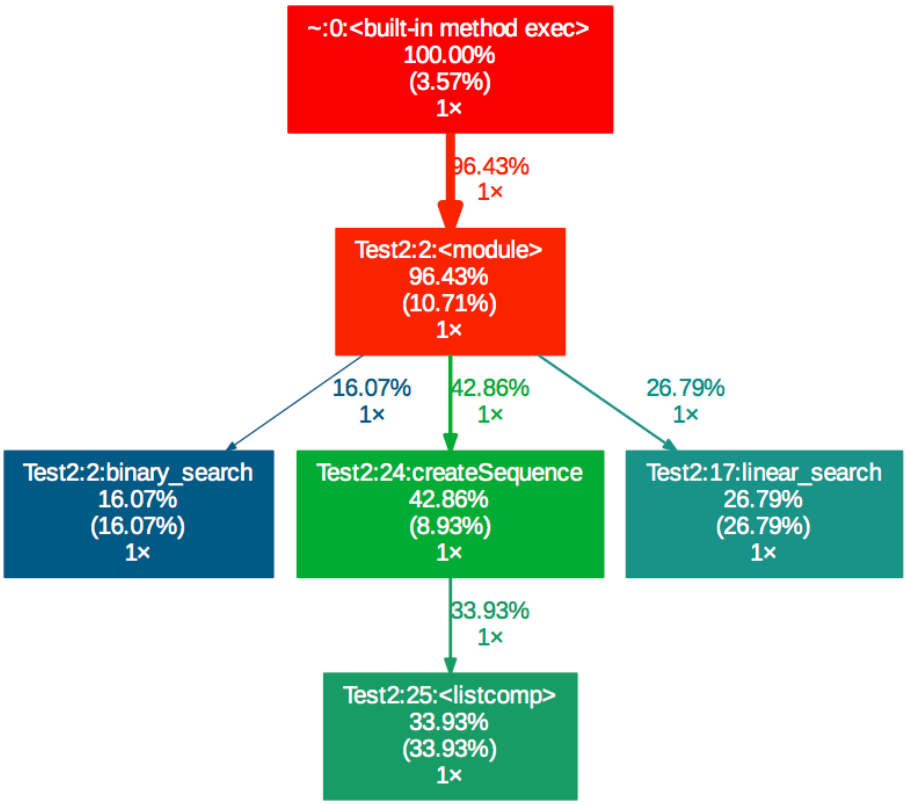
In test 1, linear search worked faster than binary search.

TEST 2

In this test the program is going to do both linear and binary search for ’792’ in a sequence containing multiples of 3 between 0 and 1000 (including 0).

Number of elements in sequence: 334

The results are:



Key information to note from the above diagram (Test 2 profiling results):

- The function to create a sequence of 334 elements was called once, and it took 42.86% of the *total time* (including its children).
- The linear search function to search value ‘792’ in created sequence was called once, and it took 26.79% of the *total time.*
- The binary search function to search value ‘792’ in created sequence was called once, and it took 16.07% of the *total time*.

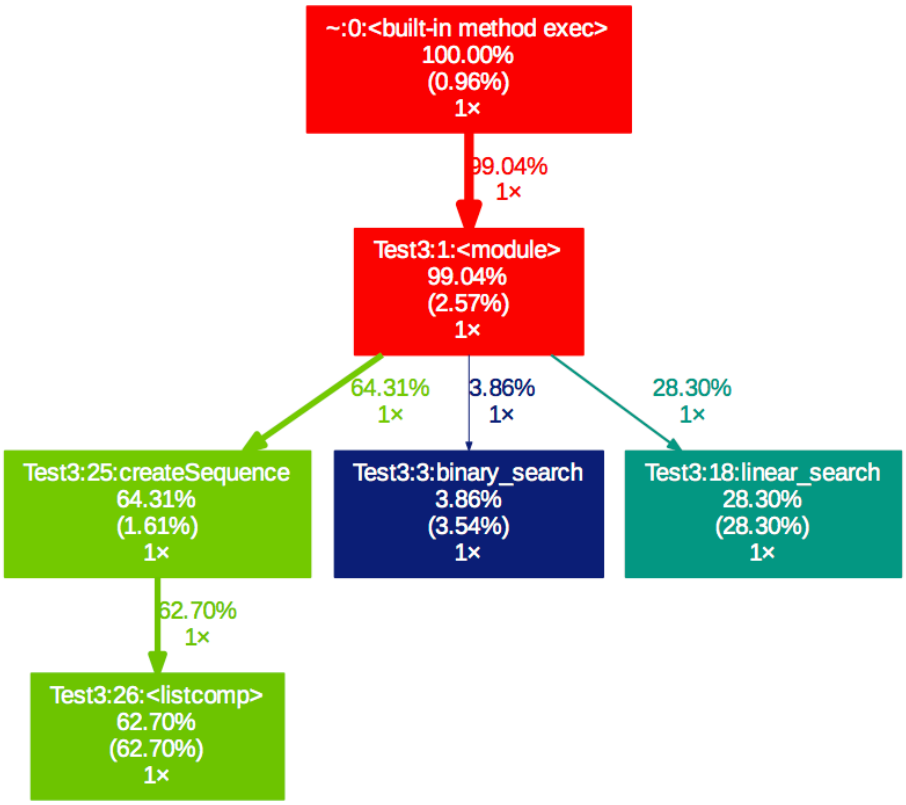
In Test 2 binary search worked faster than linear search.

TEST 3

In this test the program is going to do both linear and binary search for '4884' in a sequence containing multiples of 3 between 0 and 10000 (including 0).

Number of elements in sequence: 3334

The results are:



Key information to note from the above diagram (Test 3 profiling results):

- The function to create a sequence of 3334 elements was called once, and it took 64.31% of the *total time* (including its children).
- The linear search function to search value '4884' in created sequence was called once, and it took 28.30% of the *total time.*
- The binary search function to search value '4884' in created sequence was called once, and it took 3.86% of the *total time*.

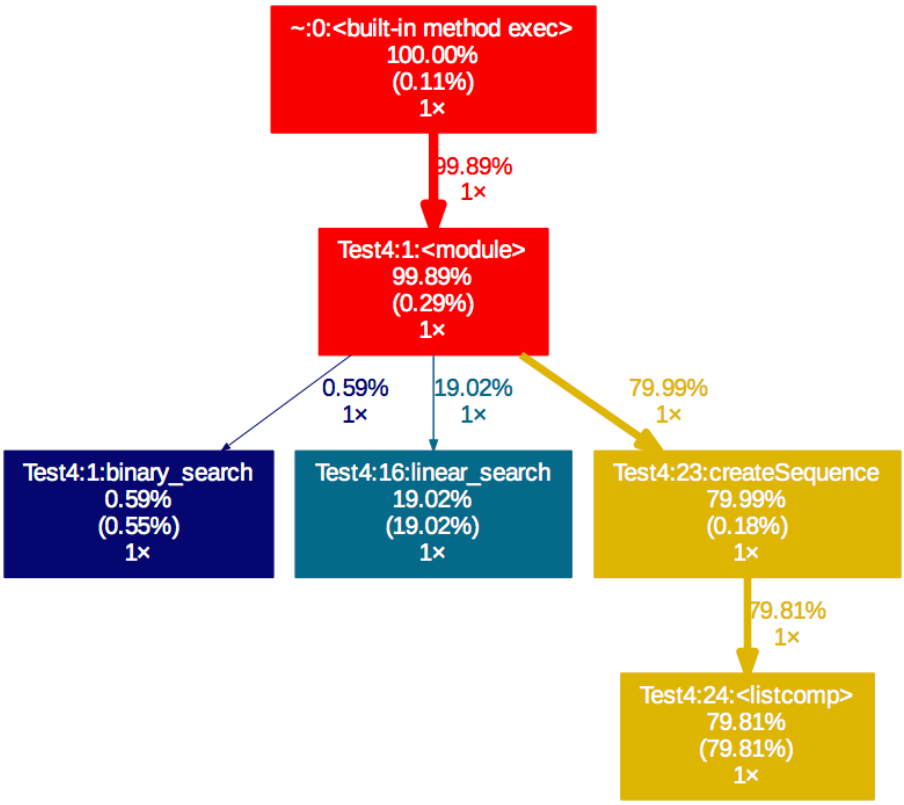
In Test 3 binary search worked faster than linear search.

TEST 4

In this test the program is going to do both linear and binary search for ’29994’ in a sequence containing multiples of 3 between 0 and 100000 (including 0).

Number of elements in sequence: 33334

The results are:



Key information to note from the above diagram (Test 4 profiling results):

- The function to create a sequence of 33334 elements was called once, and it took 79.99% of the *total time* (including its children).
- The linear search function to search value ‘29994’ in created sequence was called once, and it took 19.02% of the *total time.*
- The binary search function to search value ‘29994’ in created sequence was called once, and it took 0.59% of the *total time*.

In Test 4 binary search worked way faster than linear search.

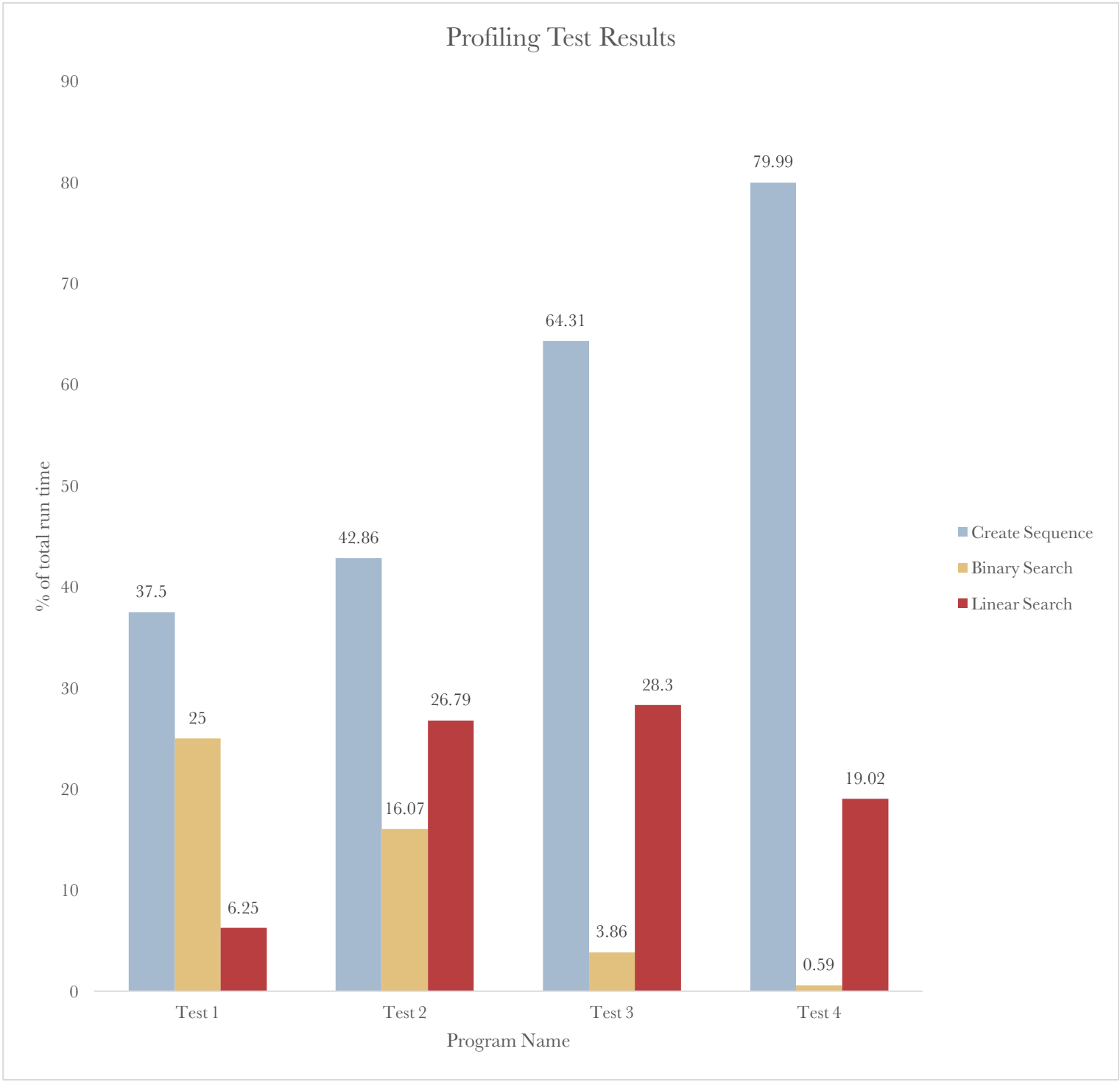
DISCUSSION OF RESULTS

We have observed the performances of linear and binary search algorithms for different sequence sizes, from the profiling results of 4 different test programs. Here is summary table showing the times taken to search respective elements.

Name of Test	Size of Sequence	Time taken by binary search (% of total run time)	Time taken by linear search (% of total run time)
Test 1	34	25.00	6.25
Test 2	334	16.07	26.79
Test 3	3334	3.86	28.30
Test 4	33334	0.59	19.02

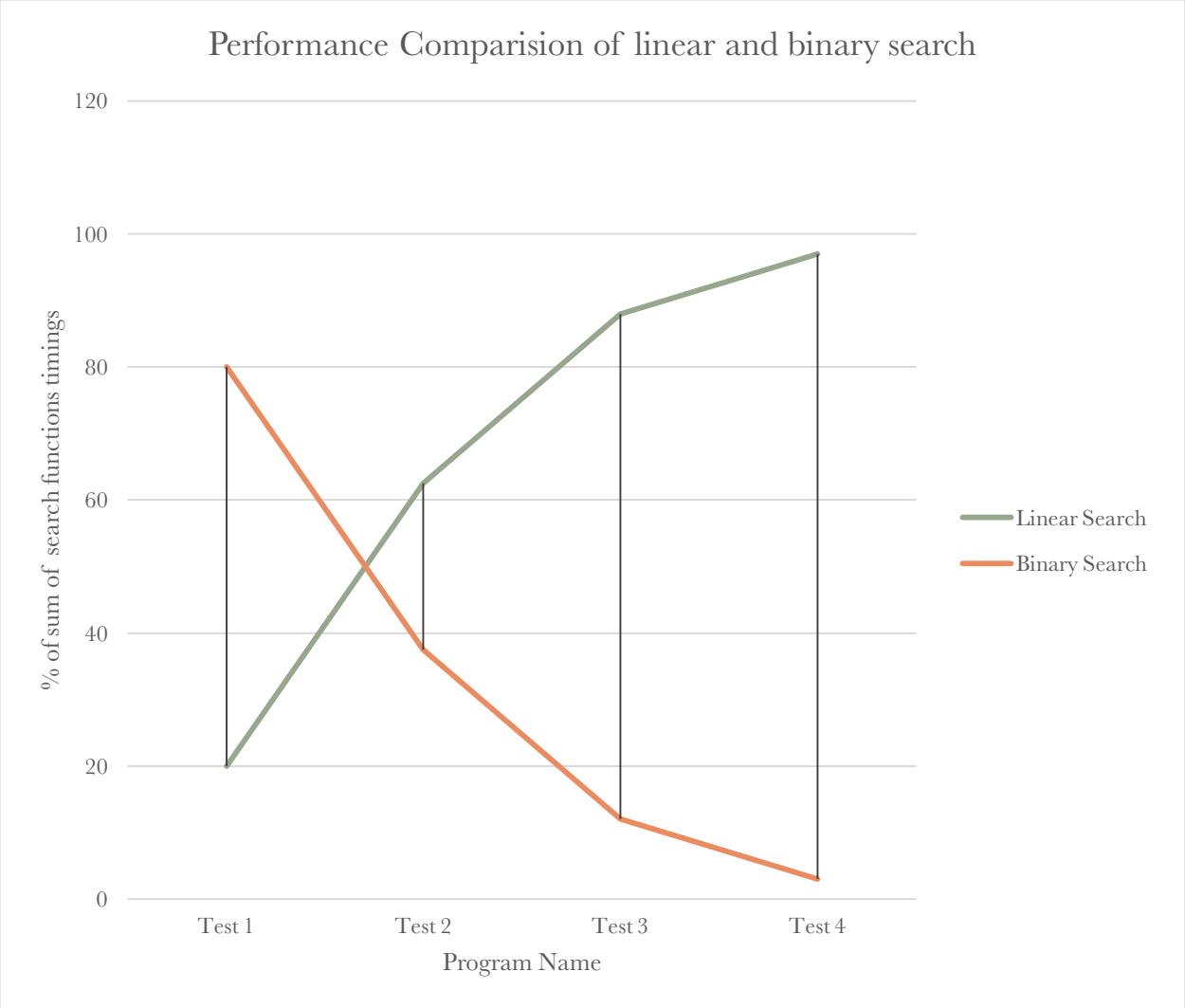
In test 1, the sequence has 34 elements and since the value being searched for '33' is in the top half of the sequence, linear search took way lesser time than binary search. As the size of the sequence increases like in Test 2, binary search algorithm works a lot more efficiently. Binary search took 16.07% of total time in Test 2 whereas linear search took 26.79% of total time. Moreover, in test 3, where the sequence contains 3334 elements we can notice huge differences in times taken. Linear search took 28.30% of total time, but binary search algorithm did the job in just 3.86% of total time. Lastly in Test 4, where the sequence contains 33334 elements linear search took 19.02% of the total time. On the other hand, binary search took just 0.59% of the total time!

We can note that timings are not completely accurate as they are scaled and most % of program run time is taken for creating a huge sequence. However, we can still notice the pattern of the complexities of both the algorithms. Linear search algorithm attributes O(n) linear complexity. 'n' (the number of elements in list) is directly proportional to the time/space required. Binary search algorithm attributes O(log n) logarithmic complexity. The time to search for a value increases in proportion to the logarithm of 'n' (the number of elements in list).



In order to plot a line graph of performances, I will be considering times of only binary and linear functions instead of the whole program. Which means that, I will be calculating cumulative time of linear and binary search functions, and assume it as the total time.

Name of Test	Sum of search functions timings (% of total run time)	Time taken by binary search (% of search function timings sum)	Time taken by linear search % of search function timings sum)
Test 1	6.25% + 25% = 31.25	(25.00 / 31.25) * 100 = 80	(6.25 / 31.25) * 100 = 20
Test 2	16.07% + 26.79% = 42.86	(16.07 / 42.86) * 100 = 37.49	(26.79 / 42.86) * 100 = 62.5
Test 3	3.86% + 28.3% = 32.16	(3.86 / 32.16) * 100 = 12	(28.30 / 32.16) * 100 = 87.99
Test 4	0.59% + 19.02% = 19.61	(0.59 / 19.61) * 100 = 3	(19.02 / 19.61) * 100 = 97



From the above graph, we can see that Linear search algorithm's efficiency is restricted to only small sequences like in Test 1. But for tests 2, 3 and 4 Linear search gets expensive. What I mean by expensive is that, it takes a lot of time and memory space to search up a value. However, binary search's line decreases heavily as the sequences becomes bigger. In fact, If the sequence had larger number of elements than test 4 the binary search function statistics won't even show up in the profiling diagram, because the % of total time gets so low that **gprof2dot** doesn't bother to display it. You can notice the gap between the two lines in the graph, it is drastically increasing. This point just highlights the efficiency of binary search algorithm.

In conclusion results prove that; Linear search algorithm is acceptable for small sequences but Binary search algorithm is clearly the faster and more efficient way for searching in massive sequences.