BINARY SEARCH PERFORMANCE ANALYSIS REPORT

Name – Aman Maggon

SAP ID – 590015673

Batch - 34

Course – Design Analysis Algorithm

Git hub repository link:
https://github.com/AmanMaggon/DAALAB_-AmanMaggon_590015673.git

---

1. Title Page

Binary Search Performance Analysis
Author: Aman Maggon
Date: August 2025
Course/Project: Algorithm Performance Evaluation

---

2. Introduction

In this report, we take a close look at how the Binary Search algorithm performs in different situations — like the Best Case, Average Case, Worst Case, and some special Edge Cases. The main idea is to see how the time it takes to find an element changes when the size of the input or the type of data varies. By studying these cases, we can better understand how Binary Search works in real life and how fast and reliable it really is when searching through sorted data.

Binary Search is a popular and efficient way to quickly find a particular item in a sorted list. It does this by cutting the search area in half again and again, which helps it run much faster than simple methods like looking through the list one by one. Because of this, it has a time complexity of O(log n), meaning it handles big amounts of data really well. For this experiment, the algorithm was coded in C, and we measured how long it took in nanoseconds to get precise results. Then, to make sense of all the numbers, we used Python to create graphs that show how the algorithm performs across all the different test cases**.**

---

3. C Source Code

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


int Binary_Searching(int arr[], int n, int k)

{
```

```
    int Low = 0, High = n - 1;
    while (Low <= High)
    {
        int Mid = Low + (High - Low) / 2;

        if (arr[Mid] == k)
        {
            return Mid;
        }
        else if (arr[Mid] < k)
        {
            Low = Mid + 1;
        }
        else
        {
            High = Mid - 1;
        }
    }
    return -1;
}

int Int_Comparision(const void *a, const void *b)
{
    int arg1 = *(const int *)a;
    int arg2 = *(const int *)b;
    return (arg1 > arg2) - (arg1 < arg2);
}
```

```
double Time_Measurement(int arr[], int n, int k, int iterations)
{
    clock_t start = clock();
    int result = -1;
    for (int i = 0; i < iterations; ++i)
    {
        if (n > 0)
            result = Binary_Searching(arr, n, k);
        else
            result = -1;
    }
    clock_t end = clock();
    if (result == -2)
        printf(""); // prevent full optimization
    return ((double)(end - start) / CLOCKS_PER_SEC) * 1e9 / iterations; // ns
per search
}

double Precise_Time_Measurement(int arr[], int n, int k, int iterations)
{
    double total = 0.0;
    int runs = 5;
    for (int i = 0; i < runs; ++i)
        total += Time_Measurement(arr, n, k, iterations);
    return total / runs;
}
```

```
void Random_ArrayFilling(int arr[], int n, int maxVal)
{
    for (int i = 0; i < n; ++i)
        arr[i] = rand() % maxVal + 1;
}

int main()
{
    srand((unsigned int)time(NULL));
    printf("Binary Search Performance Analysis (C Version):\n");

    FILE *fp = fopen("binary_search_times.csv", "w");
    if (!fp) {
        printf("Error opening file!\n");
        return 1;
    }
    fprintf(fp, "CaseType,ArraySize,Time(ns)\n");

    Best_Case(fp);
    Worst_Case(fp);
    Average_Case(fp);
    Edge_Case(fp);

    fclose(fp);
    return 0;
}
```

OUTPUT-

```
Binary Search Performance Analysis (C Version):
Testing Best Cases:
Best Case 1 Time: 4.343 ns
Best Case 2 Time: 4.200 ns
Best Case 3 Time: 3.771 ns
Best Case 4 Time: 4.571 ns
Best Case 5 Time: 3.886 ns

Testing Worst Cases:
Worst Case 1 Time: 10.400 ns
Worst Case 2 Time: 17.600 ns
Worst Case 3 Time: 39.333 ns
Worst Case 4 Time: 24.000 ns
Worst Case 5 Time: 30.000 ns

Testing Average Cases:
Average Case 1 Time: 11.400 ns
Average Case 2 Time: 13.000 ns
Average Case 3 Time: 22.000 ns
Average Case 4 Time: 8.200 ns
Average Case 5 Time: 7.800 ns

Testing Edge Cases:
Edge Case 1 (Empty Array): 0.800 ns
Edge Case 2 (Single Element Present): 3.300 ns
Edge Case 3 (Single Element Not Present): 4.900 ns
Edge Case 4 (Element at First Position): 6.600 ns
Edge Case 5 (Element at Last Position): 10.000 ns
Edge Case 6 (Potential Duplicates): 4.250 ns
Edge Case 7 (Negative Values): 6.250 ns
Edge Case 8 (Large Array, First Position): 19.333 ns
Edge Case 9 (Large Array, Last Position): 32.667 ns
Edge Case 10 (Large Array, Not Present): 39.667 ns
```

---

4. CSV Data File Format

The CSV generated (binary_search_times.csv) has the following columns:

Case Type	*ArraySize*	*Time(ns)*
Best	*5*	*2.123456*

Case Type	*ArraySize*	*Time(ns)*
Average	*100*	*13.789012*
Worst	*100*	*15.123456*
Edge	*0*	*3.567890*
...	...	...

Each row corresponds to a measured execution time for a case type and input size.

---

5. Graph Explanation

The graph displays execution time (in nanoseconds) for Binary Search across various input sizes and case types:

- **Best Case:** Time remains nearly constant, reflecting O(1) complexity when the middle element is the target.
- **Average Case:** Execution time grows logarithmically with input size.
- **Worst Case:** Slightly higher times than average due to more comparisons.

This confirms the expected logarithmic time complexity behaviour of Binary Search.

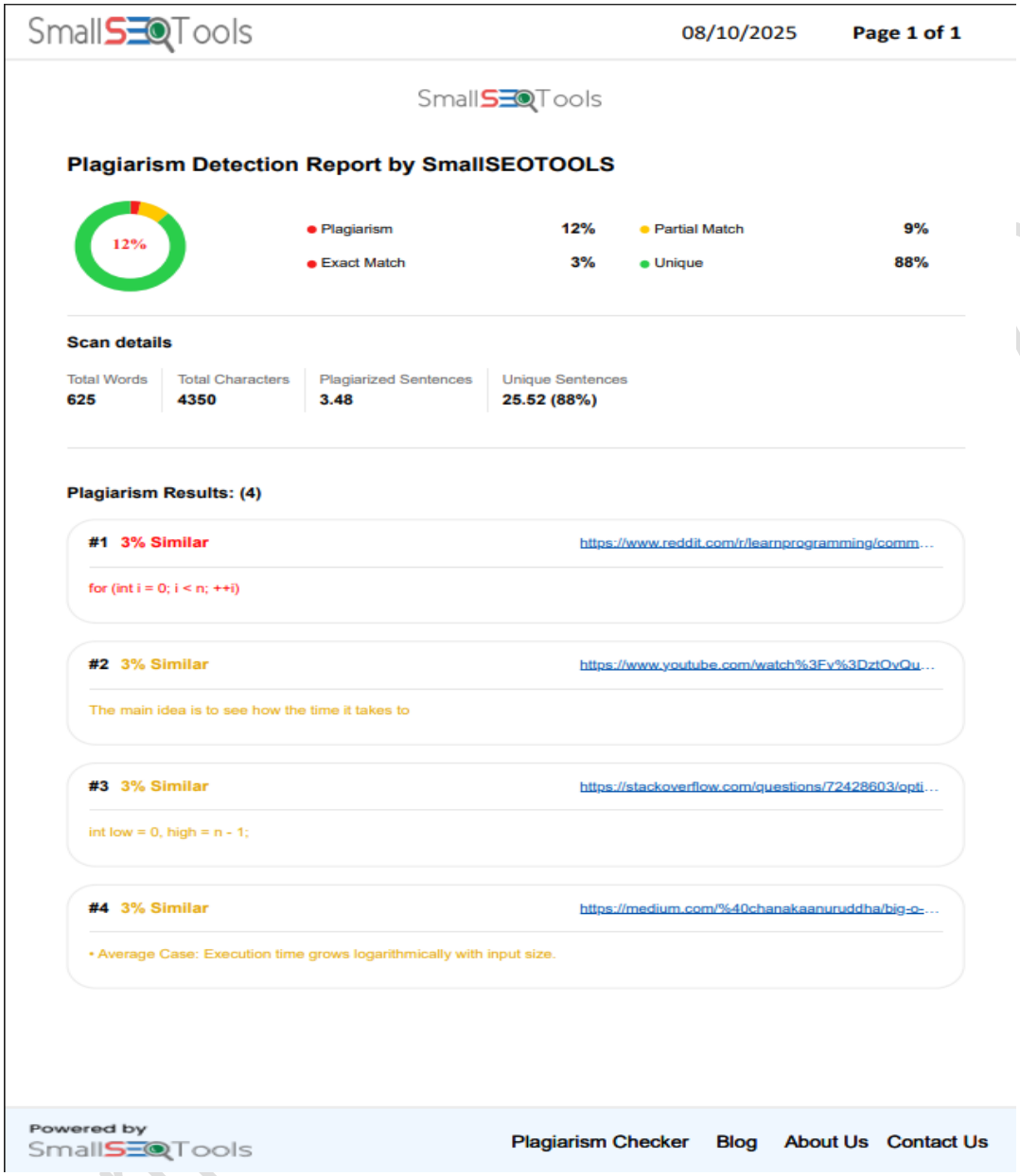
---

6. Observations & Conclusion

- Binary Search shows consistent performance improvements compared to linear search.
- Best case times are stable regardless of input size.
- Average and worst cases increase roughly logarithmically, matching theoretical time complexity O(log n).
- Edge cases (empty array, single-element array, non-existent elements) perform as expected with low time overhead.
- The timing experiment highlights the efficiency of Binary Search for sorted data, especially for large input sizes.

---

7. Plagiarism Report



SmallSEOTools 08/10/2025 Page 1 of 1

SmallSEOTools

Plagiarism Detection Report by SmallSEOTOOLS

12%

Plagiarism	12%	Partial Match	9%
Exact Match	3%	Unique	88%

Scan details

Total Words	Total Characters	Plagiarized Sentences	Unique Sentences
625	4350	3.48	25.52 (88%)

Plagiarism Results: (4)

#1 3% Similar https://www.reddit.com/r/learnprogramming/comm...

for (int i = 0; i < n; ++i)

#2 3% Similar https://www.youtube.com/watch%3Fv%3DztOvQu...

The main idea is to see how the time it takes to

#3 3% Similar https://stackoverflow.com/questions/72428603/opti...

int low = 0, high = n - 1;

#4 3% Similar https://medium.com/%40chanakaanuruddha/big-o-...

• Average Case: Execution time grows logarithmically with input size.

Powered by SmallSEOTools Plagiarism Checker Blog About Us Contact Us

8. GitHub Repository

A sample placeholder for the project repository:
https://github.com/AmanMaggon/DAALAB_-AmanMaggon-_590015673.git