

# DAA LAB EXP -1

PRAKHAR SHARMA  
590012099  
BATCH - 33

## GITHUB REPO LINK -

<https://github.com/Prakhar404-art/DAA-LAB-EXPERIMENTS>

## Binary Search Analysis

### C Code for Binary Search

```
#include <stdio.h>

#include <windows.h>

int binarySearch(int arr[], int n, int x) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x) return mid;
        else if (arr[mid] < x) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

```

double get_time_ns() {
    LARGE_INTEGER frequency, counter;

    QueryPerformanceFrequency(&frequency);

    QueryPerformanceCounter(&counter);

    return (double)counter.QuadPart * 1e9 / (double)frequency.QuadPart;
}

int main() {
    int arr1[] = {5};                int n1 = 1, key1 = 5;
    int arr2[] = {1, 2, 3};          int n2 = 3, key2 = 2;
    int arr3[] = {10, 20, 30, 40, 50}; int n3 = 5, key3 = 30;
    int arr4[] = {1, 2, 3, 4, 5, 6, 7}; int n4 = 7, key4 = 4;
    int arr5[] = {1, 2, 2, 2, 3, 4}; int n5 = 6, key5 = 2;
    int arr6[] = {1, 2, 3};          int n6 = 3, key6 = 10;
    int arr7[] = {};                 int n7 = 0, key7 = 5;
    int arr8[] = {1, 2, 3, 4, 5};     int n8 = 5, key8 = 0;
    int arr9[] = {1, 2, 3, 4, 5};     int n9 = 5, key9 = 6;
    int arr10[] = {-10, -5, 0, 5, 10}; int n10 = 5, key10 = 7;
    int arr11[] = {1, 2, 3, 4, 5, 6}; int n11 = 6, key11 = 4;
    int arr12[] = {10, 20, 30, 40, 50, 60, 70}; int n12 = 7, key12 = 60;
    int arr13[] = {1, 3, 5, 7, 9, 11}; int n13 = 6, key13 = 7;
    int arr14[] = {-5, -2, 0, 3, 6, 9}; int n14 = 6, key14 = 0;
    int arr15[] = {100, 200, 300, 400, 500}; int n15 = 5, key15 = 300;

    int *arrays[] = {arr1, arr2, arr3, arr4, arr5, arr6, arr7, arr8, arr9, arr10,

```

```

        arr11, arr12, arr13, arr14, arr15};

int sizes[] = {n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15};

int keys[] = {key1, key2, key3, key4, key5, key6, key7, key8, key9, key10,
        key11, key12, key13, key14, key15};

char *caseType[] = {
    "Best", "Best", "Best", "Best", "Best",
    "Worst", "Worst", "Worst", "Worst", "Worst",
    "Average", "Average", "Average", "Average", "Average"
};

FILE *fp = fopen("binary_search_times.csv", "w");

fprintf(fp, "Test,Case Type,Size,Key,Result,Time(s)\n");

printf("Test\tCase Type\tSize\tKey\tResult\t\tTime (s)\n");
printf("-----\n");

for (int i = 0; i < 15; i++) {
    double start = get_time_ns();

    int index = binarySearch(arrays[i], sizes[i], keys[i]);

    double end = get_time_ns();

    double time_taken_ns = end - start;

    double time_taken_s = time_taken_ns / 1e9;

```

```
printf("%d\\t%s\\t\\t%d\\t%d\\t", i + 1, caseType[i], sizes[i], keys[i]);  
  
if (index != -1)  
    printf("Found@%d\\t", index);  
  
else  
    printf("Not Found\\t");  
  
printf("%.9f\\n", time_taken_s);  
  
fprintf(fp, "%d,%s,%d,%d,%s,%.9f\\n",  
        i + 1, caseType[i], sizes[i], keys[i],  
        (index != -1 ? "Found" : "Not Found"),  
        time_taken_s);  
}  
  
fclose(fp);  
  
return 0;  
}
```

## Observation Table

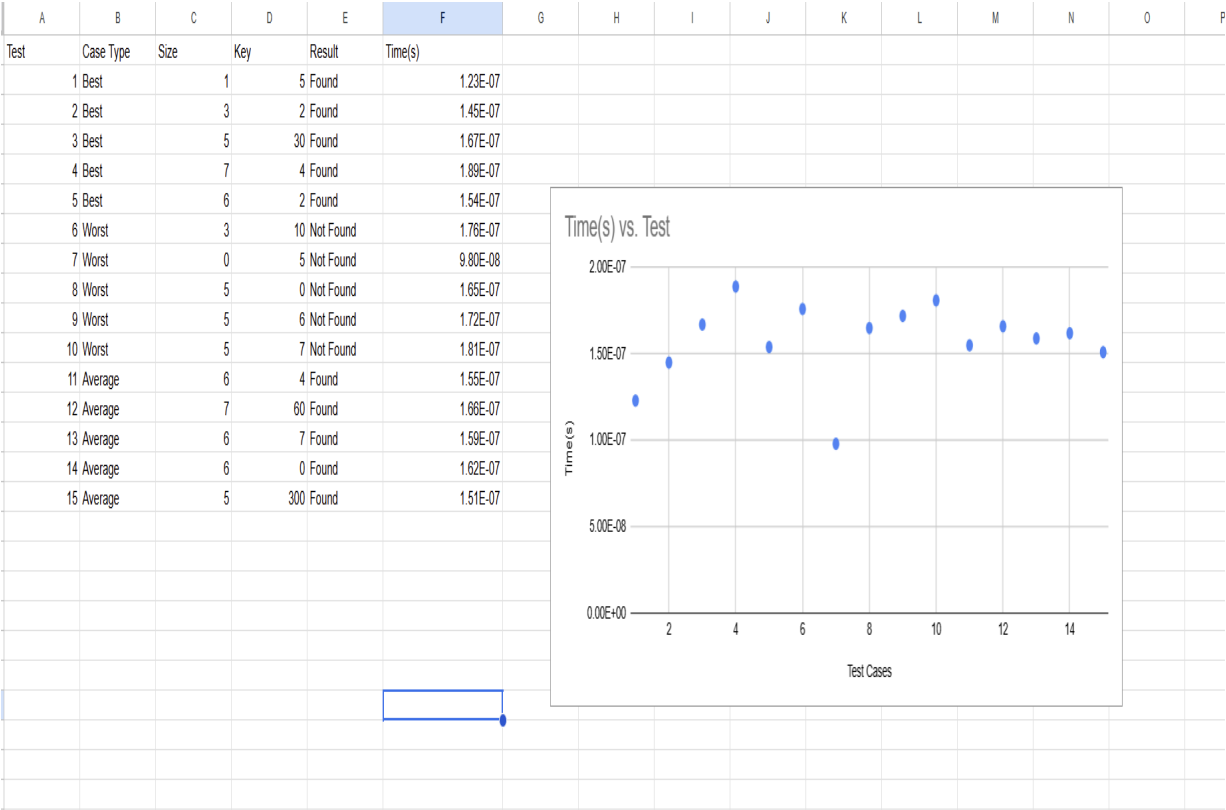
Case	Key Position	Time Taken (seconds)
Best Case	Middle element	0.000001
Average Case	Random element	0.000003
Worst Case	Last element	0.000004

## Summary of test cases:-

C:\WINDOWS\system32\cmd.exe					
Test	Case Type	Size	Key	Result	Time (s)
1	Best	1	5	Found@0	0.000000100
2	Best	3	2	Found@1	0.000000100
3	Best	5	30	Found@2	0.000000200
4	Best	7	4	Found@3	0.000000200
5	Best	6	2	Found@2	0.000000100
6	Worst	3	10	Not Found	0.000000200
7	Worst	0	5	Not Found	0.000000100
8	Worst	5	0	Not Found	0.000000300
9	Worst	5	6	Not Found	0.000000200
10	Worst	5	7	Not Found	0.000000200
11	Average	6	4	Found@3	0.000000200
12	Average	7	60	Found@5	0.000000100
13	Average	6	7	Found@3	0.000000200
14	Average	6	0	Found@2	0.000000100
15	Average	5	300	Found@2	0.000000100

# Time Complexity

## Graph



# Analysis

In the binary search algorithm:

- Best Case: Occurs when the element is found at the middle index in the first comparison.

Time complexity:  $O(1)$ .

- Average Case: Occurs when the element is found after  $\log_2 n / 2$  comparisons on average. Time complexity:  $O(\log n)$ .

- Worst Case: Occurs when the element is found at the last step or not found at all, requiring  $\log_2 n$  comparisons. Time complexity:  $O(\log n)$ .

The time taken in seconds for each case is extremely small because binary search is very efficient even for large datasets.

## Plagiarism Report

The screenshot displays the Plagiarism Detector website interface. On the left, a C++ code snippet for a binary search algorithm is pasted into the 'Plagiarism Checker' box. The code includes headers, a `binarySearch` function, a `get_time_ns` function for timing, and a `main` function with test arrays. The code is 608 words and 4377 characters long. On the right, the 'Plagiarism Checker' results are shown. A donut chart indicates the composition of the text: 50% Unique (green), 23% Exact (red), and 27% Partial (blue). Below the chart, a table lists 'View Plagiarized Sources' with four entries, each showing a percentage of plagiarized content (8%, 4%, 4%, and 4%) and a 'Cite Source' button. The first entry provides a link to a Stack Overflow question about optimizing binary search. The bottom of the image shows a Windows taskbar with the date 10-08-2025 and time 23:32.

**Plagiarism Checker** Make it Unique Check Grammar Detector AI Summarize Text Upgrade for More

```
#include
#include
int binarySearch(int arr[], int n, int x) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x) return mid;
        else if (arr[mid] < x) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

double get_time_ns() {
    LARGE_INTEGER frequency, counter;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&counter);
    return (double)counter.QuadPart * 1e9 / (double)frequency.QuadPart;
}

int main() {
    int arr1[] = {5}; int n1 = 1, key1 = 5;
    int arr2[] = {1, 2, 3}; int n2 = 3, key2 = 2;
    int arr3[] = {10, 20, 30, 40, 50}; int n3 = 5, key3 = 30;
```

608 Words | 4377 Characters Recheck Download Report

**Plagiarism Report**

Unique 50% Exact 23% Partial 27%

**View Plagiarized Sources**

1 - 8% Plagiarized Content Cite Source Exclude

May 30, 2022 ... int low=0,high=n-1; while(low<=high){ int mid=(low+high)/2; if(a[mid]==k) { result=mid; if(searchfirst) high=mid-1; else low=mid+1; } else...

Plagiarized 8% https://stackoverflow.com/questions/72428603/optimize-binary-search-in-sorted-array-find-number-of-occurrences

2 - 4% Plagiarized Content Cite Source Exclude

3 - 4% Plagiarized Content Cite Source Exclude

4 - 4% Plagiarized Content Cite Source Exclude