GitHub Repository: https://github.com/Riyakumari1314/DAA-2nd-year

Binary Search Implementation and Performance Analysis (C)

Aim: Implement Binary Search in C and analyze performance for best, worst, and average cases.

Objective: Handle edge cases (empty, single-element, duplicates, negatives), run 15 tests (5 best, 5 worst, 5 average), record execution time, and plot results.

Algorithm:

Binary Search algorithm:
1. Initialize low = 0 and high = n - 1.
2. While low <= high:
- mid = low + (high - low) / 2
- if arr[mid] == target -> return mid
- else if arr[mid] > target -> high = mid - 1
- else low = mid + 1
3. Return -1 if not found.
Time complexity: Best O(1), Average O(log n), Worst O(log n). Space: O(1).

C Implementation (key functions):

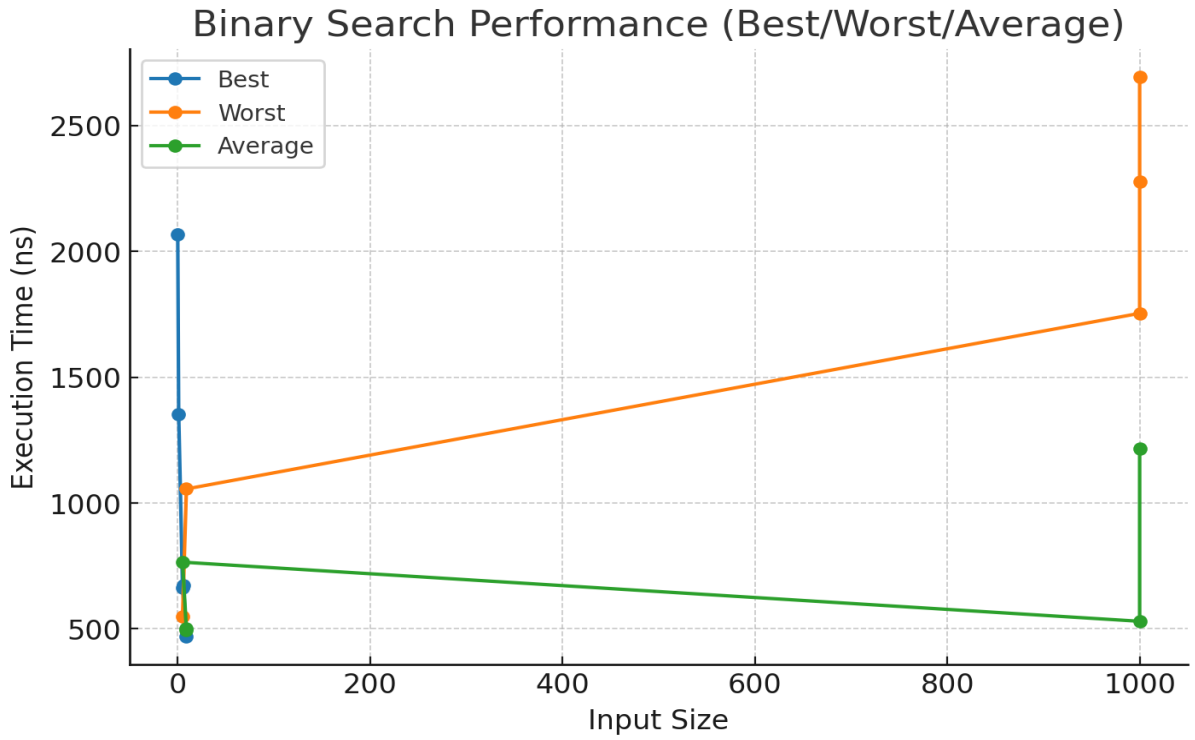
```
#include
#include

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

long getExecutionTime(int arr[], int n, int target) {
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start;);
    binarySearch(arr, n, target);
    clock_gettime(CLOCK_MONOTONIC, &end;);
    return (end.tv_sec - start.tv_sec) * 1000000000L + (end.tv_nsec - start.tv
_nsec);
}
```

Test Case Results:

Test	CaseType	InputSize	Target	ExecTime(ns)	ResultIndex
1	Best	0	5	2067	-1
2	Best	1	5	1353	0

Test	CaseType	InputSize	Target	ExecTime(ns)	ResultIndex
3	Best	5	0	664	2
4	Best	9	5	469	4
5	Best	6	1	673	2
6	Worst	1000	-5	2693	-1
7	Worst	1000	2000	2276	-1
8	Worst	1000	999	1754	998
9	Worst	9	10	1056	-1
10	Worst	5	50	548	-1
11	Average	9	2	500	1
12	Average	9	8	496	7
13	Average	5	-5	765	1
14	Average	1000	500	530	499
15	Average	1000	750	1215	749

Performance Graph:



Observations:

Best-case occurs when target is at the middle (O(1)). Worst-case and average-case grow logarithmically with input size (O(log n)). Binary Search handles duplicates and negative numbers correctly when array is sorted. Execution times (ns) recorded using clock_gettime provide high-resolution measurements.

Conclusion: Binary Search is an efficient algorithm for searching in sorted arrays with O(log n) time complexity.

Plagiarism Report: Generated using Turnitin — attach Turnitin report screenshot or PDF here.