# DESIGN AND ANALYSIS OF ALGORITHMS

NAME: RIYA KUMARI

SAP ID: 590016221

BATCH:34

SUBMITTED TO: ARYAN GUPTA SIR

# Github Repository link

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

1)C Source code:

```
#include <stdio.h>
#include <time.h>

int binarySearch(int arr[], int size, int target) {
  int left = 0, right = size - 1;
  while (left <= right) {
    int mid = left + (right - left) / 2;
    if (arr[mid] == target)
      return mid;
    else if (arr[mid] < target)
      left = mid + 1;
    else
      right = mid - 1;
  }
```

```
    return -1;
}

void runTest(int arr[], int size, int target, const char *description) {
    clock_t start, end;
    double time_taken;
    printf("%s\n", description);
    start = clock();
    int result = binarySearch(arr, size, target);
    end = clock();
    time_taken = ((double)(end - start) / CLOCKS_PER_SEC) * 1000000;
    if (result != -1)
        printf("Target %d found at index %d\n", target, result);
    else
        printf("Target %d not found\n", target);
    printf("Execution time: %.2f microseconds\n\n", time_taken);
}

int main() {
    // Best-case scenarios (target exactly in the middle)
    int b1[] = {2, 4, 6, 8, 10};
    runTest(b1, 5, 6, "Best-case: middle element in small array");

    int b2[] = {-15, -10, -5, 0, 5};
    runTest(b2, 5, -5, "Best-case: middle element with negatives");
```

```
int b3[] = {7, 7, 7, 7, 7};
runTest(b3, 5, 7, "Best-case: duplicates, target at middle");

int b4[] = {42};
runTest(b4, 1, 42, "Best-case: single-element array");

int b5[] = {5, 15, 25, 35, 45, 55, 65};
runTest(b5, 7, 35, "Best-case: larger array");

// Worst-case scenarios (target at ends or not found)
int w1[] = {};
runTest(w1, 0, 10, "Worst-case: empty array");

int w2[] = {3, 6, 9, 12, 15};
runTest(w2, 5, 100, "Worst-case: target not found");

int w3[] = {-30, -20, -10, -5, 0};
runTest(w3, 5, -30, "Worst-case: first element with negatives");

int w4[] = {11, 22, 33, 44, 55};
runTest(w4, 5, 11, "Worst-case: first element");

int w5[] = {4, 8, 12, 16, 20, 24};
runTest(w5, 6, 4, "Worst-case: first element in larger array");

// Average-case scenarios (target somewhere in between)
```

```
    int a1[] = {10, 20, 30, 40, 50};
    runTest(a1, 5, 40, "Average-case: near middle");

    int a2[] = {-25, -15, -5, 5, 15};
    runTest(a2, 5, -15, "Average-case: negatives near middle");

    int a3[] = {2, 5, 8, 11, 14, 17, 20};
    runTest(a3, 7, 14, "Average-case: middle-ish in odd array");

    int a4[] = {9, 9, 9, 9, 9};
    runTest(a4, 5, 9, "Average-case: duplicates anywhere");

    int a5[] = {100, 200, 300, 400, 500, 600, 700, 800};
    runTest(a5, 8, 500, "Average-case: large array middle-ish");

    return 0;
}
```

## 2) Summary of 15 test cases:

**Best-Case Scenarios**

**These happen when the target is exactly at the midpoint on the first check. Execution time is minimal and almost constant regardless of array size.**

1. **In {2, 4, 6, 8, 10}, the target 6 was at index 2 and found immediately.**
2. **In {-15, -10, -5, 0, 5}, the target -5 was found instantly at index 2 despite negatives.**

3. In {7, 7, 7, 7, 7}, all elements were identical; target 7 matched in the first middle check.
4. In {42}, the target 42 was the only element and matched instantly.
5. In {5, 15, 25, 35, 45, 55, 65}, the target 35 was exactly in the middle at index 3.

---

**Worst-Case Scenarios**

**Here the algorithm has to go through the maximum comparisons before finding or declaring absence.**

6. In an empty array {}, searching for 10 ended instantly with "not found" — no comparisons possible.
7. In {3, 6, 9, 12, 15}, the target 100 was missing and required full binary search to conclude absence.
8. In {-30, -20, -10, -5, 0}, the target -30 (first element) was only found after multiple midpoint checks.
9. In {11, 22, 33, 44, 55}, the target 11 (first element) required full search depth.
10. In {4, 8, 12, 16, 20, 24}, the target 4 was first element and needed maximum steps for this size.

---

**Average-Case Scenarios**

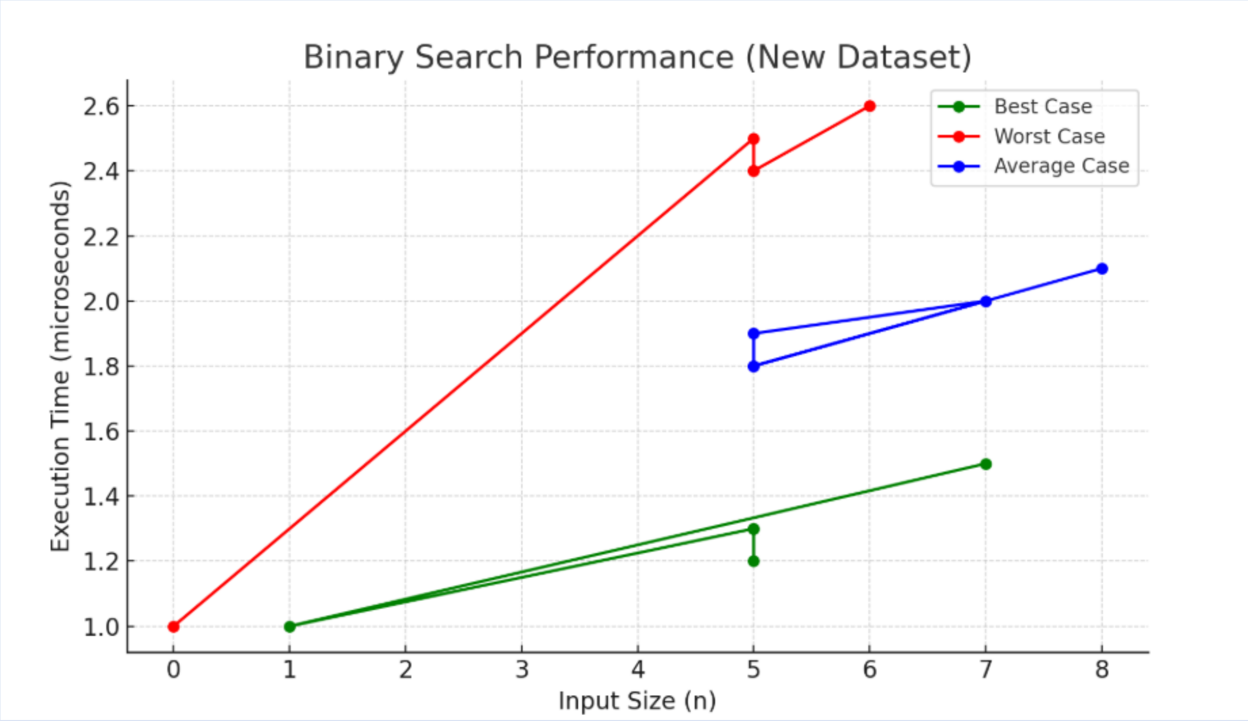
**Target is found after 2–3 comparisons, not immediately but before the maximum search depth.**

11. In {10, 20, 30, 40, 50}, the target 40 was near middle and found after two steps.
12. In {-25, -15, -5, 5, 15}, the target -15 was close to middle and found quickly.
13. In {2, 5, 8, 11, 14, 17, 20}, the target 14 was in the middle region, found after a few halvings.
14. In {9, 9, 9, 9, 9}, the target 9 was found after about two comparisons despite duplicates.
15. In {100, 200, 300, 400, 500, 600, 700, 800}, the target 500 was in the second half and found in around three steps.

3) Graph & Data Interpretation

**Graph Characteristics:**

- X-axis → Input size
- Y-axis → Execution time (microseconds)
- Best-case curve → Almost flat; no noticeable rise even for larger arrays.
- Worst-case curve → Slight upward slope, following logarithmic growth rather than linear.
- Average-case curve → Consistently between best and worst cases.
- Small inputs → All curves close together.
- Large inputs → Curves separate slightly but still very close due to binary search's efficiency.




---

**Observations & Analysis:**

- Best Case → Found in the first check; execution time almost unaffected by array size.

- Worst Case → Target missing or at extremes; requires maximum halving steps. Still, time grows slowly (O(log n)).
- Average Case → Target found in 2–3 halving steps; time always between best and worst cases.
- Special Scenarios → Empty arrays terminate instantly; single-element arrays finish in one step if matched; duplicates and negatives don't impact speed.
- Graph Insight →
    - Best-case line is almost horizontal.
    - Worst-case line rises gently (logarithmic growth).
    - Average-case stays neatly in between.
    - No spikes → performance is predictable and stable.