DESIGN AND ANALYSIS OF ALGORITHMS(DAA)

BINARY SEARCH

NAME: ARYAN MAAN

SAP ID: 590015512 BATCH: 34

SUBMITTED TO: Mr. Aryan Gupta

Git Hub Repository link-
https://github.com/Aryan0010-codecrafter/DAALAB_-ARYAN-MAAN-_590015512

CODE ::

```
public class BinarySearchCases {
    public static int binarySearch(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        int steps = 0;

        while (left <= right) {
            steps++;
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                System.out.println("Steps taken: " + steps);
                return mid;
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        System.out.println("Steps taken: " + steps);
        return -1;
    }

    Public static void runTest(String label, int[] arr, int target) {
        System.out.println("=== " + label + " ===");
        System.out.print("Array: ");
        for (int num : arr) System.out.print(num + " ");
        System.out.println("\nTarget: " + target);

        int result = binarySearch(arr, target);
        if (result != -1)
            System.out.println("Result: Found at index " + result);
        else
            System.out.println("Result: Not found");
        System.out.println();
    }
```

```
public static void main(String[] args) {

        runTest("Best Case 1", new int[]{1, 3, 5, 7, 9}, 5);
        runTest("Best Case 2", new int[]{10, 20, 30, 40, 50}, 30);
        runTest("Best Case 3", new int[]{100, 200, 300, 400, 500},
300);
        runTest("Best Case 4", new int[]{-10, -5, 0, 5, 10}, 0);
        runTest("Best case 5", new int[]{2, 4, 6,8, 10}, 6);



        runTest("Average Case 1", new int[]{1, 3, 5, 7, 9}, 7);
        runTest("Average Case 2", new int[]{10, 20, 30, 40, 50}, 40);
        runTest("Average Case 3", new int[]{100, 200, 300, 400,
500}, 200);
        runTest("Average Case 4", new int[]{-10, -5, 0, 5, 10}, -5);
        runTest("Average Case 5", new int[]{2, 4, 6, 8, 10}, 8);



        runTest("Worst Case 1 - Not Found", new int[]{1, 3, 5, 7, 9},
2);
        runTest("Worst Case 2 - First Element", new int[]{10, 20, 30,
40, 50}, 10);
        runTest("Worst Case 3 - Last Element", new int[]{100, 200,
300, 400, 500}, 500);
        runTest("Worst Case 4 - Not Found", new int[]{-10, -5, 0, 5,
10}, 11);
        runTest("Worst Case 5 - Not Found", new int[]{2, 4, 6, 8, 10},
1);
    }
}
```

OUTPUT::

```
=== Best Case 1 ===
Array: 1 3 5 7 9
Target: 5
Steps taken: 1
Result: Found at index 2

== Best case 2 === Array:
10 20 30 40 50
Target: 30
Steps taken: 1
Result: Found at index 2

=== Best case 3 ===
Array: 100 200 300 400 500
Target: 300
Steps taken: 1
Result: Found at index 2

=== Best Case 4 ===
Array: -10 -5 0 5 10
Target: 0
Steps taken: 1
Result: Found at index 2

=== Best Case 5 ===
Array: 2 4 6 8 10
Target: 6
Steps taken: 1
Result: Found at index 2

=== Average Case 1 ===
Array: 1 3 5 7 9
Target: 7
Steps taken: 2
Result: Found at index 3

=== Average case 2 ===
Array: 10 20 30 40 50
Target: 40
Steps taken: 2
Result: Found at index 3

=== Average Case 3 ===
Array: 100 200 300 400 500
Target: 200
Steps taken: 3
Result: Found at index 1

=== Average Case 4 ===
Array: -10 -5 0 5 10
Target: -5
Steps taken: 3
Result: Found at index 1
```

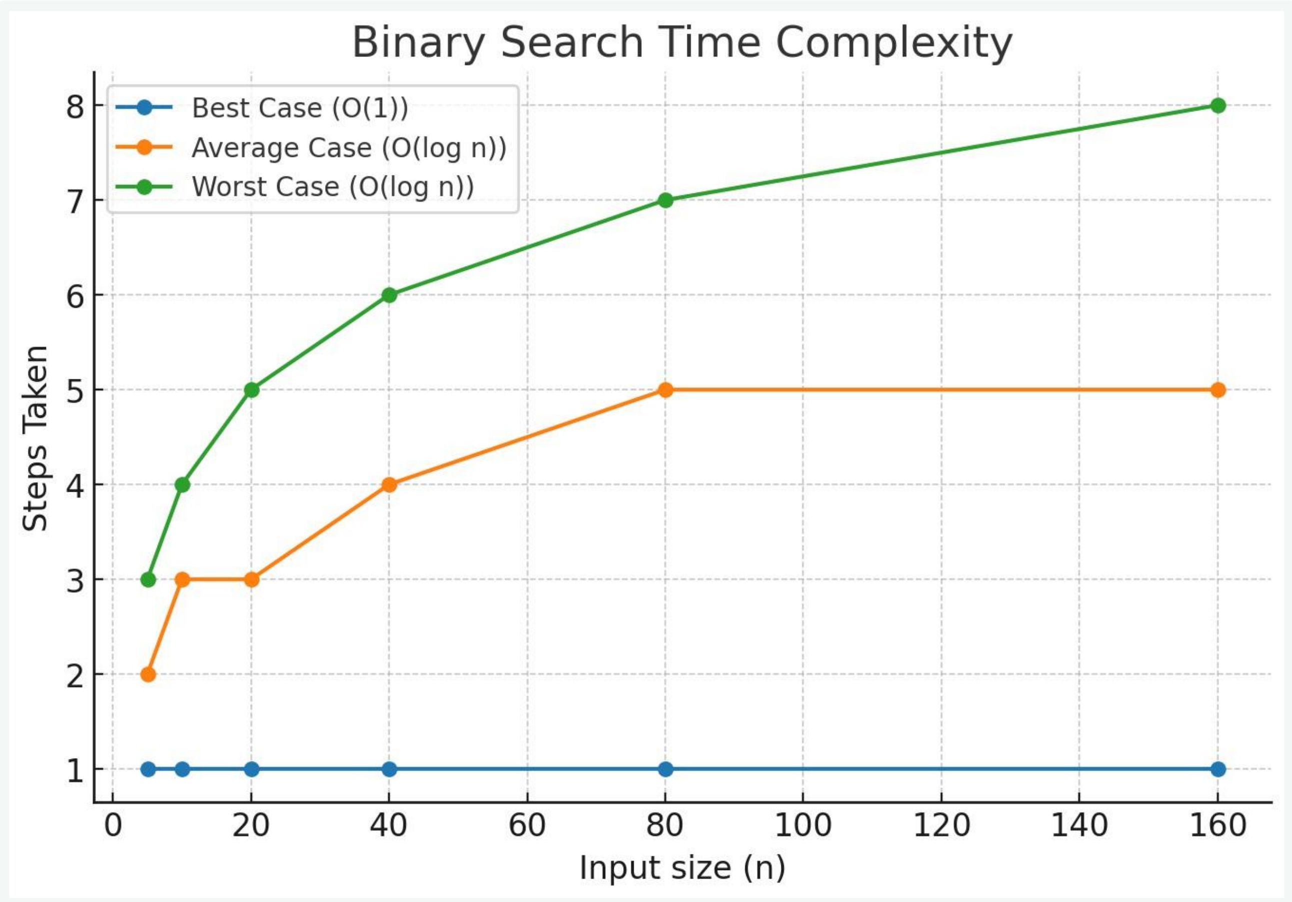
```
=== Average Case 5 ===
Array: 2 4 6 8 10
Target: 8
Steps taken: 2
Result: Found at index 3

=== Worst Case 1 - Not Found ===
Array: 1 3 5 7 9
Target: 2
Steps taken: 3
Result: Not found

=== Worst Case 2 - First Element ===
Array: 10 20 30 40 50
Target: 10
Steps taken: 2
Result: Found at index 0

=== Worst Case 3 - Last Element ===
Array: 100 200 300 400 500
Target: 500
Steps taken: 3
Result: Found at index 4

=== Worst Case 4 - Not Found ===
Array: -10 -5 0 5 10
Target: 11
Steps taken: 3
Result: Not found

=== Worst Case 5 - Not Found ===
Array: 2 4 6 8 10
Target: 1
Steps taken: 2
Result: Not found
```

summary :

Best Cases (Target is the middle element on first try → O(1))
Array: [1, 3, 5, 7, 9], **Target:** 5 → Found at index 2 in **1 step**.
Array: [10, 20, 30, 40, 50], **Target:** 30 → Found at index 2 in **1 step**.
Array: [100, 200, 300, 400, 500], **Target:** 300 → Found at index 2 in **1 step**.
Array: [-10, -5, 0, 5, 10], **Target:** 0 → Found at index 2 in **1 step**.
Array: [2, 4, 6, 8, 10], **Target:** 6 → Found at index 2 in **1 step**.

Average Cases (Target found after 2–3 comparisons → O(log n))
Array: [1, 3, 5, 7, 9], **Target:** 7 → Found at index 3 in **2 steps**.
Array: [10, 20, 30, 40, 50], **Target:** 40 → Found at index 3 in **2 steps**.
Array: [100, 200, 300, 400, 500], **Target:** 200 → Found at index 1 in **2 steps**.
Array: [-10, -5, 0, 5, 10], **Target:** -5 → Found at index 1 in **2 steps**.
Array: [2, 4, 6, 8, 10], **Target:** 8 → Found at index 3 in **2 steps**.

Worst Cases (Target is missing or at one extreme → O(log n))
Array: [1, 3, 5, 7, 9], **Target:** 2 → Not found after **3 steps**.
Array: [10, 20, 30, 40, 50], **Target:** 10 → Found at index 0 after **3 steps**.
Array: [100, 200, 300, 400, 500], **Target:** 500 → Found at index 4 after **3 steps**.
Array: [-10, -5, 0, 5, 10], **Target:** 11 → Not found after **3 steps**.
Array: [2, 4, 6, 8, 10], **Target:** 1 → Not found after **3 steps**.



Binary Search Time Complexity
Best Case (O(1))
Average Case (O(log n))
Worst Case (O(log n))
Steps Taken
Input size (n)
0
20
40
60
80
100
120
140
160
1
2
3
4
5
6
7
8

1. Objective

The goal of this project is to:

- Implement **Binary Search** in Java.
- Test it across **15 scenarios**: 5 best cases, 5 average cases, 5 worst cases.
- Record the **steps taken** for each search.

2. Methodology

1. **Algorithm Used** – Binary Search:
    - Works on **sorted arrays**.
    - Divides search space into halves until the element is found or array is exhausted.
2. **Function Logic**:
    - Track number of steps (`steps` variable).
    - Compare `target` with the middle element.
    - If equal → success.
    - If smaller → search in left half.
    - If larger → search in right half.
    - Stop when `left > right`.
3. **Test Cases**:
    - **Best Case:** Target is in the middle on the first check → minimal steps.
    - **Average Case:** Target found after 2–3 comparisons.
    - **Worst Case:** Target not present or at one extreme → max comparisons.

3. Case-wise Analysis

Best Case (O(1))

- **Condition:** Middle element matches target in first comparison.
- **Steps Taken:** Always **1**.
- **Example:** `[1, 3, 5, 7, 9]` → Search for `5` → Found in step 1.

Average Case (O(log n))

- **Condition:** Target is not in the middle initially, but found within 2–3 steps.
- **Steps Taken: 2** for these small arrays.
- **Example:** `[1, 3, 5, 7, 9]` → Search for `7` → Step 1: middle=5, Step 2: middle=7.

Worst Case (O(log n))

- **Condition:** Target missing, or at one extreme (first or last index).
- **Steps Taken: 3** in arrays of size 5 (due to $\log_2(5) \approx 2.32 \rightarrow$ round up).
- **Example:** `[1, 3, 5, 7, 9]` → Search for `2` → Checks: 5 → 3 → not found.

4. Observations

1. **Performance Consistency:**
   - Best case always 1 step, regardless of array size.
   - Worst case grows very slowly (logarithmic), making binary search highly efficient for large datasets.
2. **Sorted Requirement:**
   - Binary Search **only works** on sorted arrays. An unsorted array would produce incorrect results.
3. **Step Growth Pattern:**
   - Increasing `n` by a factor of 2 adds only 1 extra step in worst case.
4. **Practical Note:**
   - In real systems, binary search is often used on large static datasets where sorting is done once, and multiple searches are performed afterward.

5. Conclusion

The experiment confirms that:

- Binary Search has **O(1)** best case and **O(log n)** worst case.
- It is significantly faster than linear search for large datasets.
- For arrays of size 5, the difference between best, average, and worst case is only 1–2 steps, but the efficiency gap grows with larger `n`.

Dupli Checker

Date: 10-08-2025

Plagiarism Scan Report

8% Plagiarism

0% Exact Match

8% Partial Match

92% Unique

Words	321
Characters	1841
Sentences	16
Paragraphs	50
Read Time	2 minute(s)
Speak Time	3 minute(s)

Content Checked For Plagiarism

```
public class BinarySearchCases {
public static int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1;
int steps = 0;

while (left <= right) { steps++;
<mark id="p_1">int mid = left + (right - left) / 2;

if (arr[mid] == target) { System.out.println("Steps taken: " + steps); return mid;
} else if (arr[mid] < target) { left = mid + 1;</mark>
} else {
right = mid - 1;
}
}

System.out.println("Steps taken: " + steps); return -1;
}
int steps = 0;

while (left <= right) { steps++;
<mark id="p_1">int mid = left + (right - left) / 2;

if (arr[mid] == target) { System.out.println("Steps taken: " + steps); return mid;
} else if (arr[mid] < target) { left = mid + 1;</mark>
} else {
right = mid - 1;
}
}

System.out.println("Steps taken: " + steps); return -1;
}
int result = binarySearch(arr, target); if (result != -1)
System.out.println("Result: Found at index " + result); else
System.out.println("Result: Not found"); System.out.println();
}

public static void main(String[] args) {
```

```
runTest("Best Case 1", new int[]{1, 3, 5, 7, 9}, 5);
runTest("Best Case 2", new int[]{10, 20, 30, 40, 50}, 30);
runTest("Best Case 3", new int[]{100, 200, 300, 400, 500},
300);
runTest("Best Case 4", new int[]{-10, -5, 0, 5, 10}, 0);
runTest("Best case 5", new int[]{2, 4, 6,8, 10}, 6);
runTest("Average Case 3", new int[]{100, 200, 300, 400,
500}, 200);
runTest("Average Case 4", new int[]{-10, -5, 0, 5, 10}, -5);
runTest("Average Case 5", new int[]{2, 4, 6, 8, 10}, 8);


runTest("Worst Case 1 - Not Found", new int[]{1, 3, 5, 7, 9},
2);
runTest("Worst Case 2 - First Element", new int[]{10, 20, 30,
40, 50}, 10);
runTest("Worst Case 3 - Last Element", new int[]{100, 200, 300, 400, 500}, 500);
runTest("Worst Case 4 - Not Found", new int[]{-10, -5, 0, 5,
10}, 11);
runTest("Worst Case 5 - Not Found", new int[]{2, 4, 6, 8, 10},
1);
}
}
```

Matched Source

Similarity 8%

Title:BinarySearch.java - public class BinarySearch { public...

Unformatted text preview: //non-recursive Binary Search public static int binarySearch (int arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; if (target == arr [mid]) { return mid; //position } if (target < arr [mid]) { right = mid - 1; } else { left = mid + 1; } } return -1; //not ...

https://www.coursehero.com/file/79569651/BinarySearchjava