TG ACADEMIC RESOURCE SERIES

BINARY SEARCH

A Conceptual Introduction with a Worked Dry Run

SUBJECT	Data Structures & Algorithms
LEVEL	BTech / BCA / MCA / BSc / MSc / MTech
RESOURCE TYPE	Sample learning note
PUBLICATION ID	DSA-001 | Version 1.0

Prepared by Tanmoy Gupta

Computer Science Educator & Industry Professional

This sample introduces the core idea, demonstrates one complete execution trace, and provides a short practice set. It is designed as a focused preview of a structured learning approach.

1. Learning Objectives

After studying this note, you should be able to:

• Explain the principle behind Binary Search.
• Identify when Binary Search can be applied.
• Trace the values of **low**, **high** and **mid** during execution.
• State its time and auxiliary-space complexity.

2. The Core Idea

Binary Search is a searching technique that repeatedly reduces the search interval by approximately half. Instead of checking every element, it examines the middle element and decides which half may contain the target.

> The essential precondition is that the data must be **sorted** according to the order used by the search. Without sorted data, discarding half of the elements is not logically justified.

3. Algorithm

For an ascending array A and target value x:

1. Set **low = 0** and **high = n** – **1**.
2. While low ≤ high, calculate **mid = low + (high** – **low) // 2**.
3. If A[mid] = x, report the position.
4. If A[mid] < x, continue in the right half: low = mid + 1.
5. Otherwise, continue in the left half: high = mid – 1.
6. If the interval becomes empty, the target is not present.

Why use low + (high – low) // 2?

This midpoint expression avoids the potential integer overflow that can occur in some languages when calculating (low + high) directly for very large index values.

Pseudocode

```
low ← 0
high ← n – 1

WHILE low ≤ high
mid ← low + (high – low) DIV 2
IF A[mid] = x: RETURN mid
ELSE IF A[mid] < x: low ← mid + 1
ELSE: high ← mid – 1

RETURN NOT_FOUND
```

4. Worked Example: Complete Dry Run

Given the sorted array below, search for the target **23**.

Index	0	1	2	3	4	5	6	7	8
A[index]	3	7	11	15	19	23	27	31	35

Pass	low	high	mid	A[mid]	Decision
1	0	8	4	19	19 < 23 → low = 5
2	5	8	6	27	27 > 23 → high = 5
3	5	5	5	23	23 = 23 → FOUND

Result: The target 23 is found at index 5 (the sixth position in one-based counting). Only three middle-element comparisons were required.

Dry-run interpretation

• Pass 1: The middle value is 19. Since 19 is smaller than 23, indices 0 through 4 can be discarded.
• Pass 2: The middle value of the remaining interval is 27. Since 27 is larger than 23, indices 6 through 8 can be discarded.
• Pass 3: The remaining middle value is 23, so the search terminates successfully.

Notice how the search interval shrinks from 9 elements to 4, then 1. This repeated reduction is the reason Binary Search is efficient.

5. Complexity Analysis

At each iteration, Binary Search discards roughly half of the remaining search interval. After k iterations, the remaining interval is approximately n / 2■.

The search ends when the interval contains at most one candidate. Therefore, the number of iterations grows logarithmically with n.

Case	Time Complexity	Explanation
Best	O(1)	The target is at the first midpoint.
Average	O(log n)	The interval is repeatedly halved.
Worst	O(log n)	The target is last to be found or absent.
Iterative auxiliary space	O(1)	Only a fixed number of variables are used.

6. Common Errors to Avoid

1. Applying Binary Search to unsorted data.
2. Using **low < high** instead of **low** ≤ **high** in this particular inclusive-boundary algorithm.
3. Updating low to mid instead of mid + 1, which can cause an infinite loop.
4. Updating high to mid instead of mid – 1, which can also prevent progress.
5. Confusing an array index (zero-based) with a position (one-based).

7. Where Binary Search Is Useful

Binary Search is useful when data is ordered and random access to the middle element is efficient, as with arrays. Related ideas also appear in searching a range of answers, finding insertion positions, and solving monotonic decision problems.

Important limitation

Although Binary Search uses few comparisons, it is not automatically the best choice for every data structure. For example, repeatedly accessing the middle node of a linked list is not constant-time, so the usual array-based efficiency advantage does not transfer directly.

8. Check Your Understanding

Try these questions before checking the suggested answers. They test the central ideas from this sample; they are not a complete practice set.

Practice Questions

Q1. Can Binary Search be applied directly to the array [4, 9, 2, 12, 7]? Explain why or why not.

Q2. For the sorted array [2, 5, 8, 12, 16, 23, 38], trace Binary Search for target 16. Record low, high, mid and A[mid] at each pass.

Q3. State the worst-case time complexity of Binary Search and explain why it is logarithmic.

Suggested Answers (brief)

A1. Not directly. The array is not sorted. Sort it first if the task permits that preprocessing step; include sorting cost when analysing the complete solution.

A2. Pass 1: low=0, high=6, mid=3, A[mid]=12. Since 12 < 16, set low=4. Pass 2: low=4, high=6, mid=5, A[mid]=23. Since 23 > 16, set high=4. Pass 3: low=4, high=4, mid=4, A[mid]=16. Found at index 4.

A3. O(log n), because each comparison reduces the remaining search interval to approximately half its previous size.

Continue Learning

This note is a focused introduction. A complete study of searching algorithms should also examine implementation in a programming language, recursive Binary Search, duplicate values, first/last occurrence, insertion position, boundary conditions and a wider range of problem-solving exercises.

In guided classes: these extensions can be explored through live dry runs, code walkthroughs, debugging, progressively harder problems and individual feedback.

About this resource
TG Academic Resource Series | DSA-001 | Version 1.0
Prepared by Tanmoy Gupta — Computer Science Educator & Industry Professional

Original educational material prepared for student learning and revision. Redistribution or commercial reproduction without permission is not authorised.