Parallel & Distributed Computing

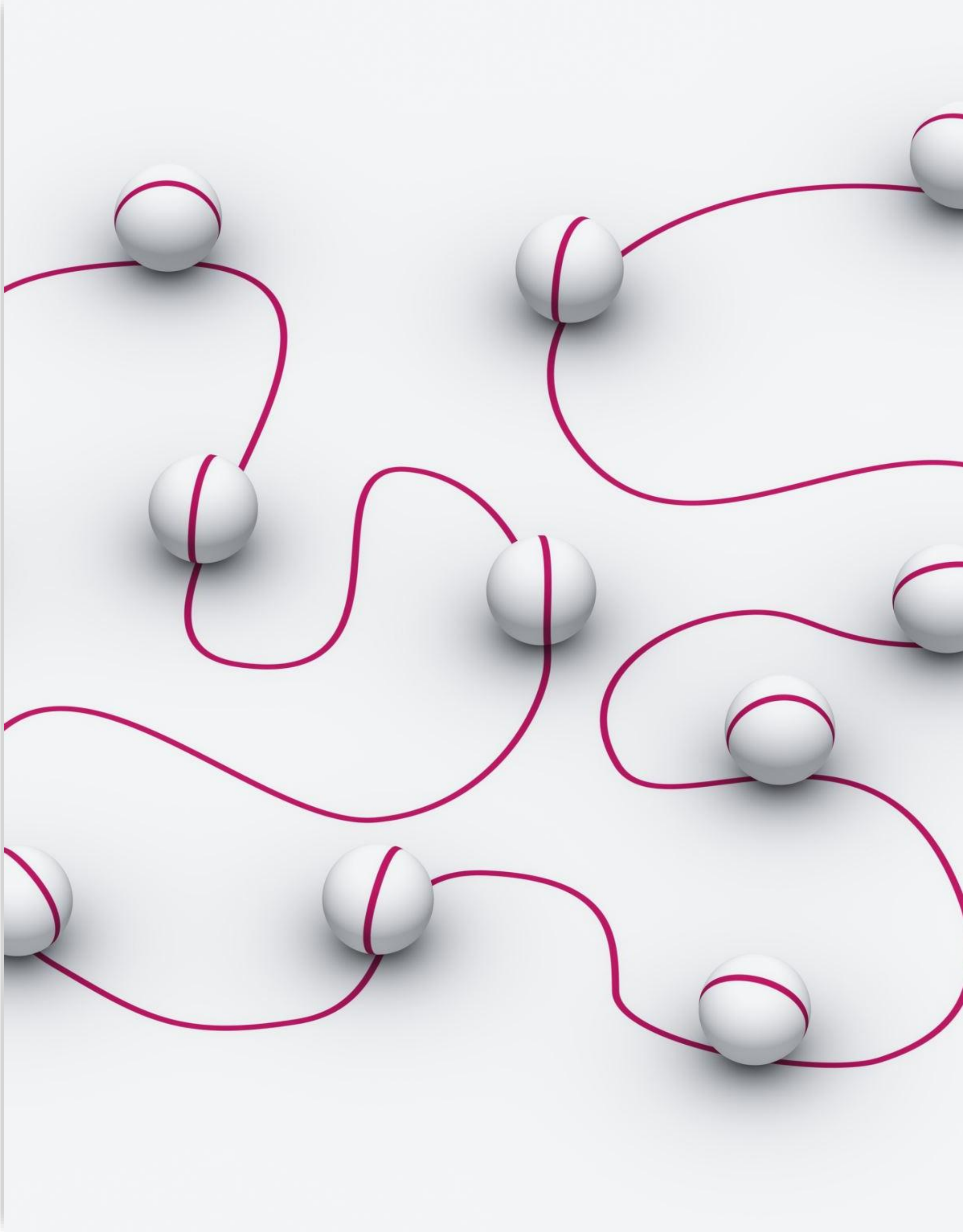
Lecture # 9

By,

Dr. Ali Akbar Siddique

Introduction to Divide and Conquer

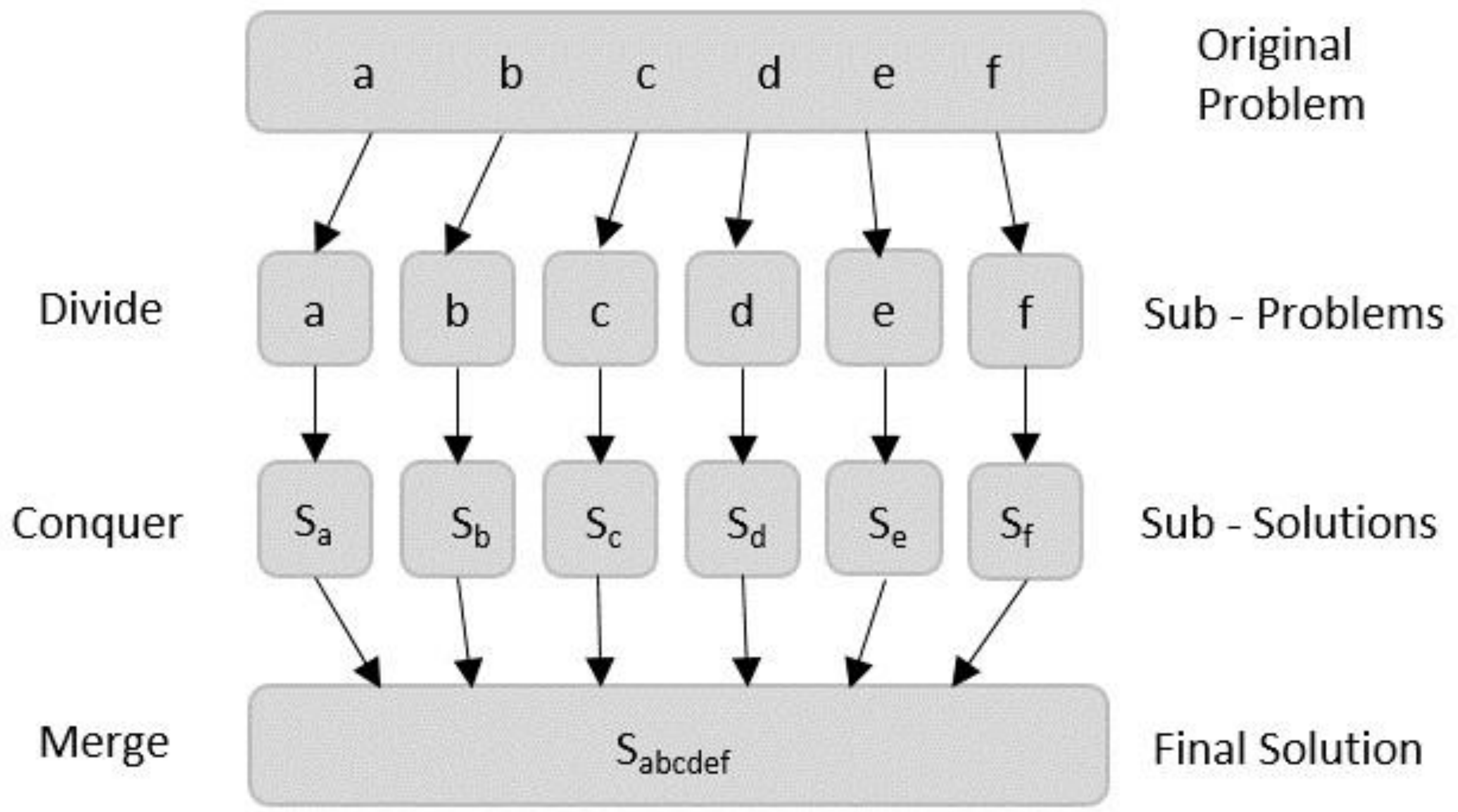
- The Divide and Conquer approach is a fundamental algorithmic strategy used to solve complex problems by breaking them down into simpler subproblems, solving each subproblem independently, and then combining their results to form a solution to the original problem.



Core Concept

The method works in **three main steps**:

1. **Divide**
   1. Break the original problem into **smaller**, **manageable subproblems**.
   2. Ideally, the subproblems should be of the same type as the original problem.
   3. The division continues **recursively** until the subproblems become **simple enough** to solve directly.
2. **Conquer**
   1. Solve each subproblem **recursively**.
   2. If the subproblem size is small enough (base case), solve it **directly** (non-recursively).
3. **Combine**
   1. Integrate the solutions of the subproblems into a **final solution**.
   2. This step depends heavily on the nature of the original problem.


a b c d e f
Original Problem
Divide
a
b
c
d
e
f
Sub - Problems
Conquer
S_a
S_b
S_c
S_d
S_e
S_f
Sub - Solutions
Merge
S_{abcdef}
Final Solution

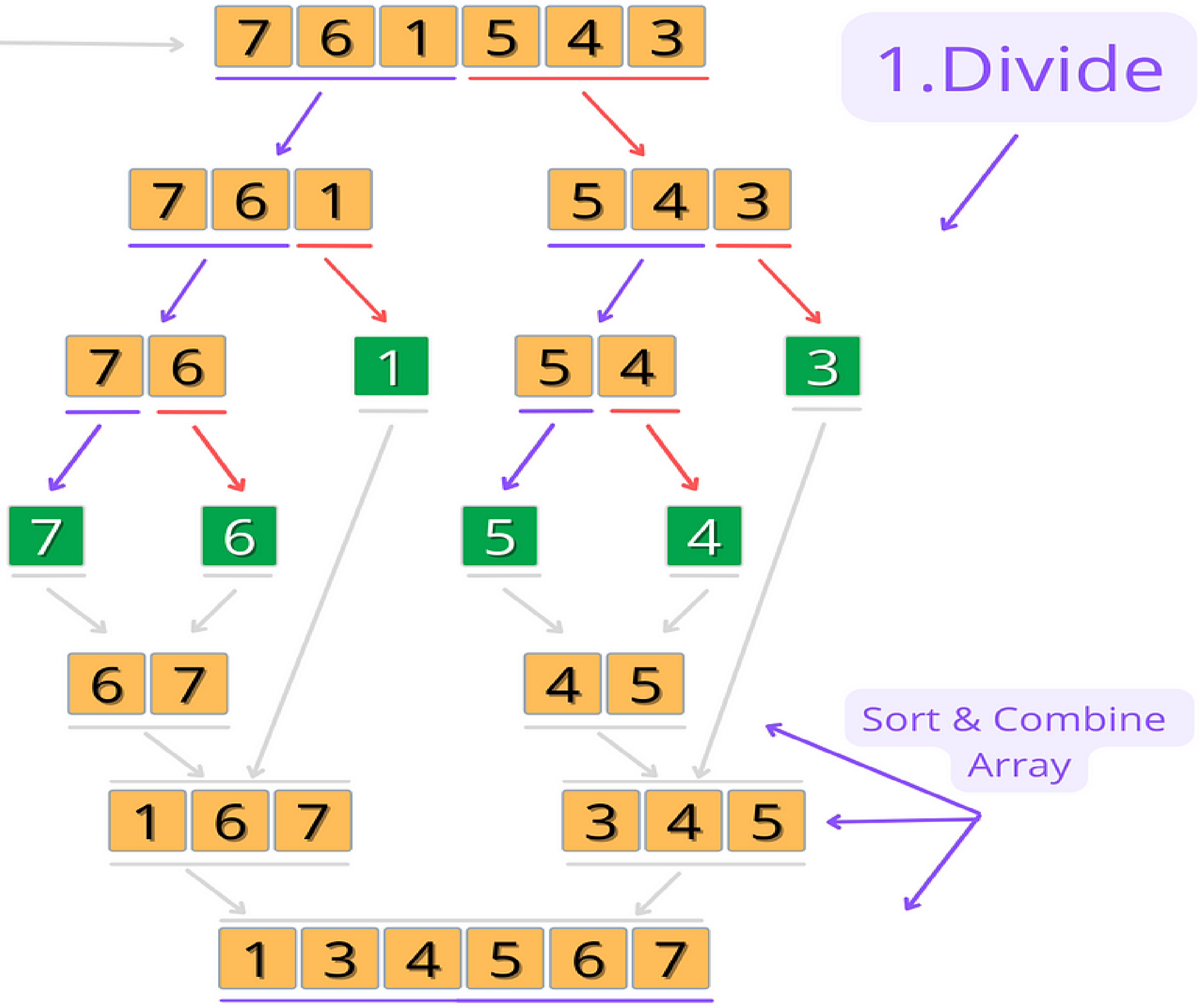
Problem: Sort an Array using **Divide & Conquer** technique

1. Let the given array be

2. Divide the array into two halves.

3. Again, divide each subpart recursively into two halves until we get **individual elements (end-nodes)**

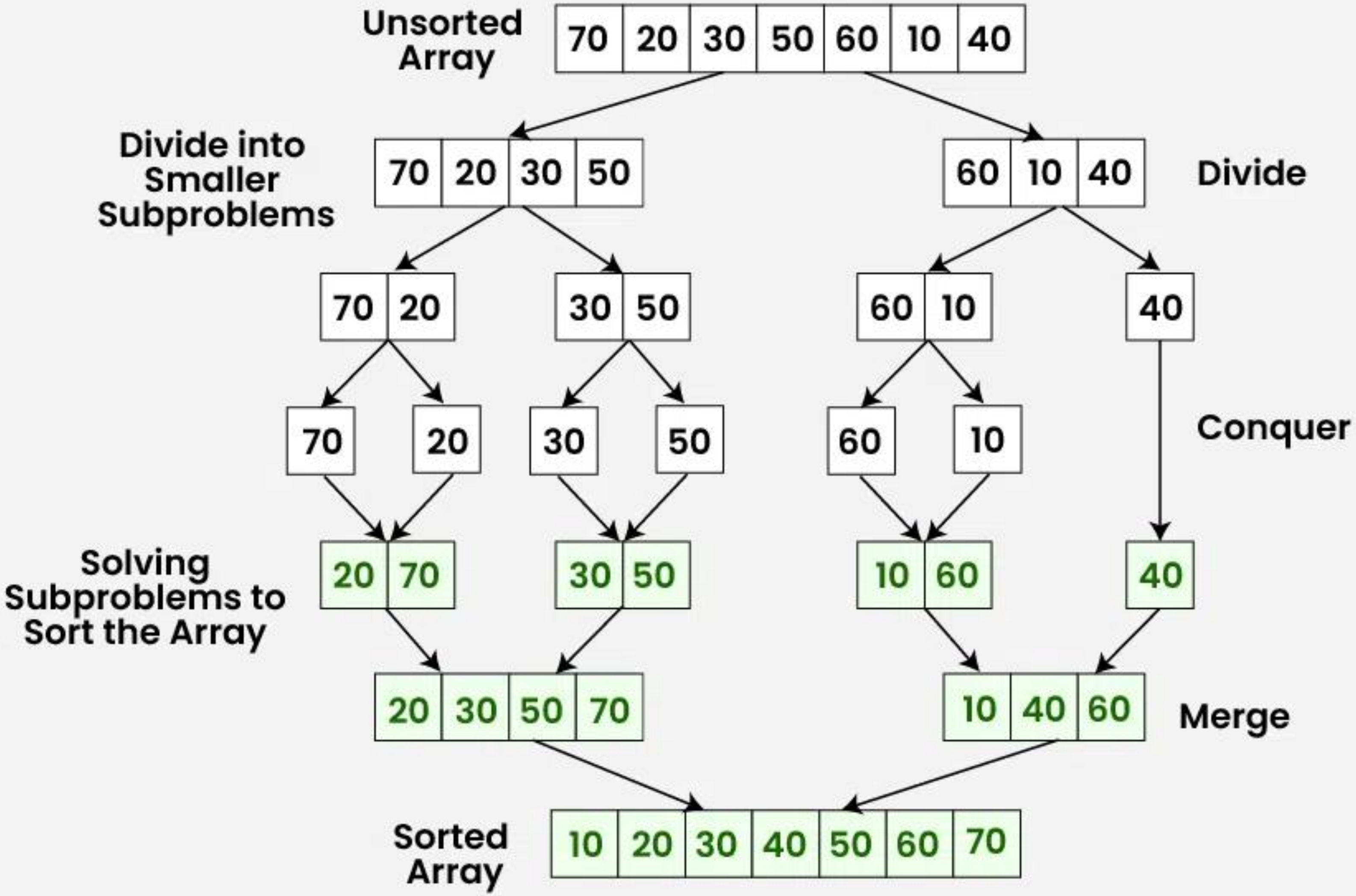
4. Now, sort the elements and combine them individually.

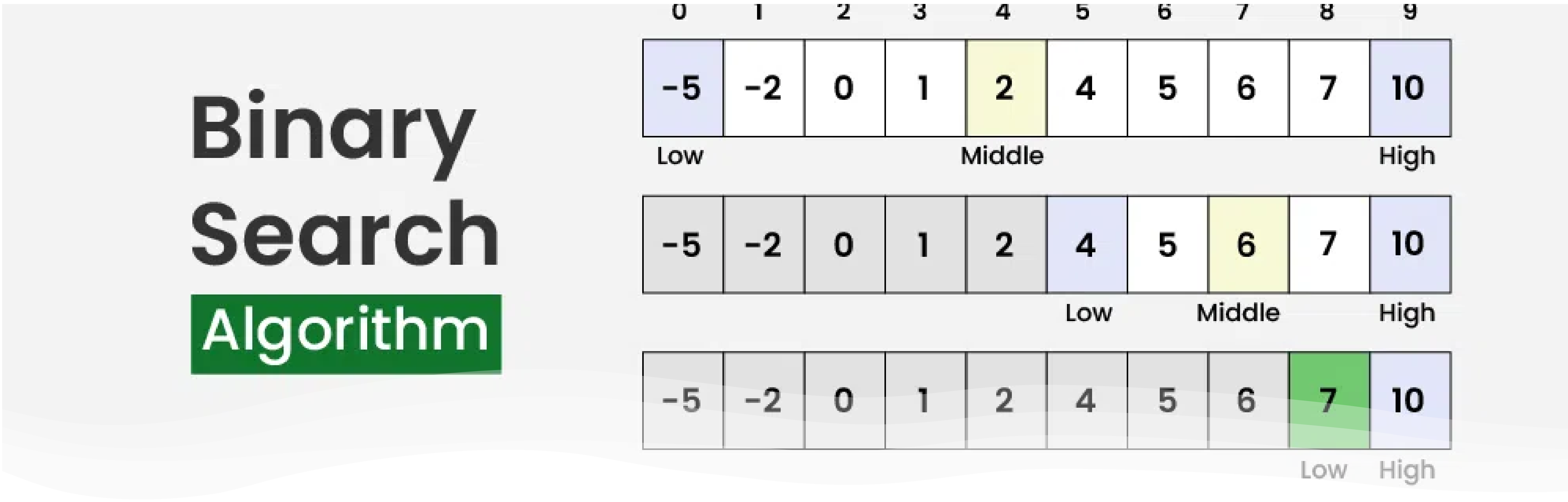


2. Conquer and Combine

Conquer the subproblems by solving them recursively. Combine the solution for subproblems into the solution for original problem.

Working of Divide & Conquer Algorithm



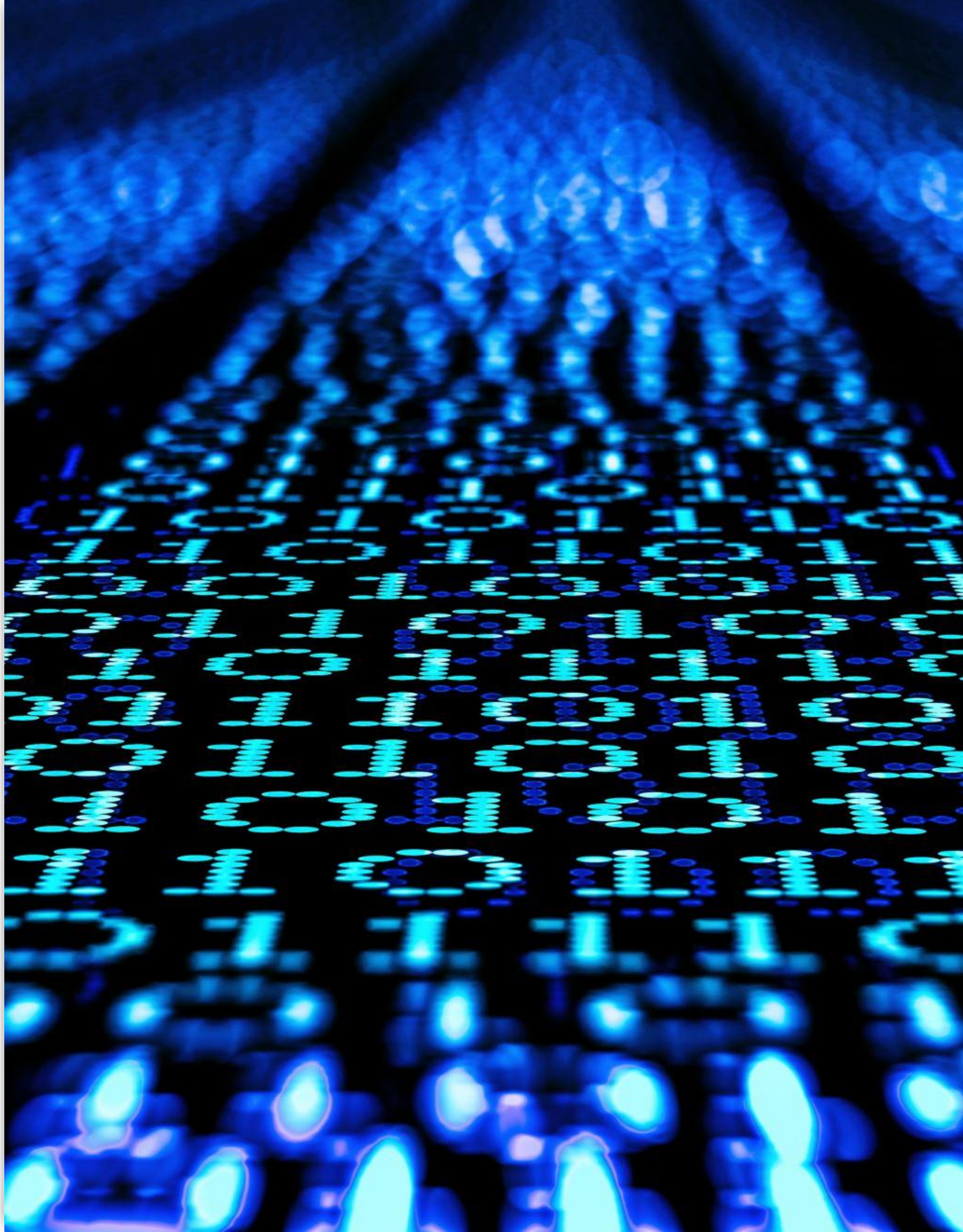


- **Binary Search Algorithm** is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.

Binary Search Algorithm

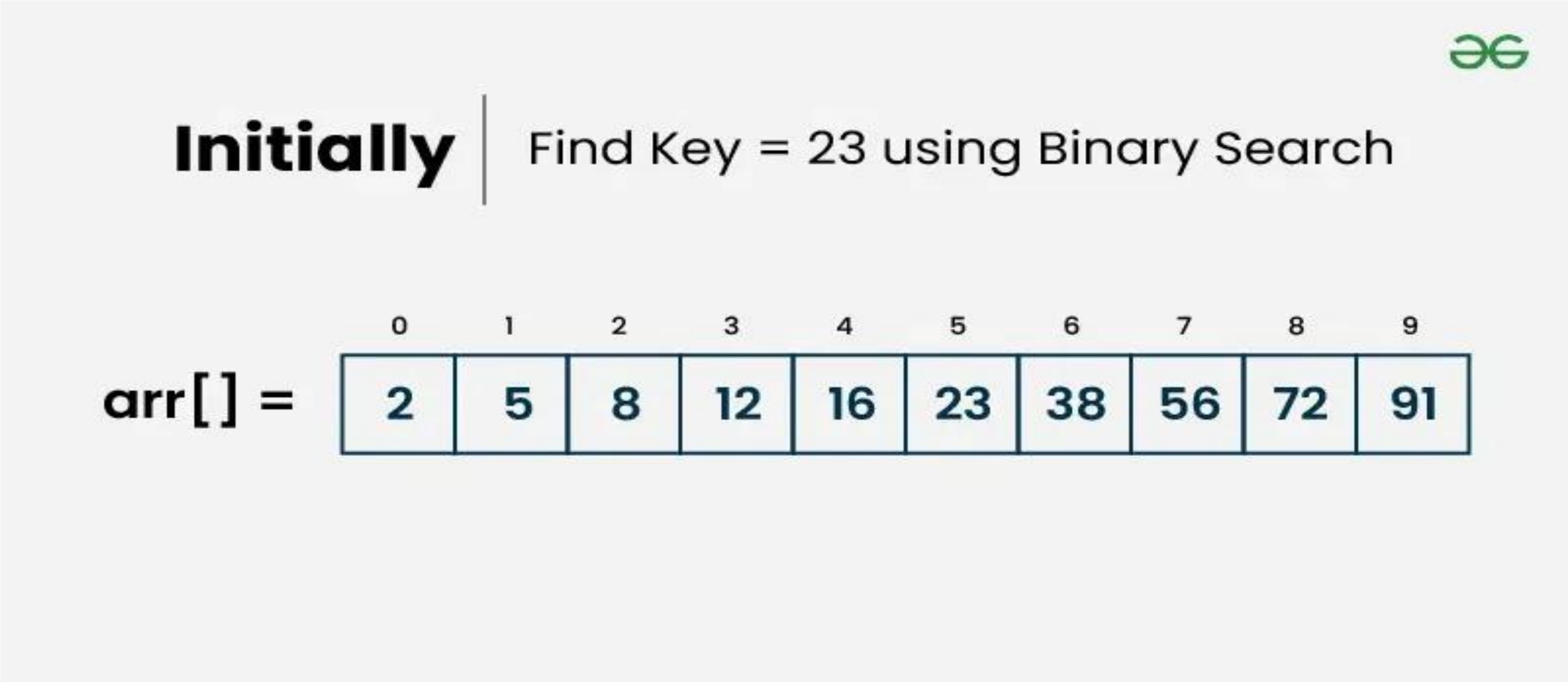
Below is the step-by-step algorithm for Binary Search:

- Divide the search space into two halves by **finding the middle index "mid"**.
- Compare the middle element of the search space with the **key**.
- If the **key** is found at middle element, the process is terminated.
- If the **key** is not found at middle element, choose which half will be used as the next search space.
    - If the **key** is smaller than the middle element, then the **left** side is used for next search.
    - If the **key** is larger than the middle element, then the **right** side is used for next search.
- This process is continued until the **key** is found or the total search space is exhausted.



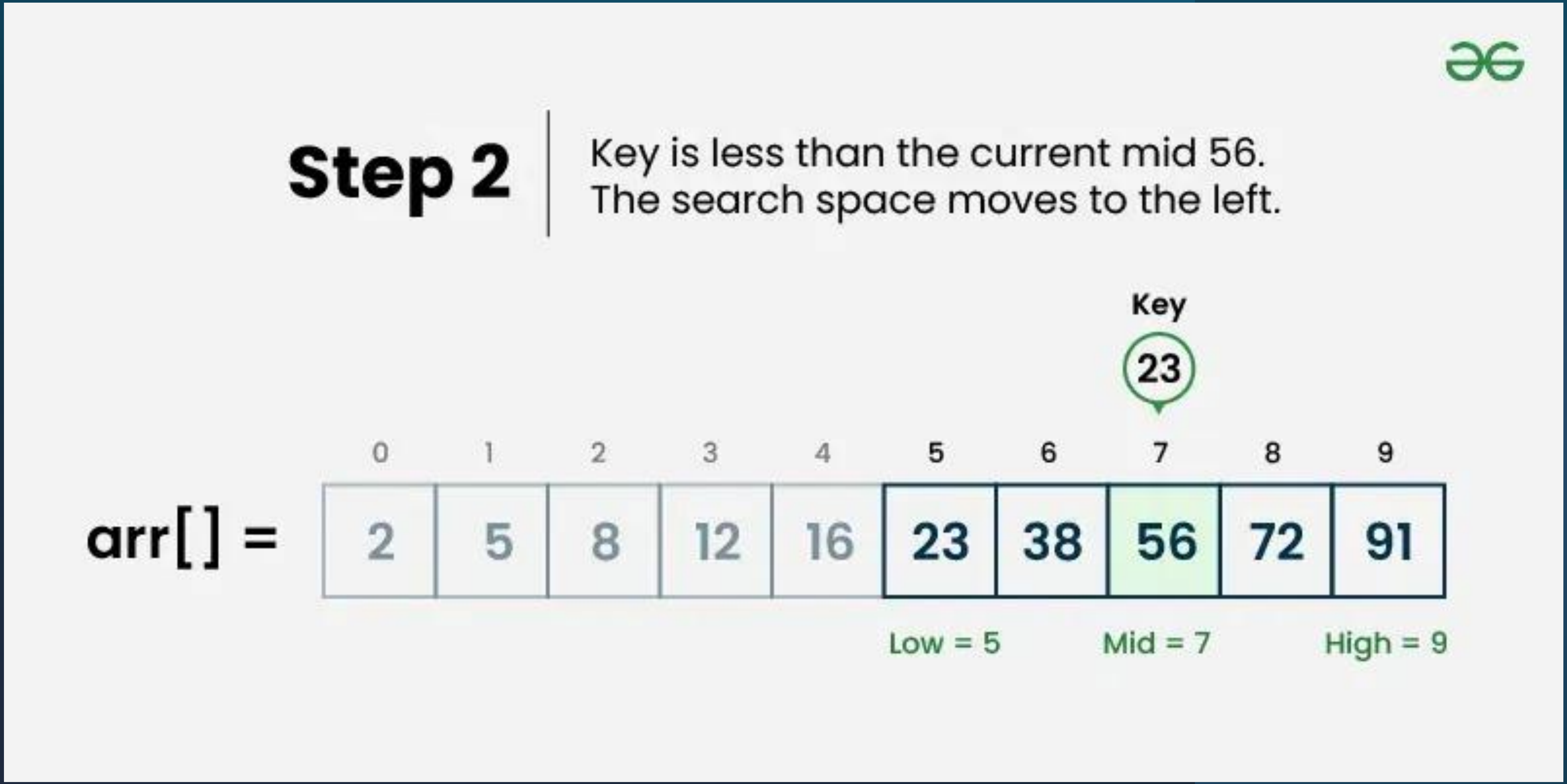
Example

- Consider an array **arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}**, and the **target = 23**.



Step 1

Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.


Step 2
Key is less than the current mid 56.
The search space moves to the left.
Key
23
0 1 2 3 4 5 6 7 8 9
arr[] =
2 5 8 12 16 23 38 56 72 91
Low = 5
Mid = 7
High = 9

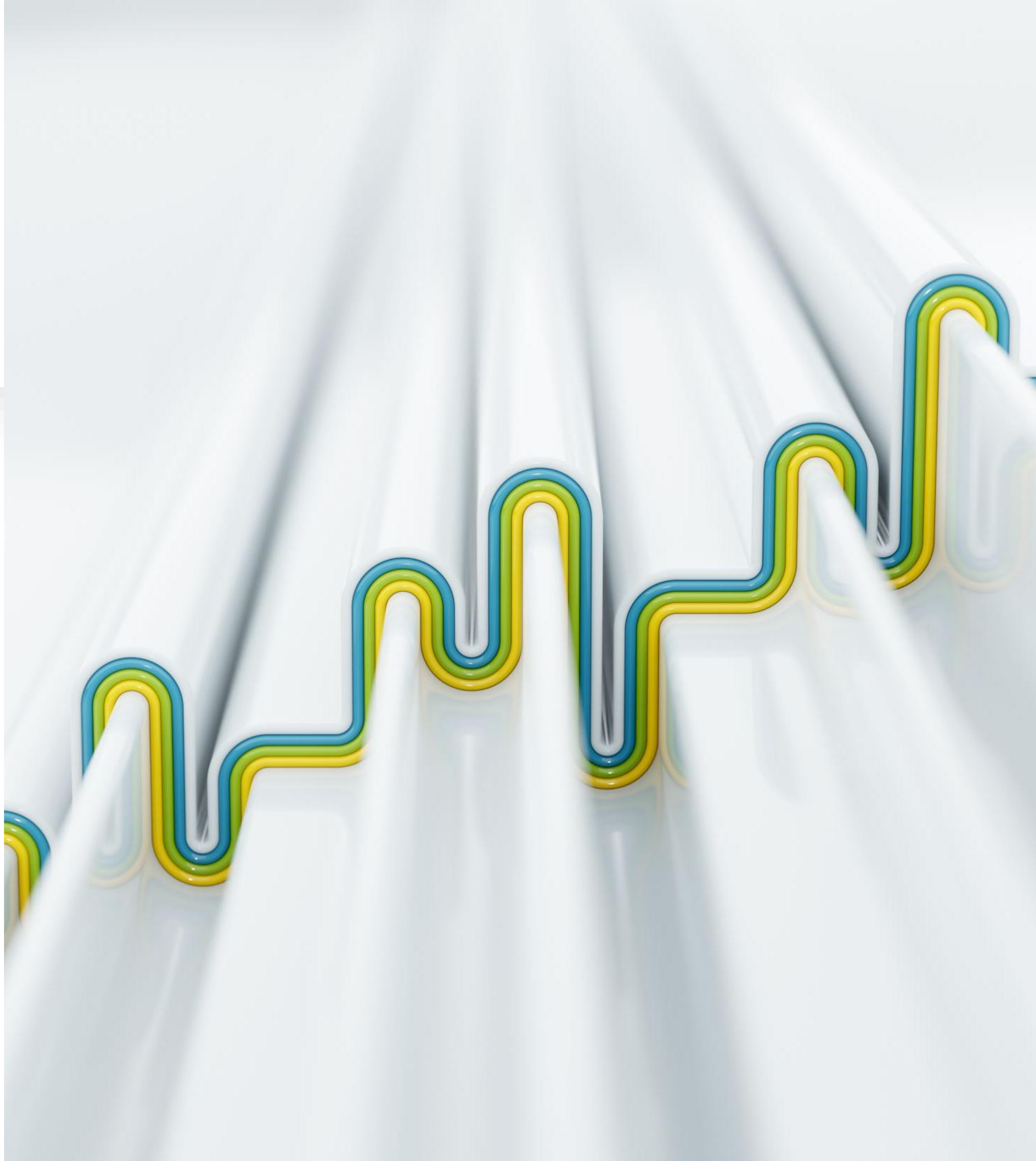
Step 3 | If the key matches the value of the mid element, the element is found and stop search.

Non-Comparison Sorting

- Non-comparison sorting algorithms sort elements without directly comparing them against one another. Unlike traditional comparison-based sorting (like Merge Sort or Quick Sort), these algorithms use element properties, such as digit values or frequency counts, to organize the data.

Key Concept

- These algorithms **rely on assumptions** about the data—often that elements are integers or can be mapped to integers within a known, limited range.
- They typically achieve **linear time complexity**, making them highly efficient for specific kinds of inputs.



Comparison vs Non-Comparison Sorting

Feature	Comparison-Based	Non-Comparison Based
Time Complexity (Best)	O(n log n)	O(n) (under constraints)
Data Type	Any (as long as it's comparable)	Usually integers or mappable to integers
Example Algorithms	Quick Sort, Merge Sort	Radix Sort, Counting Sort, Bucket Sort
Based on	Element-to-element comparison	Value distribution or structure