# Problem Statement: Implementation of Binary Search Algorithm

Heet Trivedi a.k.a Arjun

August 2025

## 1. Introduction

The project focuses on computational methods for solving mathematical problems, including algorithms for searching and processing data. As part of extending the project's algorithmic library, this problem statement outlines the implementation of a binary search algorithm across multiple programming languages. Binary search is a fundamental algorithm that efficiently locates a target value in a sorted array or list by repeatedly dividing the search interval in half, serving as an optimized alternative to linear search for large datasets.

## 2. Problem Description

The objective is to develop a robust and consistent implementation of the binary search algorithm in the following programming languages: C, C++, Java, Python, Go, Rust, Kotlin, MATLAB, Scala, Perl, Ruby, Julia, and Haskell. The algorithm must efficiently locate a target value within a sorted array or list and return its index, adhering to the project's proprietary restrictions.

### 2.1. Algorithm Overview

- **Binary Search**: Searches a sorted array/list by repeatedly dividing the search interval in half, comparing the middle element to the target value and narrowing the search to the left or right half accordingly.
- **Input**:
  - A sorted array or list of elements (e.g., integers, in ascending order).
  - A target value to search for.
- **Output**:
  - The index of the target value if found (0-based indexing, or 1-based for MATLAB).
  - $-1$ if the target value is not present in the array/list.

- **Time Complexity**: $O(\log n)$, where $n$ is the length of the array/list.
- **Space Complexity**: $O(1)$ for iterative implementation.
- **Example**:
  - Input: Array $[1, 3, 4, 7, 9]$, Target 9
  - Output: 4 (index of 9)

## 2.2. Requirements

- **Functionality**:
  - Implement a function/method named `binary_search` (or language-appropriate equivalent) that accepts a sorted array/list and a target value.
  - Return the index of the first occurrence of the target or $-1$ if not found.
  - Use an iterative approach to minimize space complexity.
- **Input Constraints**:
  - Array/list contains integers, sorted in ascending order (adaptable to other comparable types if needed).
  - Array/list size: $1 \leq n \leq 10^6$.
  - Target value is an integer within the range of array elements.
- **Output Format**:
  - Print a message: "Target `<target>` found at index: `<index>`" (e.g., "Target 9 found at index: 4").
- **Language-Specific Guidelines**:
  - Use idiomatic constructs for each language (e.g., `for` loops in C, `while` loops in Python, `div` in Haskell).
  - Ensure portability and minimal dependencies (no external libraries required).
  - Account for MATLAB's 1-based indexing where applicable.
- **Test Case**:
  - Use sorted array $[1, 3, 4, 7, 9]$ and target 9 for consistency across implementations.

# 3. Scope

- **Included Languages**: C, C++, Java, Python, Go, Rust, Kotlin, MATLAB, Scala, Perl, Ruby, Julia, Haskell.

- **Excluded Languages**:
  - **Solidity**: Unsuitable for general-purpose algorithms due to its blockchain focus.
  - **Cirq/Qiskit**: Python-based quantum computing frameworks; their implementation would be redundant with Python's.

# 4. References


* Cormen, T. H., et al. (2009). *Introduction to Algorithms*. MIT Press.
* Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.