ALGORITHM

- Input: A sorted array of book titles, ‘book_list’ of n elements, and a target book title, ‘target_book’.
- Output: The index of target_book in book_list, or -1 if target_book is not found.

1. Set low = 0
2. Set high = n - 1
3. Set loc = -1
4. While low <= high do:
    a) Set mid = (low + high) / 2
    b) If book_list[mid] == target_book:
        I. Set loc = mid
        II. Break
    c) Else If book_list[mid] < target_book:
        I. Set low = mid + 1
    d) Else:
        I. Set high = mid - 1

ALGORITHM

5. If loc >= 0:
   a) Print "Book is found at location 'loc' and searching is successful."

6. Else:
   a) Print " 'loc' Book is not found and searching is unsuccessful."

Time Complexity Analysis Binary Search

- **Best Case**: O(1) – Target is the middle element in the first step
- **Average Case**: O(log n) – As the search is halved each time, it takes $\log_2(n)$ steps on average.
- **Worst Case**: O(log n) – Target is not found, or it's in the last possible half.
- Binary search is faster than linear search O(n), as binary search minimizes comparisons.

Why is Time Complexity O(log n)?

- Logarithmic Reduction: Each iteration reduces the search space by half.

- Formula Derivation: For an array size of n, after each split:
  - n/2 ... ≈ reaches 1
  - Formula: $n/2^k$ where *k* is the number of steps or divisions we make.
  - the process continues until $n/2^k = 1$
  - we multiply both sides by $2^k = n = 2^k$
  - To solve for *k*, we use the base-2 logarithm on both sides: $\log_2(n) = \log_2(2^k)$
  - $\log_2(n) = k$
  - So $k = \log_2(n)$

Example 1

- n = [2,5,7,12,15,18,21,24,30, 35]
- Target element = 15
- **Step 1:**
    - Low = 0, high = 9
    - Mid = (low + high) //2 = 4
    - Element at index 4 is 15 (target found)
- **Result**: Found 15 in just 1 step (best case)

Example 2

- n = [2,5,7,12,15,18,21,24,30, 35]
- Target element = 21
- **Step 1:**
  - Low = 0, high = 9
  - Mid = (low + high) //2 = 4
  - Element at index 4 is 15 (less than 21)
  - Set low = 5
- **Step 2 low = 5, high = 9:**
  - Mid = (low + high) //2 = 7
  - Element at index 7 is 24 (greater than 21)
  - Set high = 6

Example 2 continue

- n = [2,5,7,12,15,18,21,24,30, 35]
- Target element = 21
- **Step 3 low = 5, high = 6:**
  - Mid = (low + high) //2 = 5
  - Element at index 5 is 18 (less than 21)
  - Set low = 6
- **Step 4: low = 6, high = 6**
  - Mid = (low + high) //2 = 6
  - Element at index 6 is 21 (target found)

- **Result**: Found 21 in 4 steps (worst case).

O(log n)

- $n = [2,5,7,12,15,18,21,24,30, 35]$
- $n = 10$
- The number of steps in binary search is roughly $\log_2(10) \approx 3.32$ so we round up to 4
- This tells us that on average and in the worst cases, we need at most 4 steps

Space Complexity of Binary Search

- Iterative Binary Search: O(1) space, as it only needs a few variables for indices.
- Recursive Binary Search: O(log n) space, as it requires stack space for each recursive call.
- The iterative approach is more space-efficient, while recursion provides a simpler code structure.