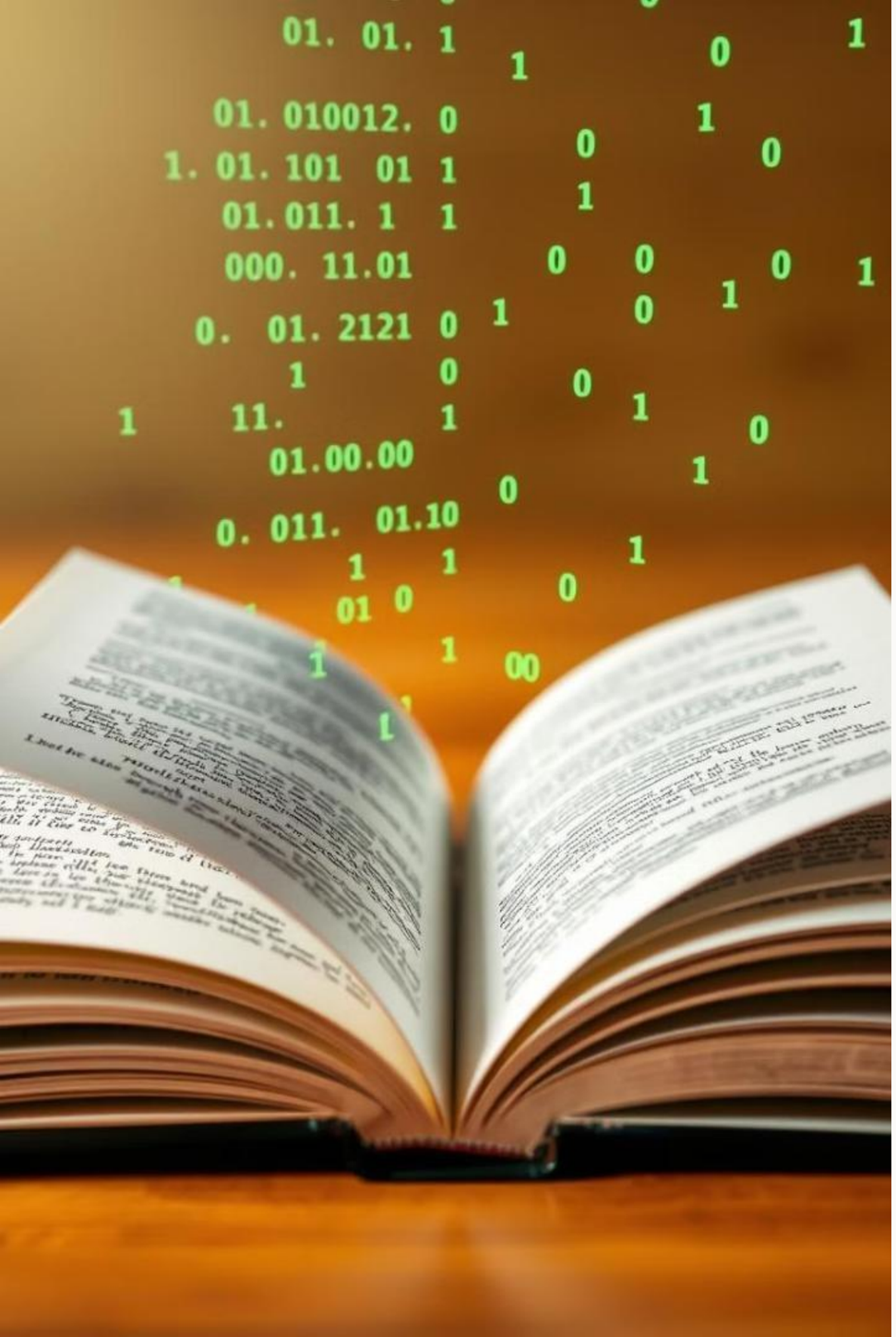
Asansol Engineering College

Topic : BINARY SEARCH USING RECURSION

Presented by **SOUMANKA PAUL**

Dept : Information Technology

University Roll : 10800223147



Binary Search :

Binary Search is an efficient algorithm used to find the position of a specific value in a sorted list (or array). It works by repeatedly dividing the search interval in half. If the value of the target is less than the item in the middle of the interval, it narrows the search to the left half. Otherwise, it narrows it to the right half. This process continues until the value is found or the search interval is empty.

Array: [1, 4, 7, 12, 18, 23, 35, 42, 56, 78]

Binary Search Algorithm

Steps of Binary Search:

1. Start with the entire array (sorted).

2. Find the middle element of the array.

3. Compare the middle element with the target value:

1. ***If the middle element equals the target, you've found the target.***
2. ***If the target is smaller than the middle element, repeat the process on the left half of the array.***
3. ***If the target is larger than the middle element, repeat the process on the right half of the array.***

4. Repeat this process until the target is found or the array can no longer be divided (i.e., the range is empty).

Working of Binary Search with the help of an Example:

Consider the sorted array:
[1, 3, 7, 12, 18, 23, 35, 42, 56, 78]
We are searching for the number 23.

Step 1:
•Low: 0, High: 9
•Middle index: (0 + 9) // 2 = 4
•Middle element: 18
•23 > 18, so search the right half.

Step 2:
•Low: 5, High: 9
•Middle index: (5 + 9) // 2 = 7
•Middle element: 42
•23 < 42, so search the left half.

Step 3:
•Low: 5, High: 6
•Middle index: (5 + 6) // 2 = 5
•Middle element: 23
•Found it! 23 is at index 5

Code Implementation :

```c
int binary_Search(int arr[], int size, int target) {
    int left = 0, right = size - 1;

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

        // Check if target is at mid
        if (arr[mid] == target)
            return mid;

        // If target is greater, ignore left half
        if (arr[mid] < target)
            left = mid + 1;

        // If target is smaller, ignore right half
        else
            right = mid - 1;
    }
    return -1; // Target not found
}
```

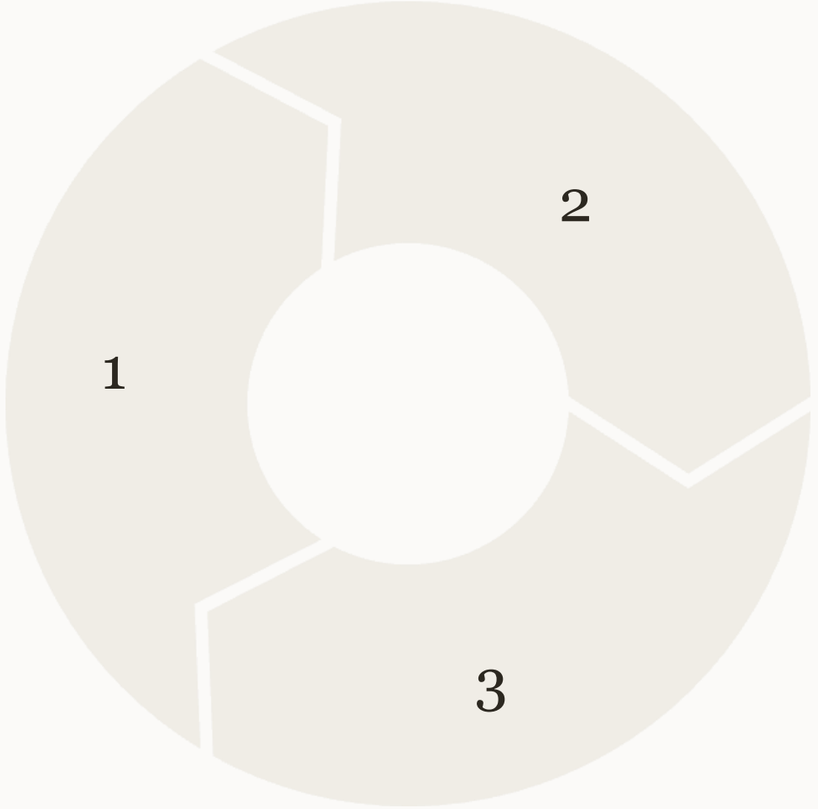
Code Implementation (Recursive Approach)

```
def binary_search_recursive(arr, target, low, high):
    if high >= low:
        mid = low + (high - low) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            return binary_search_recursive(arr, target, low, mid - 1)
        else:
            return binary_search_recursive(arr, target,mid + 1, high)
            else:
            return -1
```

Time Complexity Analysis of Binary Search

Best Case

O(1): Target found in the middle on first try.



Average Case

O(log n): Logarithmic time with each split.

Worst Case

O(log n): Target not found after repeated splits.

A real-life example of binary search is looking up a word in a dictionary:

1.You start in the middle of the dictionary and check if the word is before or after the current word.

2.Depending on the result, you then look at the middle of the half where the word might be.

3.Repeat the process until you find the word or confirm it's not in the dictionary.

THANK YOU