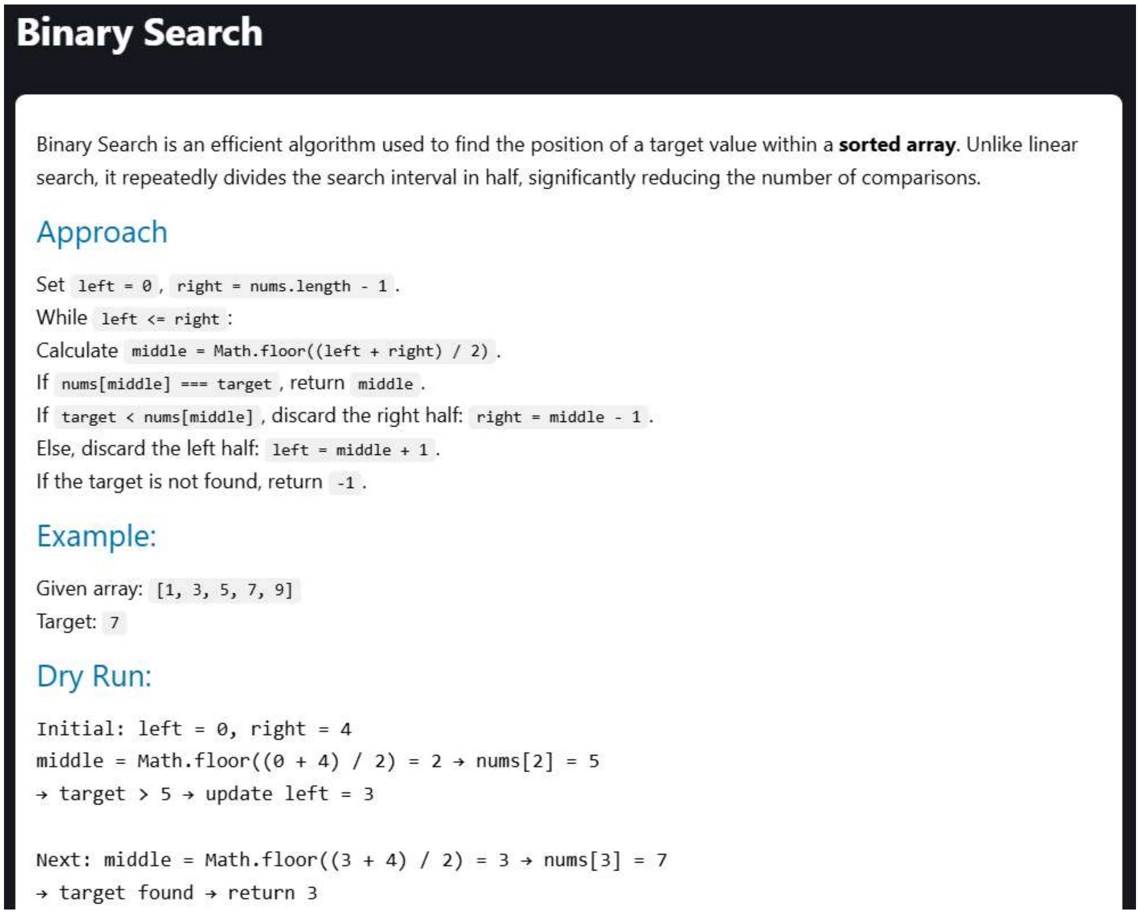


Binary Search

Binary Search is an efficient algorithm used to find the position of a target value within a **sorted array**. Unlike linear search, it repeatedly divides the search interval in half, significantly reducing the number of comparisons.

Approach

Set `left = 0`, `right = nums.length - 1`.
While `left <= right`:
Calculate `middle = Math.floor((left + right) / 2)`.
If `nums[middle] === target`, return `middle`.
If `target < nums[middle]`, discard the right half: `right = middle - 1`.
Else, discard the left half: `left = middle + 1`.
If the target is not found, return `-1`.

Example:

Given array: `[1, 3, 5, 7, 9]`
Target: `7`

Dry Run:

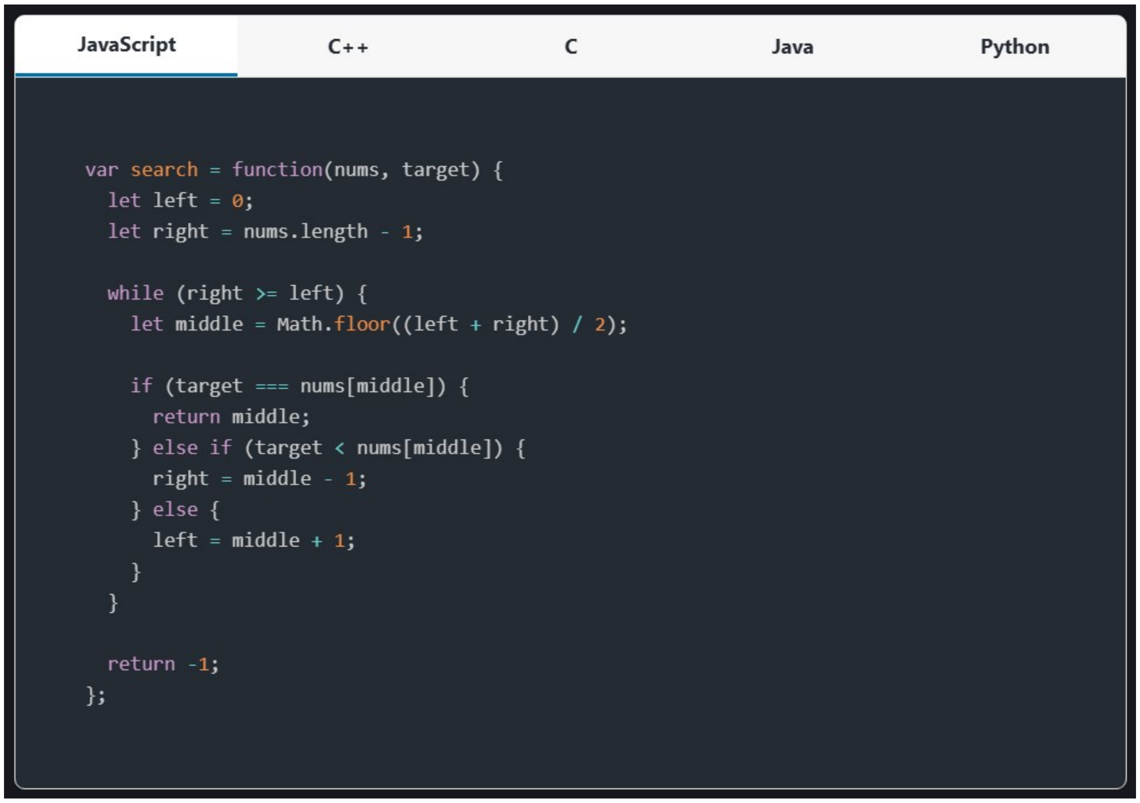
```
Initial: left = 0, right = 4
middle = Math.floor((0 + 4) / 2) = 2 → nums[2] = 5
→ target > 5 → update left = 3

Next: middle = Math.floor((3 + 4) / 2) = 3 → nums[3] = 7
→ target found → return 3
```

Time Complexity:

Best Case: O(1) – when the target is found at the middle initially
Worst Case: O(log n) – the array is halved every iteration

Space Complexity:

O(1) – constant space is used (no additional data structures)



```javascript
var search = function(nums, target) {
  let left = 0;
  let right = nums.length - 1;

  while (right >= left) {
    let middle = Math.floor((left + right) / 2);

    if (target === nums[middle]) {
      return middle;
    } else if (target < nums[middle]) {
      right = middle - 1;
    } else {
      left = middle + 1;
    }
  }

  return -1;
};
```