Binary search is a very efficient algorithm used to find a target element in a sorted array. It works by dividing the search space in half after each comparison. This drastically reduces the number of comparisons needed compared to a linear search.

Key Concepts:

1. Sorted Array: Binary search only works on arrays where the elements are sorted in order.
2. Divide and Conquer: The idea is to repeatedly divide the search space (array) in half and check which half the target value could be in.

Steps:

1. Start with the whole array and two pointers:
    - low at the start (0 index) of the array.
    - high at the end (last index) of the array.
2. Calculate the middle element's index:
    - middle = Math.floor((low + high) / 2)
3. Compare the middle element with the target value (no):
    - If the middle element is equal to the target, you've found it.
    - If the middle element is greater than the target, the target must be in the left half. So, update the high pointer to middle - 1.
    - If the middle element is less than the target, the target must be in the right half. So, update the low pointer to middle + 1.
4. Repeat the process until either:
    - You find the element, or
    - The low pointer becomes greater than the high pointer, meaning the element is not present.

Code :

```
function binarySearch(arr, no) {
   let low = 0;
   let high = arr.length - 1;

   while (low <= high) {
      let middle = Math.floor((low + high) / 2);  // Find the middle index

      if (arr[middle] === no) {  // If the middle element is the target
         return `Element found at index ${middle}`;
      }
      else if (arr[middle] < no) {  // If target is greater, search the right half
         low = middle + 1;
      }
      else {  // If target is smaller, search the left half
         high = middle - 1;
      }
   }

   return "Element not found";  // Return when the element is not in the array
}

const arr = [1, 2, 4, 6, 7, 12, 90];  // Sorted array
const no = 6;  // Target value

console.log(binarySearch(arr, no));  // Should print: Element found at index 3
```

======================

Important Notes:

1. Array must be sorted before you can use binary search. If the array isn't sorted, you either need to sort it or use a different algorithm (like linear search).
2. Binary search is much faster for large arrays compared to linear search.

=======================

Let's walk through the **binary search** step by step with the array `arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]` and the target value `7`.

Step-by-step Process:

Initial State:

- **Array:** `[1, 2, 3, 4, 5, 6, 7, 8, 9]`
- **Target:** `7`
- **low:** Initially set to the first index of the array (`low = 0`).
- **high:** Initially set to the last index of the array (`high = 8`).

Step 1:

1. Calculate the middle index:
   - `middle = Math.floor((low + high) / 2) = Math.floor((0 + 8) / 2) = 4`
2. Check the middle element:
   - `arr[middle] = arr[4] = 5`
3. Compare the middle element with the target value `7`:
   - Since `5 < 7`, the target must be in the **right half** of the array.
   - Update `low = middle + 1 = 4 + 1 = 5`.

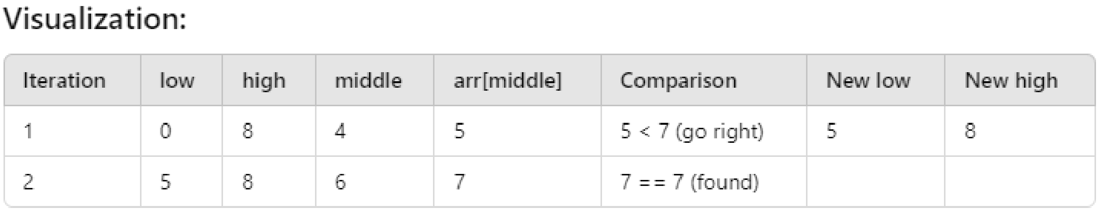
Step 2:

1. Now, `low = 5` and `high = 8`.

2. **Calculate the new middle index:**
   - `middle = Math.floor((low + high) / 2) = Math.floor((5 + 8) / 2) = 6`
3. **Check the middle element:**
   - `arr[middle] = arr[6] = 7`
4. **Compare the middle element with the target value** `7`**:**
   - Since `arr[middle] == 7`, **we have found the target** at index `6`.

Summary:

- In just **two iterations**, we found the target element `7` at index `6`.

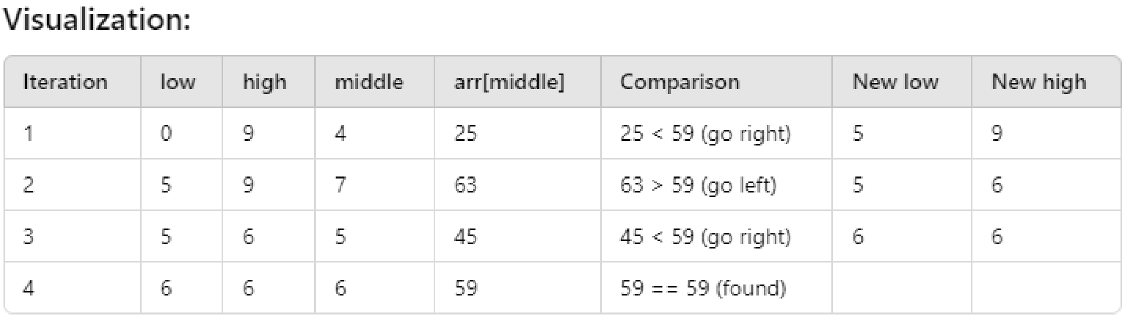


Visualization:

Iteration	low	high	middle	arr[middle]	Comparison	New low	New high
1	0	8	4	5	5 < 7 (go right)	5	8
2	5	8	6	7	7 == 7 (found)		

Eg : 2

Array: `[5, 9, 17, 23, 25, 45, 59, 63, 71, 89]`
Target: `59`



Visualization:

Iteration	low	high	middle	arr[middle]	Comparison	New low	New high
1	0	9	4	25	25 < 59 (go right)	5	9
2	5	9	7	63	63 > 59 (go left)	5	6
3	5	6	5	45	45 < 59 (go right)	6	6
4	6	6	6	59	59 == 59 (found)		