JAVA PROGRAM

Binary Search

Using Iterative Approach

Binary Search In Java With Code:

Binary Search in Java with code and implementation

Code:

```java
public class BinarySearch {
    public static void main(String[] args) {
        int[] array = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
        int target = 4;

        int left = 0;
        int right = array.length - 1;
        int mid = 0;

        boolean found = false;

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

            if (array[mid] == target) {
                found = true;
                break;
            } else if (array[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        if (found) {
            System.out.println("Element found at index " + mid);
        } else {
            System.out.println("Element not found in the array.");
        }
    }
}
```

Explanation

1. We define a class named **BinarySearchExample**.
2. In the **main()** method, we declare and initialize an array called **array** with sorted values.
3. We set the **target** variable to the element we want to find.
4. We initialize the **left** variable to 0, representing the leftmost index of the search range.
5. We initialize the **right** variable to **array.length - 1**, representing the rightmost index of the search range.
6. We declare a **mid** variable to store the index of the middle element.

7. We initialize the **found** variable to **false** to keep track of whether the target element is found.
8. We enter a **while** loop that continues as long as the **left** index is less than or equal to the **right** index.
9. Inside the loop, we calculate the **mid** index as the average of the **left** and **right** indices.
10. We compare the element at the **mid** index with the **target** element. If they are equal, we set **found** to **true** and break out of the loop.
11. If the element at the **mid** index is less than the **target**, we update the **left** index to **mid + 1** to search the right half of the array.
12. If the element at the **mid** index is greater than the **target**, we update the **right** index to **mid - 1** to search the left half of the array.
13. After exiting the loop, we check the value of **found**. If it is **true**, we display a message indicating that the element was found at index **mid**. Otherwise, we display a message indicating that the element was not found in the array.

In this example, the binary search algorithm efficiently searches for the target element by repeatedly dividing the search range in half until the element is found or the range becomes empty.

Output:

Upon executing the program, the output will be:

```
Element found at index 1
PS D:\DSA\LinkedList>
```