Technique: Record and Move On

Level: Easy

Given a sorted array of Integers, find the target. If the target is not found, return the element closest to the target.

For example,
A = [1,2,4,5,7,8,9], Target = 6 -> Output Index = 3 or 4 (since both 5 and 7 are equally close)

Questions to Clarify:
Q. What if there are two elements equally distant from target?
A. Return either one.

Q. How do you want to return the output?
A. Return the index of the closest element.

Q. What to return if array is empty or null?
A. Return -1.

Solution:
Every time we find a new mid, we are trying to move closer to the target.
If we keep track of the closest mid we have encountered, that will give us closest element to the target.

The reasoning for this is as follows: Let's take this example:

`[1,2,4,5,7,8,9], target = 6`

In the first iteration of binary search, we reach 5. We then go to the second half, since 6 is greater than 5. We consider 5 as a candidate for closest element, but anything in the first half is not a candidate, because they are further away from 6 than 5 is. This applies to every iteration of the search. Hopefully this gives you an intuition of why we only consider mids.

Pseudocode:
(Note: Never write pseudocode in an actual interview. Unless you're writing a few lines quickly to plan out your solution. Your actual solution should be in a real language and use good syntax.)

```
result = null
while low <= high:
     find mid
     if result is null or mid is closer to target than previous result
         reassign result to mid
      carry on with binary search step
```

Note: The only difference between a normal binary search and this algorithm is the part where you record the mid.

Test Cases:
Edge Cases: empty array, null array
Base Cases: single element (equal/not-equal to target)
Regular Cases: has equal element, no equal element, closer element at end/beginning

Time Complexity: O(log(n))

Space Complexity: O(1)

```
public static int closestElement(int[] a, int target) {
    if (a == null) {
        return -1;
    }

    int low = 0, high = a.length - 1;
    int result = -1;
    while (low <= high) {
        int mid = low + ((high - low) >> 1);
        result = record(a, mid, result, target);
        if (a[mid] > target) {
            high = mid - 1;
        } else if (a[mid] < target) {
            low = mid + 1;
        } else {
            return mid;
        }
    }

    return result;
}

private static int record(int[] a, int mid, int result, int target) {
    if (result == -1
            || Math.abs(a[mid] - target) < Math.abs(a[result] - target))
        return mid;

    return result;
}
```