Level: Easy

Q. Given a sorted array, search for a target item.

Implementation

You should learn one binary search implementation well.

Why learn well?
Because no one is going to directly ask you to directly implement binary search.

They will ask a problem that uses it. If you know one implementation well, you can apply it to other problems rather quickly.

We recommend the iterative implementation we have provided.

There are a few things to notice here:

First:
Instead of using `mid = (start + end)/2` to calculate the midpoint,
we use `mid = start + (end - start)/2`. This has become a popular question.

Why do we use this?
Let's say `start` and `end` were very large integers. We know they cannot be larger than `2^31 - 1 (~2 billion)`, because they are given to us as integers (assuming integer size of 32 bytes).
However, their sum could be larger. This is called an **integer overflow**.

In an integer overflow, `start + end` would wrap around the max value into the negatives. If it is an unsigned integer, the value would wrap around 0.

If you're not sure how that works, please lookup and understand integer overflow in your language.

Second:
To divide n by 2, we can use `n >> 1` (bit shift operator) instead of using `n/2`. While this doesn't make much of a difference, it's one of those things you should know. Remember to surround the `n >> 1` with parentheses, because `>>` has a lower operator precedence than `+`.

Our expression can be: `mid = start + ((end - start) >> 1)`

Questions to Clarify:
Q. How do you want the output?
A. Return the index of the target item.

Q. What do do if target was not found?
A. Return -1

Time Complexity: O(log(n))
Remember, log(n) here is base 2, because we are dividing the work by 2 each time.

Space Complexity: O(1)

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

    int low = 0, high = a.length - 1;

    while (low <= high) {
        int mid = low + (high - low)/2;
        if (a[mid] < target) {
            low = mid + 1;
        } else if (a[mid] > target) {
            high = mid - 1;
        } else {
            return mid;
        }
    }

    return -1;
}
```

Java Users: Below, we use the Comparable interface in Java. This can be used for comparing any Object class. For Java users, the Comparable interface is a must-know.

```
public static <A extends Comparable<A>> int search(A[] a, A target) {
    if (a == null || target == null) {
        return -1;
    }

    int low = 0, high = a.length - 1;

    while (low <= high) {
        int mid = low + (high - low)/2;
        if (a[mid].compareTo(target) == 0) {
            return mid;
        } else if (a[mid].compareTo(target) < 0) {
            low = mid + 1;
        } else {
            high = mid - 1;
```

```
            }
        }

        return -1;
}
```