Description

34. Search for a Range

Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

```
Example 1:
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
Example 2:
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Idea

Because the input array is already sorted and we are looking for the target in the range, so my primitive idea is to use binary search to find the first index of the target and do the binary search again to find the last index. O(logn) time and O(1) space.

Java

```
class Solution {
  public int[] searchRange(int[] nums, int target) {
      int[] res = new int[]{-1, -1};
      if (nums == null || nums.length == 0) {
          return res;
      }

      //find first target
      int start = 0;
```

```
        int end = nums.length - 1;

        while (start + 1 < end) {
            int mid = (end - start) / 2 + start;
            // we look for the first, so if mid equals target,
            // then there might be more target on left, so end = mid
            if (nums[mid] >= target) {
                end = mid;
            } else {
                start = mid;
            }
        }

        if (start == end && nums.length != 1) {
            System.out.println(start + "," + end);
            return new int[]{-2, -2};
        }

        // since we look for the first, we check the left one first
        if (nums[start] == target) {
            res[0] = start;
        } else if (nums[end] == target) {
            res[0] = end;
        } else {
            return res;
        }

        //find last target
        start = 0;
        end = nums.length - 1;

        while (start + 1 < end) {
            int mid = (end - start) / 2 + start;
            // we look for the first, so if mid equals target,
            // then there might be more target on right, so start = mid
            if (nums[mid] <= target) {
                start = mid;
            } else {
                end = mid;
            }
        }

        if (start == end && nums.length != 1) {
            System.out.println(start + "," + end);
            return new int[]{-2, -2};
        }

        // since we look for the last, we check the right one first
        if (nums[end] == target) {
            res[1] = end;
        } else {
            res[1] = start;
```

```
        }

        return res;
    }
}
```

C++

```
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> res = {-1, -1};
        if (nums.empty()) {
            return res;
        }

        int start = 0;
        int end = nums.size() - 1;

        // find first index
        while (start + 1 < end) {
            int mid = (end - start) / 2 + start;
            if (nums[mid] >= target) {
                end = mid;
            } else {
                start = mid;
            }
        }

        if (nums[start] == target) {
            res[0] = start;
        } else if (nums[end] == target) {
            res[0] = end;
        } else {
            return res;
        }

        start = 0;
        end = nums.size() - 1;
        // find last index
        while (start + 1 < end) {
            int mid = (end - start) / 2 + start;
            if (nums[mid] <= target) {
                start = mid;
            } else {
                end = mid;
            }
        }
```

```
        if (nums[end] == target) {
            res[1] = end;
        } else {
            res[1] = start;
        }
        return res;
    }
};
```

Because we are looking for a range, so we can just use a helper function to search the range directly from 0 to nums.length - 1. While searching, we update the start/mid/end index and if nums[mid] == target, we update our range int[] res. We use [5,7,7,8,8,10] as an example, initially int[] res = {MAX, -1}.

```
5, 7, 7, 8, 8, 10
s     m        e
```

At begining, nums[mid] == 7 which is smaller than the target, so we search from the right part, mid + 1 to end

```
5, 7, 7, 8, 8, 10
         s  m  e
```

The nums[mid] == 8, so we update our res[] index to {4,4}
Because we don't know if mid-1 / mid+1 are also equals target, so we need to search from both sides [start,mid-1] and [mid+1,end]

```
left side:            right side:
5, 7, 7, 8, 8, 10     5, 7, 7, 8, 8, 10
         s                         s
         e                         e
         m                         m
```

For left side, we see nums[mid] == target, we update res[] to {3,4}. For right side, we do not update res[]. Then after moving the indices, start > end, so we stop the recursion and return. O(logn) time and O(1) space. If we consider recursion stack as extra space then O(logn) space.

Java

```java
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] res = new int[]{Integer.MAX_VALUE, -1};
        helper(nums, target, res, 0, nums.length - 1);
        return  res[0] > res[1] ? new int[]{-1, -1} : res;
    }

    private void helper(int[] nums, int target ,int[] res, int start, int end) {
        if (start > end) {
            return;
        }

        int mid = (end - start) / 2 + start;
        if (nums[mid] > target) {
            // search left half
            helper(nums, target, res, start, mid - 1);
        } else if (nums[mid] < target) {
            // search right half
            helper(nums, target, res, mid + 1, end);
        } else {
            if (mid < res[0]) {
                res[0] = mid;
                helper(nums, target, res, start, mid - 1);
            }
            if (mid > res[1]) {
                res[1] = mid;
                helper(nums, target, res, mid + 1, end);
            }
        }
    }
}
```

C++

```cpp
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> res = {INT_MAX, -1};
        helper(nums, target, res, 0, nums.size() - 1);
        return res[0] > res[1] ? vector<int>{-1, -1} : res;
    }

private:
    void helper(vector<int>& nums, int target, vector<int>& res,
                int start, int end) {
```

```
        if (start > end) {
            return;
        }

        int mid = (end - start) / 2 + start;
        if (nums[mid] > target) {
            helper(nums, target, res, start, mid - 1);
        } else if (nums[mid] < target) {
            helper(nums, target, res, mid + 1, end);
        } else {
            if (res[0] > mid) {
                res[0] = mid;
                helper(nums, target, res, start, mid - 1);
            }
            if (res[1] < mid) {
                res[1] = mid;
                helper(nums, target, res, mid + 1, end);
            }
        }
    }
}
```

Summary

- Binary search.
- Divide and conquer.
- Throw away half of the impossible range at a time.