Approach 1: Brute Force

- **Idea:** Iterate through the array from start to finish to find the first occurrence of the target, and then continue to find the last occurrence.
- **Complexity:** O(n)

```cpp
#include <vector>
using namespace std;

class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int first = -1, last = -1;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] == target) {
                if (first == -1) {
                    first = i;
                }
                last = i;
            }
        }
        return {first, last};
    }
};
```

Binary Search Basics:

- **Purpose:** Binary search is an efficient algorithm for finding an item from a sorted list. It works by repeatedly dividing the search interval in half.
- **Precondition:** The array must be sorted.
- **Steps:**
    - Start with two pointers, left and right, which point to the start and end of the array.
    - Compute the mid index.
    - Compare the mid value with the target:
        - If nums[mid] == target, you've found the element.
        - If nums[mid] < target, move the left pointer to mid + 1 (search the right half).
        - If nums[mid] > target, move the right pointer to mid - 1 (search the left half).
    - Continue this process until left exceeds right.

Finding the First and Last Occurrences:

- The findBound function does a binary search for either the first or the last occurrence of the target.
- **First Occurrence (isFirst = true):**
    - If the target is found at mid, we don't stop. Instead, we move the right pointer to mid - 1 to continue searching in the left half. This ensures that even if there are multiple occurrences, we eventually find the leftmost one.
- **Last Occurrence (isFirst = false):**
    - Similarly, if the target is found, we move the left pointer to mid + 1 to continue searching in the right half to find the rightmost occurrence.

Approach 2: Binary Search (Better)

- **Idea:** Use binary search to find the first occurrence and then use binary search again to find the last occurrence. This approach narrows down the search space efficiently.
- **Complexity:** O(log n) for both first and last search.

```cpp
#include <vector>
using namespace std;

class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int first = findFirst(nums, target);
        int last = findLast(nums, target);
        return {first, last};
    }

private:
    int findFirst(const vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        int first = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                first = mid;
                right = mid - 1;  // continue searching in the left half
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return first;
    }

    int findLast(const vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        int last = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                last = mid;
                left = mid + 1;  // continue searching in the right half
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return last;
    }
};
```

Approach 3: Optimized Binary Search (Best)

In this approach, we optimize the search by performing two binary searches, one to find the first occurrence and another to find the last occurrence of the target, but we make the code more streamlined and readable. We achieve this with helper functions that focus specifically on finding the boundary conditions (first and last positions).

```cpp
#include <vector>
using namespace std;

class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int first = findBound(nums, target, true);
        int last = findBound(nums, target, false);
        return {first, last};
    }

private:
    int findBound(const vector<int>& nums, int target, bool isFirst) {
        int left = 0, right = nums.size() - 1;
        int bound = -1;

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

            if (nums[mid] == target) {
                bound = mid;
                // If searching for the first occurrence, move towards the left half
                if (isFirst) {
                    right = mid - 1;
                } else { // If searching for the last occurrence, move towards the right half
                    left = mid + 1;
                }
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return bound;
    }
};
```