Project Overview

This project focuses on two essential algorithms: Selection Sort and Recursive Binary Search. Both are fundamental for problem-solving in computer science. Selection Sort arranges data in ascending order, while Recursive Binary Search is an optimized approach to locate elements in a sorted array using recursion. Additionally, Git and GitHub were utilized for version control and collaboration, fostering a professional development environment.

Selection Sort Algorithm

Algorithm Description:

Selection Sort is a sorting algorithm that repeatedly selects the smallest element from the unsorted portion of the array and swaps it with the first unsorted element.

1. Start with the first element in the array.
2. Iterate through the unsorted portion to find the smallest element.
3. Swap the smallest element with the first unsorted element.
4. Repeat for the remaining unsorted elements until the entire array is sorted.

Python Implementation:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        # Assume the current index is the smallest
```

```
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap the smallest element with the current element
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr
```

Test Example:

```
def test_selection_sort():
    assert selection_sort([64, 25, 12, 22, 11]) == [11, 12, 22, 25, 64]
    assert selection_sort([3, 1, 4, 1, 5, 9]) == [1, 1, 3, 4, 5, 9]
    assert selection_sort([0, -1, 8, 7, -5]) == [-5, -1, 0, 7, 8]
    print("All tests passed!")
```

Recursive Binary Search Algorithm

Algorithm Description:

Recursive Binary Search works by dividing a sorted array into smaller subarrays and searching recursively until the target is found or the subarray is empty.

1. Identify the `low` and `high` indices.
2. Calculate the middle index as the average of `low` and `high`.
3. Compare the middle element with the target:
   - If it matches, return the index.

- If it is greater, recursively search the left half.
- If it is smaller, recursively search the right half.

4. Stop when the indices overlap or the target is found.

Python Implementation:

```
def recursive_binary_search(arr, target, low, high):
    if low > high:
        return -1  # Base case: element not found
    mid = (low + high) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] < target:
        return recursive_binary_search(arr, target, mid + 1, high)
    else:
        return recursive_binary_search(arr, target, low, mid - 1)
```

Test Example:

```
def test_recursive_binary_search():
    assert recursive_binary_search([1, 2, 3, 4, 5, 6], 4, 0, 5) == 3
    assert recursive_binary_search([10, 20, 30, 40], 25, 0, 3) == -1
    assert recursive_binary_search([2, 4, 6, 8, 10], 8, 0, 4) == 3
    print("All tests passed!")
```

Conclusion

The project highlights the implementation of Selection Sort and Recursive Binary Search algorithms. These methods are not only foundational but also illustrate diverse problem-solving approaches-iterative for sorting and recursive for searching. Using Git and GitHub has strengthened team collaboration and code management, promoting professional practices. Through this project, we emphasized the importance of clear algorithm design, rigorous testing, and effective version control in software development.