Name: _____

1. Recursive Binary Search

In this question, we will implement a recursive version of `binary_search`. Rather than keeping track of the search range using a loop, our recursive implementation takes two additional parameters: `left` and `right`, which indicate the current search range.

Suppose we had the following sorted list:

```
numbers = [2, 5, 8, 11, 15, 18, 21, 24, 29, 33, 37, 42, 47, 51, 56]
```

a. Write the indices that are visited after running `binary_search(numbers, 42, 0, len(numbers) - 1)`.

b. Write the indices that are visited after running `binary_search(numbers, 20, 0, len(numbers) - 1)`.

c. Now, let's implement `binary_search` recursively.

```
def binary_search(numbers, target, left, right):
    if ______________________________:
        return ________________________
    mid = ____________________________
    if ___________________ == ___________________:
        return ________________________
    elif ___________________ < ___________________:
        return ____________________________________________
    else:
        return ____________________________________________
```

2. Complexity

This problem was adapted from UC Berkeley's CS 61B course.

a. Order the following $O(\cdot)$ runtimes from smallest to largest:

$$O(2\log n),\ O(100000000000),\ O(n^3+4),\ O(n\log n),\ O(0.0005n),\ O(n!),\ O(2^n),\ O(n^2\log n),\ O(n^n)$$

b. Analyze the runtime of the following code in terms of `N`. Describe the shape of its order of growth (constant, logarithmic, linear, quadratic, etc.).

```
def idunno(N):
    for i in range(N):
        j = 0
        while j < N:
            j = N
```

Runtime: $O($_____$)$

c. Give the worst-case and best-case runtime in terms of `N`. Assume `ping(i, j)` runs in $O(1)$ and returns an `int`.

```
def mystery(N):
    for i in range(N):
        for j in range(N):
            if ping(i, j) > 64:
                break
```

Best Case Runtime: $O($_____$)$

Worst Case Runtime: $O($_____$)$

d. Give the worst-case and best-case runtime in terms of `N`. Assume `ping(i, j, k)` runs in $O(1)$ and returns a `bool`.

```
def mystery(N):
    for i in range(N):
        for j in range(N):
            k = N
            while k > 1:
                if ping(i, j, k):
                    break
                k //= 2
```

Best Case Runtime: $O($_____$)$

Worst Case Runtime: $O($_____$)$

3. Sorting

Suppose we have the following code that creates a list:

```
numbers = [42, 8, 19, 73, 1, 67, 28, 5]
```

Part I: Selection Sort

Suppose we call `selection_sort(numbers)`. Write the contents of the list after each pass of the outermost loop in the Selection Sort algorithm (i.e., after each element has been selected and placed in its final position).

After Pass 1: _____

After Pass 2: _____

After Pass 3: _____

After Pass 4: _____

After Pass 5: _____

After Pass 6: _____

After Pass 7: _____

Part II: Merge Sort

Now suppose we call `merge_sort(numbers)` on the same list instead. Show each group of sublists produced during the splitting stage and each merge step that produces a larger sorted list.

1st Split: _____

2nd Split: _____

3rd Split: _____

1st Merge: _____

2nd Merge: _____

3rd Merge: _____

4. Merge Sort Code

Write the `merge` helper function used in Merge Sort. The function takes two sorted lists, `left` and `right`, and returns a new sorted list containing all of the elements from both lists.

You may assume both lists are sorted in non-decreasing order. Do not modify the original lists.

Examples:

- `merge([1, 3, 5], [2, 4, 6])` returns `[1, 2, 3, 4, 5, 6]`
- `merge([1, 4], [2, 3, 5])` returns `[1, 2, 3, 4, 5]`
- `merge([], [2])` returns `[2]`

```
def merge(left, right):

    result = ______________________

    i = 0
    j = 0

    # Merge while both lists have elements remaining.

    while ________________________________________:

        if ________________________________________:

            ________________________________________

            ________________________________________

        else:

            ________________________________________

            ________________________________________

    result += ______________________

    result += ______________________

    return _________________________
```