Merge Sort Algorithm

Divide the unsorted list into two roughly equal halves.

Recursively sort each half:

- If the half contains more than one element, repeat the divide and sort steps.

Merge the sorted halves back together:

- Compare the elements from both halves one by one and arrange them in a sorted order.
- Continue this until both halves are completely merged into one sorted array.

Repeat the process until the entire list is sorted.

```
MergeSort(arr)
    if length of arr <= 1
        return
    mid = length of arr / 2
    left = arr[0 to mid-1]
    right = arr[mid to end]
    MergeSort(left)
    MergeSort(right)
    Merge(arr, left, right)
Merge(arr, left, right)
    i = 0, j = 0, k = 0
    while i < length of left and j < length of right
        if left[i] < right[j]
            arr[k] = left[i]
            i = i + 1
        else
            arr[k] = right[j]
            j = j + 1
        k = k + 1
    while i < length of left
        arr[k] = left[i]
        i = i + 1
        k = k + 1
    while j < length of right
        arr[k] = right[j]
        j = j + 1
        k = k + 1
```

Quick Sort Algorithm

Pick a Pivot: Choose an element from the array as the pivot (usually the last element or a random one).

Partition the Array:

- Rearrange the elements such that elements smaller than the pivot are on its left.
- Elements greater than the pivot are on its right.
- Place the pivot in its correct sorted position in the array.

Recursively Sort the Subarrays:

- Apply Quick Sort to the subarray of elements on the left of the pivot.
- Apply Quick Sort to the subarray of elements on the right of the pivot.

Repeat the process until each subarray has one or no elements, at which point the entire array is sorted.

```
QuickSort(arr, low, high)
    if low < high
        pivotIndex = Partition(arr, low, high)
        QuickSort(arr, low, pivotIndex - 1)
        QuickSort(arr, pivotIndex + 1, high)

Partition(arr, low, high)
    pivot = arr[high]
    i = low - 1
    for j = low to high - 1
        if arr[j] < pivot
            i = i + 1
            swap(arr[i], arr[j])
    swap(arr[i + 1], arr[high])
    return i + 1

swap(arr, i, j)
    temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
```

Merge Sort Time Complexity

- Best Case: O(nlogn)
- Average Case: O(nlogn)
- Worst Case: O(nlogn)
- Space Complexity: O(n)

Quick Sort Time Complexity

- Best Case: O(nlogn)
- Average Case: O(nlogn)
- Worst Case: O(n^2)
- Space Complexity: O(logn) (for the recursive call stack)