CS3012-2 || Algorithm Analysis

Merge Sort Algorithm Assignment

Student name: Razan Almahdi

Student ID: S20106649

The algorithm (Pseudocode):

```
step 1: start

step 2: declare array and left, right, mid variable

step 3: perform merge function.
    if left > right
        return
    mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)

step 4: Stop
```

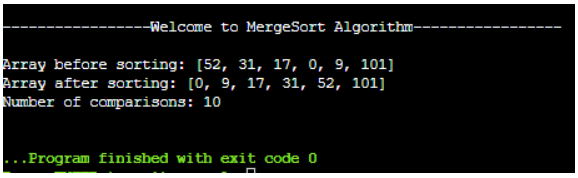
The Algorithm (Python 3):

```python
def merge_sort(arr):
    if len(arr) == 1:
        return arr, 0

    mid = len(arr) // 2
    left, left_count = merge_sort(arr[:mid])
    right, right_count = merge_sort(arr[mid:])

    i, j, k = 0, 0, 0
    count = left_count + right_count

    # store the smallest elements in the array
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            j += 1
        count += 1
        k += 1

    # update the array with the remaining elements
    # remaining elements in the left array
    while i < len(left):
        arr[k] = left[i]
        i += 1
        k += 1

    # remaining elements are in the right array
    while j < len(right):
        arr[k] = right[j]
        j += 1
        k += 1

    return arr, count
```

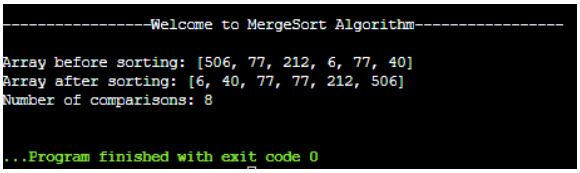
```python
#driver code
print("\n-----------------Welcome to MergeSort Algorithm-----------------\n")
arr = [506, 77, 212, 6, 77, 40]
print(f"Array before sorting: {arr}")
sorted_arr, count = merge_sort(arr)
print(f"Array after sorting: {sorted_arr}")
print(f"Number of comparisons: {count}")
```

Results:

First array:



Second array:



Time Complexity:

The time complexity for merge sort algorithm is **θ (N log(N))** in all cases: (worst, average, and best) as merge-sort constantly divides the array into two halves and takes linear time to repeatedly sort the arrays and merge them using the merge function.

With each recursive call for the merge-sort, the array is divided in half, which requires **O (log N)** time because the array size is reduced by a factor of 2 in each call.

After which, the merge function merges two sorted arrays of size n/2, which takes O(n) time to compare each element in both of the arrays. This operation is performed for both halves of the array; hence the total time complexity of the merge function is **O(n).**

Combining both of the time complexities for both the merge and the sort functions, we find that the merge-sort algorithm has an **O (N log N)** time complexity to perform **O (log N)** splits, and **O(n)** comparisons. This makes the merge sort algorithm an efficient sorting algorithm for large data sets in all cases.