Mert Kilic

100334228

Lab4

CPSC 2150

2)

a)

T(n) = T(n/3) + T(n/3) + T(n/3) + n

T(n) = 3T(n/3) + n

T(n/3) = 3T(n/9) + n

T(n) = 3(3T(n/9) + n) + n

T(n) = 3k T(n/3k) + kn ≈ nlogn

Time Complexity = O(nlogn)

b)

3-way merge sort is better than 2-way merge sort in terms of **time complexity**. This is because, by using 3-way sort, we can break down the list into smaller list faster. For example, if the programmer is sorting a list with size 10000 then, using the 3-way merge sort will break the array into sub arrays faster because it constantly divides the current list to 3 sub-lists. Therefore, using 3-way merge sort is more efficient than 2-way merge sort in terms of time complexity.

c)

Instead of initializing new arrays, we can keep track of the indexes of the sub-parts.

Here is my approach:

inPlaceMerge(int low, int mid, int high, T& array)

For i = low To mid

For j = mid To high

IF(array[i] > array[j])

swap(array[i], array[j])

inPlaceMergeSort(int low, int high, T& array)

IF (high == low)

return high

int mid = (low + high) / 2

inPlaceMergeSort(low, mid, array)

inPlaceMergeSort(mid+1, high, array)

inPlaceMerge(low, mid, high, array)

return 0

My approach increases the time complexity of the merge but lowers the space complexity to O(1).