Merge Sort: LEXISORT - Easy Sorting

How it works:

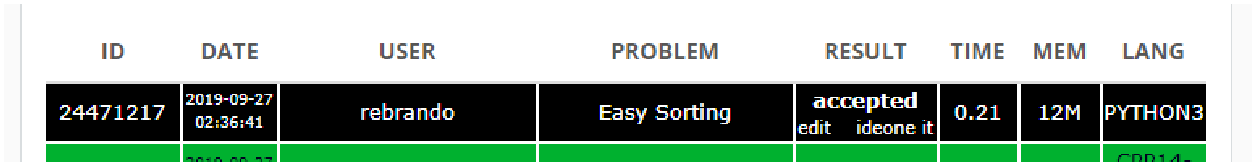
The merge sort solution to the Lexisort works the same as any other merge sort. Merge sort first divides the given array in half. In order to divide the array the program first calculates the middle of the array, then it sets everything to the right of the middle as the left array, and everything to the right of the midpoint as the right array. The next step of the merge sort function is to sort the divided arrays. If the length of the array is already 1 then it is already sorted. If the length of the merge sort is larger than 1, the sorting function continues. First the sort function keeps an index for the left array, right array, and the main (original) array. Looping through the left and right arrays, the function search both left and right arrays comparing each number together, once the function finds the lower value (the one in the lowest lexicographical order) it then inserts the lower value into to the original array at the current index. This sorting function works because the left and right arrays given are already sorted. Finally, any remaining items in the left and right arrays are added back into the original array. This merge sort function comes down to three main subfunctions: divide, sort, and merge.

Time Complexity:

For the merge_sort algorithm, the best-case, worst-case, and average-case scenario is all the same. Since there is only really two loops in the algorithm, the time complexity of the algorithm is only $T(n) = O(n)$. This time complexity is because if there is a change in the length of the array the time scales linearly technically by a factor of $T(n) = 2(n)$ since the highest power is only n^1 , and the coefficient drops we are able to simplify this to $T(n) = O(n)$.

```
def divide(array):
    midpoint = math.floor(len(array)/2); leftSide = array[:midpoint];rightSide = array[midpoint:]return
leftSide,rightSide                                          - O(1)
def merge(mainIndex, copyIndex, mainArray, copyArray):
    while copyIndex < len(copyArray):                       - O(n)
        mainArray[mainIndex] = copyArray[copyIndex]         - O(1)
        mainIndex = mainIndex + 1                           - O(1)
        copyIndex = copyIndex + 1                           - O(1)
    return mainArray, mainIndex                             - O(1)
def sort(left,right,array):
    rightIndex = 0                                          - O(1)
    leftIndex = 0                                           - O(1)
    sortedIndex = 0                                         - O(1)
    while rightIndex < len(right) and leftIndex < len(left): - O(n)
        if left[leftIndex] < right[rightIndex]:             - O(1)
            array[sortedIndex] = left[leftIndex]            - O(1)
            leftIndex = leftIndex + 1                       - O(1)
        else:                                               - O(1)
            array[sortedIndex] = right[rightIndex]          - O(1)
            rightIndex = rightIndex + 1                     - O(1)
        sortedIndex = sortedIndex + 1                       - O(1)
    merge(sortedIndex, leftIndex, array, left)              - O(1)
    merge(sortedIndex, rightIndex, array, right)            - O(1)
    return array                                            - O(1)
def merge_sort(array):
    if len(array) != 1:                                     - O(1)
        left,right = divide(array)                          - O(1)
        merge_sort(left)                                    - O(1)
        merge_sort(right)                                   - O(1)
        sort(left,right,array)                              - O(1)
        return array                                        - O(1)
```

SPOJ Reports:



ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
24471217	2019-09-27 02:36:41	rebrando	Easy Sorting	accepted edit ideone it	0.21	12M	PYTHON3