CS/ECE 374 FALL 2018
Homework 4 Problem 2

Zhe Zhang (zzhan157@illinois.edu)
Ray Ying (xinruiy2@illinois.edu)
Anqi Yao (anqiyao2@illinois.edu)

2. Suppose you are given k sorted arrays $A_1, A_2, \ldots, A_k$ each of which has n numbers. Assume that all numbers in the arrays are distinct. You would like to merge them into single sorted array A of kn elements. Recall that you can merge two sorted arrays of sizes n_1 and n_2 into a sorted array in $O(n_1 + n_2)$ time.

   - Use a divide and conquer strategy to merge the sorted arrays in $O(nk \log k)$ time. To prove the correctness of the algorithm you can assume a routine to merge two sorted arrays.
   - In MergeSort we split the array of size N into two arrays each of size $N/2$, recursively sort them and merge the two sorted arrays. Suppose we instead split the array of size N into k arrays of size N/k each and use the merging algorithm in the preceding step to combine them into a sorted array. Describe the algorithm formally and analyze its running time via a recurrence. You do not need to prove the correctness of the recursive algorithm.

Solution: 1.We could merge the first array A_1 with the second array A_2, the third array A_3 with the fourth array A_4 and so on. Then k sorted arrays will become $k/2$ sorted arrays. We will do this procedure recursively until there is only 1 sorted array containing kn elements.

The algorithm can be represented by the following pseudocode:

```
mergeArrays(A_1, A_2, ..., A_k):

    if(numberof(A_1, A_2, ..., A_k) == 1)
        return A_1
    firstpart = mergeArrays(A_1, A_2, ..., A_{k/2})
    secondpart = mergeArrays(A_{k/2+1}, A_{k/2+2}, ..., A_k)
    result = merge(firstpart, secondpart)
    return result
```

Each recursive step we do $O(kn)$ work to merge the k sorted arrays into $k/2$ sorted arrays. Since we will continue doing this procedure until there is only 1 sorted array, we have to do $O(kn)$ work for $O(logk)$ times. Thus this algorithm takes $O(nklogk)$.

We will prove the correctness of the algorithm by doing induction on the number of sorted arrays, k.

- Base case: when k = 1, the algorithm returns a sorted array which is correct.
- Inductive hypothesis: Suppose our algorithm holds for all the x such that $|x| < |k|$. The algorithm could merge x sorted arrays into a single sorted array containing xn elements.

- Since both firstpart and secondpart in our algorithm contain $k/2$ elements where $|k/2| < |k|$, by the inductive hypothesis, both firstpart and secondpart are sorted arrays containing $kn/2$ elements. Since it is a routine to merge two sorted arrays, merge() will merge firstpart and secondpart into a single sorted array containing kn elements.

Thus our algorithm is correct.

2. The algorithm can be represented by the following pseudocode:

```
arrays[1],arrays[2],...,arrays[k] mergeSort(A[1,2,...,N]):

    if(length(A[1,2,...,N]) == 1)
        return A
    for(i in 1 : (k-1))
        shift = (i-1) * (N/k)
        arrays[i] = mergeSort(A[shift+1,shift+2,...,shift+N/k]
    arrays[k] = mergeSort(A[(k-1)*(N/k)+1,...,N]
    result = mergeArrays(arrays[1],arrays[2],...,arrays[k])
    return result
```

We know from part(1) that merging k sorted arrays which are array[1],array[2],...,array[k], needs O(Nlogk).Then We analyze the running time via a recurrence.

$$T(k) = \begin{cases} O(1) & N = 1 \\ kT(N/k) + O(Nlogk) & otherwise \end{cases}$$

The overall running time is O(NlogN).

∎