COMP 6651: Assignment 1 - partial solution

PROBLEMS.

3 *Proof.* The assumption says there exist two numbers $c_1, c_2 > 0$ such that for sufficiently large n,

$$c_2 n \le f(n) \ln f(n) \le c_1 n$$

Need to show two directions.

- $f(n) = O(n/\ln n)$: By

$$c_2 n \le f(n) \ln f(n) \le f^2(n)$$

  we have $f(n) \ge \sqrt{c_2}\sqrt{n}$. Hence, $\ln f(n) \ge \ln \sqrt{c_2} + \ln \sqrt{n} \ge c \cdot \ln n$ for some constant $c > 0$ and for sufficiently large n. This implies

$$f(n) \le \frac{c_1 n}{\ln f(n)} \le \frac{c_1}{c} \cdot \frac{n}{\ln n}.$$

- $f(n) = \Omega(n/\ln n)$: For sufficiently large n,

$$f(n) \le f(n) \ln f(n) \le c_1 n.$$

  Hence, similarly as before, $\ln f(n) \le c' \ln n$ for some constant $c' > 0$ and for sufficiently large n. Hence,

$$f(n) \ge \frac{c_2 n}{\ln f(n)} \ge \frac{c_2}{c'} \cdot \frac{n}{\ln n}.$$

$\square$

6 **Solution.**

(a) Let $c > 0$ be a constant such that $Merge(A, B)$ takes time $c(|A| + |B|)$. Merging A_1 with A_2 takes time $c(n+n) = 2cn$ and results in an array of size $2n$. Then we apply $Merge$ to this array and A_3 which takes time $c(2n + n) = 3cn$ and results in array of size $3n$. Then we apply $Merge$ to this array and A_4 which takes time $c(3n + n) = 4cn$ and results in array of size $4n$, and so on. You see the pattern. In the last step, we merge an array of size $(k-1)n$ with an array of size n which takes time $c((k-1)n + n) = kcn$. The overall time taken by the algorithm is

$$2cn + 3cn + 4cn + \cdots + kcn = \left(\frac{(k)(k+1)}{2} - 1\right) cn.$$

Observe that the same calculation applies whether c comes from the big-Oh part of the statement that $Merge(A, B)$ takes $O(|A| + |B|)$ time, or if c comes from the big-Omega part of the statement that $Merge(A, B)$ takes $\Omega(|A| + |B|)$ time. Thus, we conclude that this merge procedure takes time $\Theta(k^2 n)$.

procedure $MultipleMerge(\ell, r)$

$\triangleright$ Arrays $A_1, \ldots, A_k$ are global variables and indices $\ell < r$ indicate which subsequence of arrays

$\triangleright$ needs to be merged, i.e., $A_\ell, A_{\ell+1}, \ldots, A_r$.

if $\ell = r$ **then**

  return A_ℓ

else

  $m \leftarrow \lfloor (\ell + r)/2 \rfloor$

  $L \leftarrow MultipleMerge(\ell, m)$

  $R \leftarrow MultipleMerge(m + 1, r)$

  return $Merge(L, R)$

(b) The idea is to merge the first $k/2$ arrays and the last $k/2$ arrays recursively and then merge the two returned lists. The following pseudocode implements this idea:

Let $T(n, k)$ denote the worst-case running time of the above procedure on k lists, each of size n. Then we have $T(n, 1) = O(n)$ and $T(n, k) = 2T(n, k/2) + O(n)$ for $k \geq 2$. One can use the technique of the recursion tree to see that the total amount of work is $O(nk)$ in each level of the recursion and that there are $O(\log k)$ levels in total. Therefore, the running time is $O(nk \log k)$.