


## ENEE651/CMSC751 : TA's feedback on HW2

**Exercise 6:** A few students wrote that the parallel binary search during the partitioning stage takes  $O(\frac{n}{x})$  time instead of  $O(\log n)$ .

**Exercise 8:** The best bounds are obtained by choosing the number of partitions to be  $n/\log n$  and  $m/\log m$  from arrays sized  $n$  and  $m$  respectively. The resulting time complexity remains the same with  $O(\log mn)$ , while the work complexity is  $O(n + m + \frac{n \log m}{\log n} + \frac{m \log n}{\log m})$ .

Some students got better bounds of  $O(n + m)$  work by choosing the number of partitions as  $\frac{m+n}{\log m + \log n}$ . If  $n > m$ , this quantity is not necessarily less than  $m$ , leaving the entire array  $B$  as a single partition. In step 2 of the algorithm, this could lead us to have the entire array  $B$  being ranked serially, thus worsening the asymptotic time bound. 

**Exercise 9:** Many students reported the minimum number of spawn commands that should be used to be equal to 2, while 1 spawn block is sufficient to code both of the steps. Careful study of steps 2(a) and 2(b) of the algorithm tells that for each of the  $2n/\log n$  threads, only the starting index of a partition and its rank with respect to the other array need to be known to initiate the serial ranking algorithm. Hence, the thread which ranks the head of a partition can continue with the serial ranking of step 2, until it reaches an array index which belongs to the next ranking problem.

### Exercise 10:

- A few students used 'merge sort' to sort the sub-arrays after the first iteration, when the text meant to use the same sorting algorithm (i.e., choosing the median of each sub-array always) by "... continue by sorting separately the elements larger than the median and the ones smaller than the median".
- A few students used only algorithm 1 of complete algorithm, whose asymptotic time complexity is  $O(\log^2 n)$ . This gave inferior time bound.
- Quite a few students failed to prove conclusively that the proposed algorithm can indeed be used to sort an array. One should always use an induction (on an invariant or otherwise) to prove why an algorithm works. Here, it could be proved using the following loop invariant:

After round  $i$ ,  $2^i - 1$  elements are placed in the index specified by the completely sorted array.

It could also be proved by using induction on the array size  $n$ :

Prove the algorithm stands true for  $n = 1$ . Assume it holds for  $n = k$ , and prove it holds for  $n = k + 1$ .

**Exercise 11:** A few students wrote that  $cardinality(v) = \sum_{i=1}^{n/r} ps(v, i)$ , while it is in fact equal to  $\sum_{i=1}^{n/r} number(v, i) = ps(v, n/r)$ .