1. Given two sorted sequences A and B of n elements, design an $O(\log \log n)$ time optimal speed up merging algorithm in the PRAM model.

Solution:

1. Divide elements of A and B into $\sqrt{n}$ blocks of size $\sqrt{n}$ each.
2. Let a_i and b_i be the first elements of i^{th} blocks of A and B. In parallel, compare each a_i and b_j. The number of processors required will be $O(n)$.
3. For each a_i, let $crossrank_i$ be the index such that $b_{crossrank_i} < a_i < b_{crossrank_i+1}$.
4. Compare a_i with each element of the block starting with $b_{crossrank_i}$. At the end of this step, we will have the position of each a_i in B. Hence the problem has been reduced to smaller sub-problems of size $\sqrt{n}$.

Analysis: Number of processors required are $O(n)$. Following is the recurrence for time analysis:

$$T(n) = T(\sqrt{n}) + O(1)$$

Therefore, $T(n) = O(\log \log n)$.

2. Consider the following algorithm to sort an $n \times n$ array of numbers. Assume for simplicity that $n = 2^k$.
   - Sort the four $n/2 \times n/2$ sub-arrays recursively according to some indexing scheme.
   - Rotate every alternate rows of the smaller sub-arrays by $n/2$ positions right/left.
   - Run 3 iterations of Shear Sort.

   Prove that the above algorithm correctly sorts and analyze the parallel running time in some appropriate model.

Solution:

Analysis: Let the element of $n \times n$ array to be either 0 or 1. After recursively sorting a sub-array, it may contain at most 1 *dirty* row. Let the *dirty* rows be numbered r_0, r_1, r_2, r_4 for the four sub-arrays. Therefore total number of *dirty* rows in upper half of the matrix will be $|r_0 - r_1| + 1$. Similarly total number of *dirty* rows in lower half of the matrix will be $|r_2 - r_4| + 1$.

Note that except for the dirty rows r_0, r_1, r_2, r_4, all of other dirty rows contain equal number of 0's and 1's. Clean rows are not affected by rotation. Dirty rows between r_0 and r_1 will have alternatively left half with all 0's, right half with all 1's and right half with all 0's, left half with all 1's. Similarly for dirty rows between r_2 and r_3.

First iteration of Shear sort will result in sort these alternating rows and only unsorted rows that may remain will be r_0, r_1, r_2, r_4. Now each iteration of shear sort reduces the number of dirty rows by at least 2. So the remaining two iterations sorts the array.

Step 1 of the algorithm reduces the problem to four recursive sub problems of size $O(n/2)$. Step 2 takes $O(n)$ time for half rotation. Step 3 also requires $O(n)$ time. So the recurrence is

$$T(n) = T(n/2) + O(n)$$

Therefore, parallel running time $T(n)$ will be $O(n)$.

3. Describe an optimal speed up algorithm for list ranking using the idea of *random mate* discussed in class. Choose an appropriate PRAM model to get a $O(\log n)$ time and $n/\log n$ processors algorithm.

Solution:
Algorithm:
For $O(\log_{\frac{4}{3}} \log n)$ iterations do following.

1. Randomly assign the label *male* or $female$, with equal probability, to each element.
2. Drop an element if it is a *male* and its predecessor in the list is a *female*.
3. Update the successor of an element if current successor is dropped due to Step 2.
4. Add to the rank of the element kept, the ranks of its dropped successors.

At the end of Step 3, size of the list is reduced by some constant factor. We can then repeat the above algorithm $O(\log \log n)$ times to get the size of the list down to $n/\log n$ elements. After that we can apply the *pointer jumping* algorithm for list ranking.

Analysis:

The expected size of the new list after first iteration due to Step 2 will be $O(3n/4)$. After $O(\log_{\frac{4}{3}} \log n)$ iterations, the expected size of the list will be $O(\frac{3}{4}^{\log_{\frac{4}{3}} \log n} n)$ which is $O(n/\log n)$. Step 1 and 2 takes $O((3/4)^j \log n)$ time at j^{th} iteration. So total time taken for Step 1 and 2 in all iterations is $O(\log n)$. With $O(n/\log n)$ processors and $O(n/\log n)$ elements, the time taken by *pointer jumping* algorithm is $O(\log n)$. Total time taken will be $O(\log n)$.

4. Analyze the following variation of the parallel connectivity algorithm. Each directed tree is contracted to a star following the hooking step. Instead of the adjacency matrix, use a list or array data structure to implement the algorithm using $O(E + V)$ processors and *polylog* parallel time.

Solution:
Algorithm:

1. Initialize for each node i, $parent_i$ to i.
2. For each node i, we find the neighbor with the largest index say max_i.
3. Set the $parent_i$ to be max_i and remove max_i from neighbor list of i.
4. Run list ranking or contraction procedure on $parent_i$.
5. Modify the label of node i to $parent_i$. Note that this step may lead more than one nodes with the same labels.
6. Repeat Step 2 to 5, $O(\log V)$ times.

Analysis:

For each node and edge in the graph we have a processor. So total number of processors required are $O(V + E)$. Step 2 is finding maximum among $O(V)$ (at most) elements. This can be done in $O(\log V)$ time. Step 3 is done in $O(1)$ time. Step 2 and 3 together constitutes *hooking* step. Step 4 is the list ranking step and takes $O(\log V)$ time using *pointer jumping* algorithm. These steps are repeated $O(\log V)$ time, so total running time is $O((\log V)^2)$.

5. Consider a linear array of n processors, where each processor p_i holds n_i packets. Moreover, $\sum_i n_i = n$ such that each processor is a destination of exactly one packet. Analyze the greedy routing algorithm with *furthest destination first* queue discipline for this problem giving rigorous and complete proofs.

Solution: [1]
For now consider the packets with destination to the right. Since routing in both direction is allowed simultaneously, the proof for the packets with destination to the left is also same. Consider a packet p starting from the i^{th} processor and its destination be $j(> i)$. This packet can be delayed by packets with destination greater than j. There can be at most $n - j$ such packets. Let $k_1, k_2, \cdots k_n$ be such packets in node $1, 2, \cdots n$ respectively, $\sum_{i=1}^{n} k_i \leq n - j$. Let l be a node such that $k_l > 1$ and $k_m \leq 1$ for $l < m \leq n$. The packets $k_{m+1}, k_{m+2} \cdots k_n$ are never delayed. At every step one new packet becomes part of this group. So after $n - j$ steps all the packets that can delay the packet p are in this group. They can now reach their destination step by step without delaying the packet p. Packet p need $j - i$ more steps to reach j. Total steps needed for p to reach its destination are $n - j + j - i = n - i$. So maximum time for a packet to reach its destination is $O(n)$.

References

[1] Randomized algorithms for packet routing on the mesh. `https://pdfs.semanticscholar.org/82d4/8932fad926c7de24d41508f1da14cb25f7ad.pdf`, 1991.