1. (a) Describe briefly but clearly how to efficiently merge three sorted lists, of sizes $m_1 \leq m_2 \leq m_3$, into a single sorted list. Try to minimize the number of comparisons under the assumption that $m_1 \approx m_2 \approx m_3$, but no proof of optimality required.

   (b) Exactly how many comparisons does your algorithm use in the worst case (as a function of m_1, m_2, m_3).

2. Consider a MergeSort-like algorithm in which the array is split into thirds, rather than halves.

   (a) Write pseudo-code for your algorithm. You may assume a three-way merge algorithm is available (as described in Question 1). Your algorithm should work for general n, i.e., even if the size of the array is not a power of 3.

   (b) Use the tree method to analyze exactly how many comparisons your algorithm uses (as we did in class for standard mergesort). You may assume that the size of the array is a power of 3. Assume that the number of comparisons to merge three lists, each of size m, is $6m - 3$.

3. Recall that on Homework 1, we implemented the ladder method without recursion. We will now implement it with recursion. The first version will find the largest value. The second version will find the index of the largest value. In principle, you could just implement the second version and use that index to find the largest value for the first version; you are not supposed to do so. For both versions, you may only use a constant amount of extra memory, other than the extra memory that the recursion automatically uses on the stack (or heap) during execution. You can assume that the array is stored in locations $1, \ldots, n$ or locations $0, \ldots, n-1$, whichever you prefer. Assume that n is a power of 2.

   (a) Write the pseudo-code for a *recursive* implementation of the ladder method of finding the largest of n values.

   (b) Write the pseudo-code for a *recursive* implementation of the ladder method of finding *the index of* the largest of n values.