1. Consider two sorted linked-list collections, a and b, of length N and M respectively, which need to be merged to form a new sorted collection c of size N+M. Implement the code for the method `MergeLists (c, a, b)` where the argument c is an empty, linked-list collection which upon return will contain the new sorted collection.

2. Consider sorting an array collection by splitting the array c[1..n] into two sub-arrays a[1..m] and b[m+1..n], sorting the two sub-arrays separately and then merging the result using the routine `MergeArrays (c, l, m, n)`. By applying a divide-and-conquer strategy to this sorting strategy recursively we define the MergeSort algorithm.
   (a) Implement the code for `MergeSort (c, l, n)`
   (b) What is the order of complexity for this algorithm as a function of n?
   (c) Does your answer in (b) always hold or are there pathological cases?
   (d) Compare MergeSort with QuickSort.

3. Complete the `rb_insert (Tree T, node x)` code to cover both left and right cases? What happens if the ‘uncle’ doesn’t exist? What happens if the ‘grand-parent’ doesn’t exist?

4. Use your code from Qu. 3, or map internally to a 2-3-4 tree abstraction, and show how the red-black tree grows and stays balanced as the following numbers are inserted into an initially empty tree: 1 2 3 4 5 6 7 8