Problems (1), (2), and (3) deal with merging two lists each of size two: \((A < B)\) and \((C < D)\). There are six possible final orderings: \((A < B < C < D)\), \((A < C < B < D)\), \((A < C < D < B)\), \((C < A < B < D)\), \((C < A < D < B)\), and \((C < D < A < B)\).

Problem 1.

(a) Assume your algorithm compares \(A\) and \(C\) first. Give a decision tree for merging.
(b) What is the worst case number of comparisons?
(c) What is the best case number of comparisons?
(d) What is the average case number of comparisons?

Problem 2.

(a) Assume your algorithm compares \(B\) and \(C\) first. Give a decision tree for merging.
(b) What is the worst case number of comparisons?
(c) What is the best case number of comparisons?
(d) What is the average case number of comparisons?

Problem 3. What can you conclude from Problems (1) and (2)?

Problem 4. Assume you want to sort a list of \(n\) numbers that are in \(k\) groups, so that the smallest \(n/k\) are first, then the next smallest \(n/k\) are second, etc.

(a) Give a decision tree based lower bound on the time to produce a single sorted list. (You could base a lower bound on the fact that there really are \(k\) independent sorting problems. Do NOT use this lower bound argument.)
(b) Give an algorithm for sorting this list. How fast is the algorithm? (This is an upper bound.)
(c) Compare your lower and upper bounds.

Problem 5. Assume you have an alphabet of letters from “e” to “l”. Illustrate the operation of radix sort on the following list of English words:

ill, elk, leg, elk, fig, ill, gel, eke, egg, gig, ell