1. Using Figure 1.3 as a model, illustrate the operations of merge sort on the array $A =< 3, 41, 52, 26, 38, 57, 9, 49 >$.

Reducing to smallest (working):

$< 3 >, < 41 >, < 52 >, < 26 >, < 38 >, < 57 >, < 9 >, < 49 >$

(after first merge) (merge pairwise).(only the second element in the resulting collection has had its constituent elements reordered).

$< 3, 41 >, < 26, 52 >, < 38, 57 >, < 9, 49 >$

(after second merge):
$< 3, 26, 41, 52 >, < 9, 38, 49, 57 >$
Working.

2. Write pseudocode for $MERGE(A, p, q, r)$.

Working.

3. Use mathematical induction to show that the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2, \\ 2T(n/2) + n & \text{if } n = 2^k, k > 1 \end{cases}$$

is $T(n) = n \lg n$.

The base case corresponds to $k = 1$, since $n = 2 \rightarrow n = 2^1$.

From the base case we have by definition that $T(2) = 2$. We also have $2 \lg 2 = 2 \cdot 1 = 2 = T(2)$, so we conclude that $T(n) = n \lg n$ for $n = 2$ and the recurrence is satisfied for $k = 1$.

Suppose we know that the recurrence is satisfied for $n = 2^k$, i.e., we know that $T(2^k) = 2^k \lg 2^k$ for $k \geq 1$. From the definition of the recurrence, and from substitution of the expression for $T(2^k)$, we can write:

$$\begin{aligned} T(2^{k+1}) &= 2T(2^{k+1}/2) + 2^{k+1} \\ &= 2T(2^k) + 2^{k+1} \\ &= 2^{k+1} \lg 2^k + 2^{k+1} \\ &= 2^{k+1} \cdot (\lg 2^k + 1) \end{aligned} \tag{1}$$

But $\lg 2^k = k$, so $(\lg 2^k + 1) = k + 1 = \lg 2^{k+1}$ and we can write:

$$T(2^{k+1}) = 2^{k+1} \cdot \lg(2^{k+1}) \tag{2}$$

and the recurrence is satisfied for $n = 2^{k+1}$.

Since the recurrence is satisfied for $n = 2$, and since we know that if the recurrence is true for $n = 2^k$ then it must also be true for $n = 2^{k+1}$, we conclude by induction that the recurrence is true for all $k \geq 1$.

4. Insertion sort can be expressed as a recursive procedure as follows. In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. Write a recurrence for the running time of this recursive version of insertion sort.

Working

5. Referring back to the searching problem (see Exercise 1.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against $v$ and eliminate half of the sequence from further consideration. **Binary search** is an algorithm that repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is $\Theta(\lg n)$.

Working

6. Observe that the **while** loop of lines 5-7 of the INSERTION-SORT procedure in Section 1.1 uses a linear search to scan (backward) through the sorted subarray $A[1..j-1]$. Can we use a binary search (see Exercise 1.3-5) instead to improve the overall worst-case running time of insertion sort to $\Theta(n \lg n)$?

Working

7. Describe a $\Theta(n \lg n)$-time algorithm that, given a set $S$ of $n$ real numbers and another real number $x$, determines whether or not there exist two elements in $S$ whose sum is exactly $x$.

Sort the set $S$ in $\Theta(n \lg n)$-time using a recursive algorithm like merge sort. Select the first element in the sorted set and perform a binary search

on the remaining elements to determine if the pair sum to the desired value. If no match is detected, select the next element in sorted set and again perform a binary search on the remaining elements to see if there is a match, and so on.

Binary search runs in $\Theta(\lg n)$-time and in the worst-case scenario, this binary search needs to be performed $n$ times, which gives an overall running time of $\Theta(n \lg n)$. If the sorting step takes $\Theta(n \lg n)$ and the search step takes $\Theta(n \lg n)$, the overall time performance for the procedure is $T(n) = \Theta(n \lg n)$.

Alternatively, once the set is sorted, the subsequent search can be performed in linear time by scanning the list for matches from the front and back simultaneously. In this case the time performance would still be $T(n) = \Theta(n \lg n + n) = \Theta(n \lg n)$.

Working.