Exercises 5.1

1. a. Write pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.

   b. What will be your algorithm's output for arrays with several elements of the largest value?

   c. Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.

   d. How does this algorithm compare with the brute-force algorithm for this problem?

2. a. Write pseudocode for a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of n numbers.

   b. Set up and solve (for $n = 2^k$) a recurrence relation for the number of key comparisons made by your algorithm.

   c. How does this algorithm compare with the brute-force algorithm for this problem?

3. a. Write pseudocode for a divide-and-conquer algorithm for the exponentiation problem of computing a^n where n is a positive integer.

   b. Set up and solve a recurrence relation for the number of multiplications made by this algorithm.

   c. How does this algorithm compare with the brute-force algorithm for this problem?

4. As mentioned in Chapter 2, logarithm bases are irrelevant in most contexts arising in analyzing an algorithm's efficiency class. Is it true for both assertions of the Master Theorem that include logarithms?

5. Find the order of growth for solutions of the following recurrences.

   a. $T(n) = 4T(n/2) + n,\ \ T(1) = 1$

b. $T(n) = 4T(n/2) + n^2, \;\; T(1) = 1$

c. $T(n) = 4T(n/2) + n^3, \;\; T(1) = 1$

6. Apply mergesort to sort the list *E, X, A, M, P, L, E* in alphabetical order.

7. Is mergesort a stable sorting algorithm?

8. a. Solve the recurrence relation for the number of key comparisons made by mergesort in the worst case. (You may assume that $n = 2^k$.)

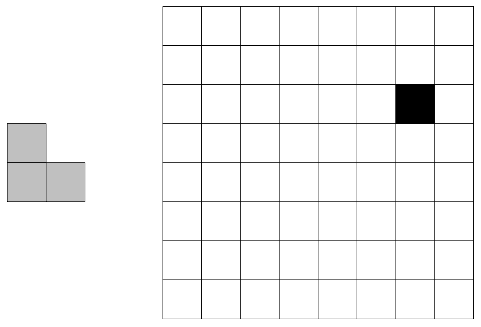
   b. Set up a recurrence relation for the number of key comparisons made by mergesort on best-case inputs and solve it for $n = 2^k$.

   c. Set up a recurrence relation for the number of key moves made by the version of mergesort given in Section 5.1. Does taking the number of key moves into account change the algorithm's efficiency class?

9. Let $A[0..n-1]$ be an array of n real numbers. A pair $(A[i], A[j])$ is said to be an ***inversion*** if these numbers are out of order, i.e., $i < j$ but $A[i] > A[j]$. Design an $O(n \log n)$ algorithm for counting the number of inversions.

10. One can implement mergesort without a recursion by starting with merging adjacent elements of a given array, then merging sorted pairs, and so on. Implement this bottom-up version of mergesort in the language of your choice.

11. *Tromino puzzle* A tromino is an L-shaped tile formed by adjacent 1-by-1 squares. The problem is to cover any 2^n-by-2^n chessboard with one missing square (anywhere on the board) with trominoes. Trominoes should cover all the squares of the board except the missing one with no overlaps.



    Design a divide-and-conquer algorithm for this problem.

Hints to Exercises 5.1

1. In more than one respect, this question is similar to the divide-and-conquer computation of the sum of n numbers.

2. Unlike Problem 1, a divide-and-conquer algorithm for this problem can be more efficient by a constant factor than the brute-force algorithm.

3. How would you compute a^8 by solving two exponentiation problems of size 4? How about a^9?

4. Look at the notations used in the theorem's statement.

5. Apply the Master Theorem.

6. Trace the algorithm as it was done for another input in the section.

7. How can mergesort reverse a relative ordering of two elements?

8. a. Use backward substitutions, as usual.

   b. What inputs minimize the number of key comparisons made by mergesort? How many comparisons are made by mergesort on such inputs during the merging stage?

   c. Do not forget to include key moves made both before the split and during the merging.

9. Modify mergesort to solve the problem.

10. n/a

11. A divide-and-conquer algorithm works by reducing a problem's instance to several smaller instances of the *same* problem.