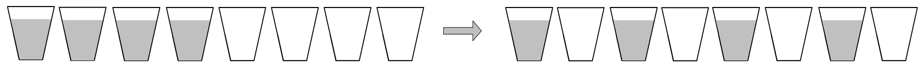
Exercises 4.1

1. *Ferrying soldiers* A detachment of n soldiers must cross a wide and deep river with no bridge in sight. They notice two 12-year-old boys playing in a rowboat by the shore. The boat is so tiny, however, that it can only hold two boys or one soldier. How can the soldiers get across the river and leave the boys in joint possession of the boat? How many times need the boat pass from shore to shore?

2. *Alternating glasses* a. There are $2n$ glasses standing next to each other in a row, the first n of them filled with a soda drink while the remaining n glasses are empty. Make the glasses alternate in a filled-empty-filled-empty pattern in the minimum number of glass moves.

   b. Solve the same problem if $2n$ glasses—n with a drink and n empty—are initially in a random order.

3. *Marking cells* Design an algorithm for the following task. For any even n, mark n cells on an infinite sheet of graph paper so that each marked cell has an odd number of marked neighbors. Two cells are considered neighbors if they are next to each other either horizontally or vertically but not diagonally. The marked cells must form a contiguous region, that is a region in which there is a path between any pair of marked cell that goes through a sequence of marked neighbors. [Kor05]

4. Design a decrease-by-one algorithm for generating the power set of a set of n elements. (The power set of a set S is the set of all the subsets of S, including the empty set and S itself.)

5. Consider the following algorithm to check connectivity of a graph defined by its adjacency matrix.

```
**Algorithm** *Connected*($A[0..n-1, 0..n-1]$)

//Input: Adjacency matrix $A[0..n-1, 0..n-1]$) of an undirected graph $G$
//Output: 1 (true) if $G$ is connected and 0 (false) if it is not
**if** $n = 1$ **return** 1      //one-vertex graph is connected by definition
**else**
    if **not** *Connected*($A[0..n-2, 0..n-2]$) **return** 0
    else for $j \leftarrow 0$ to $n-2$ do
            if $A[n-1, j]$ return 1
        return 0
```

Does this algorithm work correctly for every undirected graph with $n > 0$ vertices? If you answer "yes," indicate the algorithm's efficiency class in the worst case; if you answer "no," explain why.

6. *Team ordering* You have results of a completed round-robin tournament in which n teams played each other once. Each game ended either with a victory of one the teams or with a tie. Design an algorithm that lists the teams in a sequence so that every team did not loose the game with the team listed immediately after it. What is the time efficeincy class of your algorithm?

7. Apply insertion sort to sort the list E, X, A, M, P, L, E in alphabetical order.

8. a. What sentinel should be put before the first element of an array being sorted in order to avoid checking the in-bound condition $j \geq 0$ on each iteration of the inner loop of insertion sort?

   b. Will the version with the sentinel be in the same efficiency class as the original version?

9. Is it possible to implement insertion sort for sorting linked lists? Will it have the same $O(n^2)$ efficiency as the array version?

10. Consider the following version of insertion sort.

```
Algorithm InsertSort2($A[0..n-1]$)
for $i \leftarrow 1$ to $n-1$ do
    $j \leftarrow i-1$
    while $j \geq 0$ and $A[j] > A[j+1]$ do
        swap($A[j]$, $A[j+1]$)
        $j \leftarrow j-1$
```

   What is its time efficiency? How is it compared to that of the version given in the text?

11. Let $A[0..n-1]$ be an array of n sortable elements. (For simplicity, you can assume that all the elements are distinct.) Recall that a pair of its elements $(A[i], A[j])$ is called an ***inversion*** if $i < j$ and $A[i] > A[j]$.

   a. What arrays of size n have the largest number of inversions and what is this number? Answer the same questions for the smallest number of inversions.

b.▶ Show that the average-case number of key comparisons in insertion sort is given by the formula

$$C_{avg}(n) \approx \frac{n^2}{4}.$$

12. Shellsort (more accurately Shell's sort) is an important sorting algorithm which works by applying insertion sort to each of several interleaving sublists of a given list. On each pass through the list, the sublists in question are formed by stepping through the list with an increment h_i taken from some predefined decreasing sequence of step sizes, $h_1 > ... > h_i > ... > 1$, which must end with 1. (The algorithm works for any such sequence, though some sequences are known to yield a better efficiency than others. For example, the sequence 1, 4, 13, 40, 121, ... , used, of course, in reverse, is known to be among the best for this purpose.)

a. Apply shellsort to the list

$$S,\ H,\ E,\ L,\ L,\ S,\ O,\ R,\ T,\ I,\ S,\ U,\ S,\ E,\ F,\ U,\ L$$

b. Is shellsort a stable sorting algorithm?

Hints to Exercises 4.1

1. Solve the problem for $n = 1$.

2. You may consider pouring soda from a filled glass into an empty glass as one move.

3. It's easier to use the bottom-up approach.

4. Use the fact that all the subsets of an n-element set $S = \{a_1, ..., a_n\}$ can be divided into two groups: those that contain a_n and those that do not.

5. The answer is "no."

6. Use the same idea that underlies insertion sort.

7. Trace the algorithm as we did in the text for another input (see Fig. 4.4).

8. a. The sentinel should stop the smallest element from moving beyond the first position in the array.

   b. Repeat the analysis performed in the text for the sentinel version.

9. Recall that we can access elements of a singly linked list only sequentially.

10. Since the only difference between the two versions of the algorithm is in the inner loop's operations, you should estimate the difference in the running times of one repetition of this loop.

11. a. Answering the questions for an array of three elements should lead to the general answers.

    b. Assume for simplicity that all elements are distinct and that inserting $A[i]$ in each of the $i + 1$ possible positions among its predecessors is equally likely. Analyze the sentinel version of the algorithm first.

12. a. Note that it is more convenient to sort sublists in parallel, i.e., compare $A[0]$ with $A[h_i]$, then $A[1]$ with $A[1 + h_i]$, and so on.

    b. Recall that, generally speaking, sorting algorithms that can exchange elements far apart are not stable.