S. Schwoon / D. Suwimonteerabuth Winter Term 2008/09

Fundamental Algorithms

Exercise Sheet 3

1. Consider the following array a:

3	6	2	1	4	5

   Follow each of the sorting algorithms below with the array a, and count the number of comparisons until it is sorted.

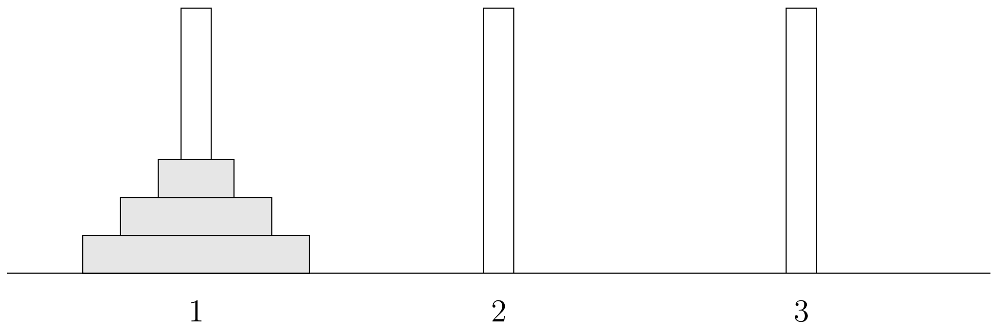
   (a) Insertion sort

   (b) Heapsort

   (c) Quicksort, where pivots are always the left-most elements

   Follow each of the sorting algorithms again, but this time with the sorted array. Which algorithm in your opinion performs best in which case? Justify your answers.

2. The puzzle *towers of Hanoi* consists of three rods, numbered 1, 2, and 3, and a number of disks of different sizes which can be slid onto any rod. Initially, all disks are stacked in rod 1, starting with the largest one on the bottom and successively smaller ones on top. The following figure illustrates the puzzle with 3 disks.



   The objective of the puzzle is to move the entire stack of disks to rod 2, by moving one disk from one rod to another rod at a time and never placing a larger disk on top of a smaller one.

   Assume that there is a procedure move(i, j) that moves the top disk from rod i, and place it on rod j. Find an algorithm that solves the puzzle by calling the procedure move until the disks are moved to rod 2. The algorithm should take a number of disks as a parameter. For example, if there are three disks, the algorithm may call move 7 times in the following order:

   `move(1,2), move(1,3), move(2,3), move(1,2), move(3,1), move(3,2), move(1,2).`

Analyze the complexity of your algorithm.

Hints: devise a more general procedure that takes three parameters: a number of disks n and two rod numbers i and j. The procedure should assume that (i) the disks on rod i and j are stacked in the correct order, (ii) there are at least n disks on rod i, and (iii) the n^{th}-smallest disk on rod i is smaller than the smallest disk on rod j. After a call to the procedure, top n disks from rod i are moved to rod j, in the correct order.

3. Apart from quicksort, there is another well-known sorting algorithm based on the divide-and-conquer paradigm called `mergesort`. Given an array a to be sorted, `mergesort` works as follows:

   > If a is of length 0 or 1, then a is returned because it is already sorted. Otherwise, a is divided into two subarrays, u and v, having lengths about half of the length of a. Then, u and v are recursively sorted by applying `mergesort` again. Once u and v are sorted, they are merged back into one sorted array, and the sorted array is returned.

   (a) Devise the procedure `merge` that takes two sorted arrays and returns an array consisting of the elements from the input arrays in sorted order.

   (b) Using the procedure `merge` above, write down the algorithm `mergesort`. Analyze its complexity.