DTU Informatics
Michael R. Hansen

13/9 12

02157 Functional Programming
Merge Sort
Exercises in connection with Lecture 2

You shall develop a version of merge sort, an interesting sorting algorithm that has an $n \log n$ worst-case execution time. The purpose of this particular exercise is to illustrate the elegance of functional programming – not to develop efficient sorting programs (though they have the "right" asymptotic complexity). We shall later on in the course study techniques addressing efficiency.

Strive for succinctness and elegance when you solve the problem — it is important that your programs and program designs can be communicated to other people.

Merge Sort

Merge sort is an efficient algorithm for sorting a list of elements that has a worst-case execution time of order $n \log n$.

A *merge* of two sorted lists, e.g. merge [1;4;9; 12], [2; 3 4; 5; 10; 13]) is a new sorted list, [1;2;3;4;4;5;9;10;12;13], made up from the elements of the arguments. This operation can be declared so that it has a worst-case running time proportional to the sum of the length of the argument lists. Declare such a function.

Declare a function to *split* a list into two lists of (almost) the same lengths by solving Exercise 4.8. What is the worst-case execution time of your function?.

The idea behind *top-down* merge sort is a recursive algorithm: take an arbitrary list xs with more than one element and split it into two (almost) equal-length lists: xs_1 and xs_2. Sort xs_1 and xs_2 and merge the results. The empty list and lists with one element are the base cases.

Declare a function for top-down merge sort in F# that has a worst-case execution time of order $n \log n$. (Argue about the worst-case running time.)

Bottom-up merge sort

The idea behind *bottom-up* merge sort is explained as follows:

1. Construct a list of one-element lists $[[a_1]; \ldots, [a_j]; [a_{j+1}]; \ldots, [a_n]]$, from the original list $[a_1; \ldots; a_n]$.

2. Traverse the list repeatedly, where each traversal merge neighbouring pairs of lists. For example, after one traversal the list has the form:

$$[\text{merge}([a_1], [a_2]); \text{merge}([a_3], [a_4]); \ldots]$$

   This process will end with a list containing one sorted list.

Declare a function for bottom-up merge sort in F# that has a worst-case execution time of order $n \log n$. (Argue about the worst-case running time.)

Which of the two merge sort programs would you prefer?

On the course homepage there is a program to generate random lists of length n. Use that program in the tests of your sorting programs.