Foundations of computing: Algorithms and Data Structures

Rasmus Pagh

September 12, 2011

1 Theoretical exercises

1. A variant of selection sort (section 2.1) finds the k smallest elements of an array of integers. Describe this variant, and analyze its running time in terms of n and k.

2. Write pseudocode for a recursive procedure that computes numbers of the form c^n, where c is a real number and n is an integer. (Similar computations are important in cryptographic protocols.) Analyze the number of recursive calls made in terms of n. Argue for correctness and time complexity.

   Oyster of the week. Describe an improved procedure that uses $\sim 2 \lg n$ recursive calls, and argue for its correctness.

3. Generalize abstract in-place merge (SW page 271) to four-way merging that takes four sorted lists and produces a single sorted list with all input items. Argue that your algorithm runs in linear time, and that it can be used to halve the number of merging phases of mergesort from $\lceil \log n \rceil$ to $\lceil \log(n)/2 \rceil$. (This gives a version of mergesort that makes better use of the CPU cache, a topic we will come back to later in the course.)

2 Programming exercise

Words can be categorized into *anagrams* that contain the same set of letters, with the same counts. To recognize anagrams, we may use the following:

a) Write a procedure *String sort(String x)* that returns a string containing the letters of the string x in sorted order. The procedure should use linearithmic time in the length n of the string.
Hint. Avoid creating strings of length $1, 2, 3, \ldots, n$ as this would lead to quadratic algorithm. For example, you can consider using Java's `StringBuilder` API.

Two strings x_1 and x_2 are anagrams if and only if *sort* returns the same string on both inputs. Two strings x_1 and x_2 are *almost-anagrams* if they contain the same letters except for one character change or addition. For example, `test` and `taste` are almost-anagrams, and so are `glad` and `lads`.

b) Write a procedure *Boolean almostAnagram(String x1, String x2)* that returns `true` if `x1` and `x2` are almost-anagrams. The procedure should use linearithmic time. Argue to a fellow student that your procedure is correct.