Week 6: Lab

Collaboration level 0 (no restrictions). Open notes.

Topics: asymptotic analysis, comparison-based sorting, sorting lower bound, linear-time sorting, heaps, selection.

1. Recall that in the (smart) SELECT() algorithm described in the notes, the input elements are divided into groups of 5. In this problem we'll look at what happens if the input is divided into groups of 7 element instead.

   (a) As before, the algorithm finds a "good" pivot before calling Partition: In this case, for each group of 7 elements it computes its median, and then finds the median of these medians. Denote it by x. Using the same argument as in the notes, find out how many elements in the input are guaranteed to be $< x$; and how many elements are guaranteed to be $> x$, respectively.

   (b) Write the recurrence corresponding to this version of the Select() algorithm.

   (c) Does this solve to $O(n)$ time?

   (d) Based on this, does dividing the input into groups of 7 elements lead to a linear time SELECT() algorithm?

   Note: Similarly, it can be shown that groups of size > 5 lead to a linear time algorithm, and groups of size < 5 do not lead to a linear algorithm.

2. Let A be a list of n (not necessarily distinct) integers. Describe an $O(n)$-algorithm to test whether any item occurs more than $\lceil n/2 \rceil$ times in A.

   (a) You may assume that the integers are in a small range, $K = O(n)$.

   (b) Come up with a general solution, without making any additional assumptions about the integers (in particular you may not assume that the range is small). Hint: use Select()

   We expect: pseudocode, why is it correct and analysis

3. Given an unsorted sequence S of n elements, and an integer k, we want to find the $k-1$ elements that have rank $\lceil n/k \rceil$, $2\lceil n/k \rceil$, $3\lceil n/k \rceil$, and so on, up to $(k-1)\lceil n/k \rceil$.

   (a) Describe the "naive" algorithm that works by repeated selection, and analyze its running time function of n and k (do not assume k to be a constant).

   (b) Describe an improved algorithm that runs in $O(n \lg k)$ time. You may assume that k is a power of 2. After you describe it, argue why its running time is $O(n \lg k)$.

   We expect: pseudocode, why it's correct and analysis