Assignment 7

CSCI 3110: Design and Analysis of Algorithms

Due July 10, 2018

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

Banner ID: _____ Name: _____

Given a set S of n items, a *heavy hitter* is an item that occurs often in S, according to some appropriate notion of "often". Here, we consider an item to occur often in S if it occurs at least k times, for some parameter k given as part of the algorithm's input. So, a *heavy hitter* of S is an element that occurs at least k times in S. Your task is to develop an algorithm that finds all the heavy hitters in S in $O(n \lg(n/k))$ time. (In the interest of simplifying the analysis, let us define $\lg x := \max(1, \log_2 x)$ here.) Describe your algorithm, prove that it is correct, and show that its running time is indeed $O(n \lg(n/k))$.

Hint: Sorting can get you close: After sorting the input, you scan it and output all elements that you see at least k times in a row in the sorted sequence. This takes $O(n \lg n)$ time, which is $O(n \lg(n/k))$ as long as $k \leq n^{1-\varepsilon}$, for any $\varepsilon > 0$. For large values of k, however, the algorithm takes too long.

Sorting gets you close in a different sense as well. Take Quick Sort as a starting point. Assume the current invocation partitions around a pivot p. Can you afford to check whether p is a heavy hitter? How do you find all other heavy hitters? When can you stop recursing because you can be certain that the current input contains no heavy hitters? Stopping the recursion early enough is the key to reducing the running time from $O(n \lg n)$ to $O(n \lg(n/k))$.