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: In general 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. 5 is the smallest size which leads 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

More practice

1. For each algorithm listed below, give a recurrence that describes its worst-case running time, and give its worst-case running time in Θ-notation. You do not need to show your work, only the recurrence and its solution.

   - binary search

   - merge sort

2. Let A be an array of n elements. Recall that the partition algorithm used by Quicksort runs in $O(n)$ time and partitions the array into two sub-arrays A_1 and A_2 such that all elements in A_1 are smaller (or equal) than all elements in A_2.

   Now consider a partition of A into 3 arrays A_1, A_2, A_3 such that the elements in array A_1 are smaller (or equal) than the elements in array A_2, which are all smaller (or equal) than the elemenst in A_3; furthermore, we'll assume that the partition is so that A_1, A_2 and A_3 have equal size. We call this a 3-partition.

   (a) Let $A = 9, 8, 4, 6, 5, 1, 2, 7, 3$. Show one possible 3-partition of this array. Note that it is not specified how to compute a 3-partition, so you only need to show a possible 3-partition, that is, one partition that satisfies the definition.

   (b) Describe a generalization of Quicksort that uses a 3-partition. Assume that you are given a black-box to compute a 3-partition (you do not need to describe how the 3-partition works, only how the sorting works). For e.g. you could assume that the 3-partition returns two indices say i, j so that all elements from $0..i$ are smaller (or equal) than the elements in $i + 1..j$, which are smaller (or equal) than the elements in $j + 1..r$.

   ```
   //sort a[p..r] using a 3-partition
   quicksort( array a, int p, int r)
   ```

   (c) Give a recurrence for the running time; in your recurrence you can assume that computing a 3-partition on an array of size n runs in $O(PARTITION(n))$.