Problem Set 4

Data Structures and Algorithms, Fall 2020

Due: October 15, in class.

Problem 1

Suppose that you are given a sequence of n elements to sort. The input sequence consists of n/k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n/k subsequences.

(a) Bob claims an $\Omega(n \lg k)$ lower bound on the number of comparisons needed to solve this variant of the sorting problem. The basis idea of his proof is: "any comparison based sorting algorithm needs $\Omega(k \lg k)$ comparisons to sort k elements; since we need to sort n/k subsequences each containing k elements, we need $(n/k) \cdot \Omega(k \lg k) = \Omega(n \lg k)$ comparisons." Unfortunately, this proof is invalid. Can you see why?

(b) Interestingly enough, indeed $\Omega(n \lg k)$ comparisons are needed to solve this variant of the sorting problem. Can you provide a valid proof for this claim?

Problem 2

(a) Develop an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ in $O(1)$ time. Here, $a \in \mathbb{N}$ and $b \in \mathbb{N}$. Your algorithm should use $O(n + k)$ time for preprocessing.

(b) You are given an array of integers, where different integers may have different numbers of digits, but the total number of digits over all the integers is n. Develop an algorithm to sort the array in $O(n)$ time.

Problem 3

Suppose that, instead of sorting an array, we just require that the elements increase on average. More precisely, we call an n-element array A *k-sorted* if, for all $1 \leq i \leq n - k$, the following holds:

$$\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k}$$

(a) Prove that an n-element array is k-sorted if and only if $A[i] \leq A[i + k]$ for all $1 \leq i \leq n - k$.

(b) Give an algorithm that k-sorts an n-element array in $O(n \lg (n/k))$ time.

(c) Give an algorithm that sorts a k-sorted array of length n in $O(n \lg k)$ time.

(d) Prove that when k is a constant, k-sorting an n-element array requires $\Omega(n \lg n)$ time.

Problem 4

Assume n is a power of 2. Develop an algorithm that finds the second smallest of n elements using at most $n + \lg n - 2$ comparisons. Argue your algorithm indeed uses at most $n + \lg n - 2$ comparisons.

Problem 5

(a) In the algorithm QUICKSELECT we introduced in class, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? What about groups of 3? You need to justify your answer.

(b) Assume n and k are both some power of 2. The k^{th} quantiles of an n-element set are the $k-1$ order statistics that divide the sorted set into k equal-sized sets. Develop an $O(n \lg k)$ time algorithm to list the k^{th} quantiles of a set. That is, for every $1 \leq i \leq k-1$, find the $(in/k)^{\text{th}}$ order statistic of the set.

Problem 6

For n distinct elements $x_1, x_2, \cdots, x_n$ with positive weights $w_1, w_2, \cdots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the *weighted median* is the element x_k satisfying $\sum_{x_i<x_k} w_i < 1/2$ and $\sum_{x_i>x_k} w_i \leq 1/2$.

For example, if the elements are 0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2 and each element equals its weight (that is, $w_i = x_i$ for all i), then the median is 0.1, but the weighted median is 0.2.

Develop an algorithm that computes the weighted median in $O(n)$ worst-case time.

Problem 7

The pre/in/post-order numbering of a binary tree labels the nodes of a binary tree with the integers $0, \cdots, n-1$ in the order that they are encountered by a pre/in/post-order traversal. (See Figure 1.)

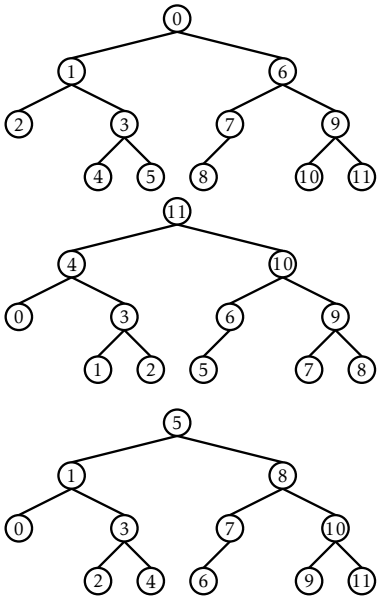

Figure 1: Pre-order, post-order, and in-order numberings of a binary tree.

Suppose you are given a set of nodes with pre-order and in-order numbers assigned. Prove there is at most one binary tree with this pre/in-order numbering and develop an algorithm to construct that tree.

*Problem 8 [Bonus Problem]

(a) Suppose we are given a full binary tree with pre-, post-, and in-order numbers assigned to the nodes. (That is, for each node, there is a data field storing the pre-, post-, and in-order numbers of this node. Also, recall that a binary tree is full if each node has either zero or two children.) Given a node u, show how these numbers can be used to determine the size of the subtree rooted at u, in $O(1)$ time.

(b) Show that the shape of any binary tree on n nodes can be represented using at most $2n+1$ bits.