

## Homework 3

February 21, 2006

Due Date: March 2

I recommend that you wait until after Thursday's lecture to work on problems 2, 3, and 4. If you complete problem 1 by Thursday's lecture, you are in good shape :-)

1. (50 points) In this problem you must use the **decision tree lower bound technique** (even if you could prove a better bound with a different technique) for proving lower bounds for the following four *problems*. Be sure to very clearly explain anything in your proof that is specific to the problem including a careful argument for the lower bound for the number of leaves that must be in the decision tree.

- (a) (10 points) You are given an array `votes` of size 4 with each element holding either a 1 or a 2 (which indicates the candidate for which a vote was cast). The goal is to determine if there is a candidate who received 3 or more votes. The output should either be the empty set if no candidate received at least 3 votes, or otherwise a set holding the indices in `votes` for a candidate with 3 or more votes.

Consider a model of computation in which the algorithm can only access `votes` by asking if `v[i] == v[j]` for any  $i, j$ . You are to give the best lower bound you can on the number of comparisons.

- (b) (10 points) The problem of computing which elements in an *UNSORTED* array  $A$  are less than a given element  $x$  under the comparison-based model of computation.
- (c) (15 points) The problem of computing which elements in an *SORTED* array  $A$  are less than a given element  $x$  under the comparison-based model of computation.
- (d) (15 points) You are given  $n$  coins identical in appearance; either all are genuine or exactly one is fake. It is unknown whether the fake coin is heavier or lighter than the genuine ones. Your model of computation to solve this problem is as follows. You can only learn about the coins through a provided `weigh(set1, set2)` method that takes as input two sets of coins `set1` and `set2` and returns one of the three possibilities: the two sets of coins have the same weight, `set1` is heavier, or that `set2` is heavier.

The problem is to determine if any coin is fake, and if so which coin is fake and whether it is heavier or lighter than the genuine ones.

2. (5 points) Consider the following list of 3-letter initials

CWA  
HWA  
JKK  $\implies$   
HCA  
CWJ  
ACA

Illustrate the execution of radix sort on the above list of initials by showing the list after each of the three phases of counting sort has completed.

3. (10 pts) Suppose you are given the task to sort 10000 numbers between 0 and  $10^{12} - 1$  (i.e. the keys are 12 digit numbers). You have decided to use radix sort but need to decide how many digits to group for each radix sort digit. Which is best among having 1 digit per radix sort digit, 3 digits per radix sort digit, 6 digits per radix sort digit, or directly using counting sort (i.e. 12 digits per radix sort digit)? You are provided with a counting sort procedure with exact time complexity of  $13n + 9k + 4$ . Show your work.
4. (30 pts) Let  $L_1, \dots, L_r$  be  $r$  unsorted lists, whose elements hold integers in the range  $[0, k - 1]$ . Let  $n$  be the total number of elements among all of the lists. That is, if  $n_i$  is the number of elements in  $L_i$ , then  $n = n_1 + n_2 + \dots + n_r$ . Describe an algorithm with **total worst-case** asymptotic time complexity of  $O(r + k + n)$  for getting all of the  $r$  lists into sorted order. So your final output will be  $r$  sorted lists (containing  $L_1, L_2, \dots, L_r$  in sorted order). Be sure to analyze the time complexity of your algorithm.

If you cannot think of an algorithm with the specified time complexity, then for partial credit describe the most efficient algorithm you can and correctly analyze its time complexity.

### Challenge Problems:

Only work on this after you have completed the required problems. Note that you can obtain 100% without doing this.

1. (3 pts) As discussed in class, the expected time complexity of randomized quicksort can be improved in practice by taking advantage of the fast running time of insertion sort when the input is “nearly” sorted. When quicksort is called on a subarray with fewer than  $k$  elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertions sort on the entire array to finish the sorting process. Analyze the expected time complexity of this variant of quicksort as a function of  $n$  and  $k$ .

*Hint: You may compute the savings in the quicksort phase by determining the expected cost of the savings make use of the expected time complexity computed in class for randomized quicksort.*

2. (2 points) Prove that any comparison-based algorithm for constructing a binary search tree from an arbitrary list of  $n$  elements takes  $\Omega(n \log n)$  time in the worst case.

*Hint: Think about reducing the problem of sorting to performing a set of operations on a binary search tree. That is, show that if a faster algorithm existed for constructing a binary search tree then you would violate the  $\Omega(n \log n)$  comparison-based sorting lower bound. You may want to review an in-order traversal of a binary search tree.*