

Homework 3

October 11, 2005

Due Date: October 18

This homework will be a bit shorter than the first two homeworks so that you also have time to review homeworks 1 and 2 in preparation for the exam on October 20th. Thus this homework is worth 75 versus 100 points. You should be able to determine if you made any major errors by looking over the solutions handed out on October 18th. If you have a different solution than that given out that you want someone to look at to give you feedback, let me know (by Wednesday morning). I'll look your solution over and give you feedback. You can pick up your graded homeworks from the CSE office on Thursday morning (Oct 20th) if you want to have a chance to look over the graded homework before the exam.

1. (40 points) In this problem you are to use the **decision tree lower bound 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.

- (a) Prove the best lower bound you can (using the decision tree technique) on the time complexity of the problem of computing which elements in an *UNSORTED* array A are less than a given element x under a comparison-based model of computation where you can only access A by asking if $A[i] < A[j]$ or $A[i] < x$, for i and j integers from 0 to $n - 1$.
- (b) Prove the best lower bound you can (using the decision tree technique) on the time complexity of the problem of computing which elements in an *SORTED* array A are less than a given element x under the same model of computation as in part (a).
- (c) Give the best lower bound you can (using the decision tree lower bound technique) for the number of comparisons used in the worst-case by any comparison-based algorithm to solve the disjointness problem as defined in HW 2.
- (d) 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.

Using the decision tree lower bound technique give the best lower bound you can on the number of times `weigh` must be called in the worst-case 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. (10 pts) Suppose you are given the task to sort 500,000 numbers between 0 and $10^8 - 1$ (i.e. the keys are 8 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, 2 digits per radix sort digit, 4 digits per radix sort digit, or directly using counting sort (i.e. 8 digits per radix sort digit)? You are provided with a counting sort procedure with exact time complexity of $13n + 9k + 4$. Show your work.

3. (25 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 Problem:

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

1. (3 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.