Problem Set 2—Due Wed., Oct. 25 5PM

(10) Problem 1.

Suppose we want to find the first prime larger than n, a randomly chosen 160 bit number. We argued that this will take about $ln(2^{160})/2$ (about 56) calls to the Miller-Rabin primality test (on page 841). The value of s chosen is the maximum number of tests we do on a given number and also determines how confident we are in our result. Compare the expected running time of the Miller-Rabin algorithm using $s = 20$ and $s = 50$ in this setting. To simplify analysis, you may assume that exactly 56 calls to Miller-Rabin are used before the algorithm terminates, and you only need count the number of calls to the Witness routine (so you can ignore gains due to early termination in line 7).

(20) Problem 2. Suppose we are given two sorted lists $A_1 < A_2 < \ldots < A_n$ and $B_1 < B_2 < \ldots < B_n$. Prove that any comparison based algorithm for merging the two lists requires $2n - 1$ comparisons in the worst case. (Hint: use an adversary argument to show that if an algorithm doesn't compare A_1 to B_1 and A_i to both B_{i-1} and B_i for $i = 2, 3, ..., n$ then we can find two inputs with the same comparison results, but different final sorted lists).

(20) Problem 3. Problem 4-7 page 75.

(20) Problem 4.

We consider some variations of the knapsack problem.

(a) Suppose we could take multiple copies of an item (thus for an item with value 3 and weight 5 we could take four of this item getting value 12 at a cost of weight 20). Describe how to solve the knapsack problem under this setting (include both how to find the best total value and the best multi-set of items). Give the time and space used.

(b) The knapsack problem can be solved in $O(Wn)$ time and $O(W)$ space. Suppose you knew a bound V, such that $V \geq V^*$ the optimal value of the final solution. If all values are integers, describe how to solve the knapsack problem in $O(nV)$ time.

(c) Describe an efficient method to find a bound V such that $2V^* \geq V \geq V^*$ (your method should be much faster than solving the knapsack problem).

(20) Problem 5.

Consider the following scheduling problem: you are given n jobs to be scheduled on a single processor. Each job i has two parameters associated with it: a processing time p_i and a deadline d_i. A set of jobs is *feasible* if every job can be completed by its deadline.

For example if we have jobs with $p_i = i$ for $i = 1, 2, 3, 4$ and $d_1=4$, $d_2 = 2, d_3 = 6$ and $d_4=11$, the set is feasible since we can schedule jobs in the order 2,1,3,4 and complete each job by its deadline. However if we changed the deadline of job 3 so that $d_3=5$, then no schedule completes all jobs by their deadline.

(a) Prove that the following strategy schedules any feasible set of jobs to complete before each of their deadlines. Schedule the jobs in increasing order of their deadlines. (Hint: Show that starting with a feasible schedule which violates this strategy, you can rearrange the jobs so that they are in increasing order by their deadlines).

(b) Suppose you are given an infeasible set of jobs. Construct a fast algorithm which finds a maximum cardinality subset of jobs which is feasible. (Hint: this can be solved in O($nlogn$) time using a varriation of the algorithm in a).

(c) Suppose that each job also has a weight w_i associated with it. Given an infeasible set of jobs we want to find a subset of jobs of maximum weight which is feasible (the weight of a set of jobs is the sum of the weights of the jobs in the set). Construct a fast algorithm which finds a feasible subset of maximum weight. Hint: If W is the sum of all the weights you can use dynamic programming to solve the problem in O(Wn) time after sorting the jobs by their deadlines.

(15) Problem 6. Problem 16-4, page 326.