

## Problem Set 1—Due Wednesday, January 14, 3:15PM

**Homework Info** Homeworks are due by 3:15PM on the due date. They are to be turned in at the marked box in Kemper Hall, 2131. No late homeworks will be accepted.

Much of what one learns in this course comes from trying to solve the homework problems, so work hard on them. Doing a conscientious job on the homeworks is the best preparation for the exams. We hope that you will ultimately solve the majority of the problems, but don't be surprised if some of them stump you; some of the problems may be quite challenging.

Your solutions should be terse, correct, and legible. Understandability of the solution is as necessary as correctness. Expect to lose points if you provide a "correct" solution with a not-so-good writeup. As with an English paper, you can't expect to turn in a first draft: it takes refinement to describe something well. Typeset solutions are always appreciated. If you can't solve a problem, briefly indicate what you've tried and where the difficulty lies. Don't try to pull one over on us.

If you think a problem was misgraded, please see the TA (our grader) first.

**(20) Problem 1.** Suppose you had a choice of four algorithms to solve a problem, with the following approximate running times as a function of input size  $n$ :

- Algorithm 1 takes  $2^n$  seconds
- Algorithm 2 takes  $32n^2$  seconds
- Algorithm 3 takes  $128n \lg n$  seconds
- Algorithm 4 takes  $4096n$  seconds

Give the range of  $n$  for which each algorithm is optimal. You need only consider values of  $n > 0$ . (note  $\lg n$  is log base 2 of  $n$ ). For example, when  $n = 2$ , Alg. 1 takes 4 seconds, Alg. 2 takes 128 seconds, Alg. 3 takes 256 seconds, and Alg. 4 takes 8,192 seconds, so Alg. 1 is best here. You also can restrict your answer to positive integers for  $n$ .

**(30) Problem 2. COIN CHANGING VARIATIONS**

- (a)** The programs for computing the minimum number of coins return only the *number* of total coins used, not the actual change to be made (e.g. for  $n = 85$  cents it returns 4, not 3 quarters and 1 dime). Describe how to modify the memoized version of the program so it computes the actual change for the minimum number of coins for this setting. Ideally your modification should not have a major effect on the running time (but you may use some extra space to store information to return the more detailed answer).
- (b)** If we have quarters, dimes, nickles and pennies we claimed that the greedy algorithm (use quarters till below 25 cents) then dimes, then nickles, finally pennies uses the fewest coins possible.

We consider an optimal solution with  $q$  quarters,  $d$  dimes,  $n$  nickles and  $p$  pennies for  $m$  cents change (so  $25q + 10d + 5n + p = m$ ). Show the following facts are true which will show that this algorithm works.

- i)  $p < 5$
- ii)  $n < 2$
- iii)  $d < 3$
- iv) if  $n = 1$  then  $d < 2$
- v) Suppose  $m = 25k + r$  where  $k, r$  are nonnegative integers and  $r < 25$ . Then there is an optimal solution using  $k$  quarters (this shows the first step of the greedy algorithm is optimal, since it uses  $k$  quarters). Hint: show that if  $q < k$  then the solution can be improved, or converted to an equally good one, by replacing some of the existing coins with quarters.

- (c) Use the result from part (b) to describe a constant-time algorithm that, given an input  $m$ , returns  $q, d, n$  and  $p$  the number of each coin to give change with the fewest coins. You may assume (somewhat unrealistically) that you can do standard arithmetic operations on any size numbers in  $O(1)$  time. Note: you may use the result of part (b) even if you didn't solve part (b).

**(25) Problem 3.** Given an unordered list of  $n$  distinct numbers  $x_1, x_2, \dots, x_n$ , and an integer  $B$ , find a maximum cardinality subset of the list which sums to at most  $B$ . (Thus given a list 35, 27, 22, 55, 87, 74 and  $B=100$ , a maximum cardinality set has three elements, e.g. 35, 22, 27; since any four elements sums to more than 100).

- (a) Give a simple  $O(n \log n)$  algorithm to solve this problem.
- (b) Describe how to use the select algorithm to solve the problem in  $O(n)$  time. Justify your time bound. Note you may use the select algorithm as a black box routine which given any list, and a value  $k$  returns the  $k$ th smallest element in linear time. (Hint: start by finding the median of the list, and testing if all the numbers smaller than the median sum to  $B$  or less).

**(25) Problem 4.**

In our discussion of the Select algorithm we assumed there were no duplicate items in our input list. We now consider the more general case where our input list of numbers,  $x_1, x_2, \dots, x_n$  may have multiple copies of the same item. Note that even with duplicates, our notion of the  $k$ th smallest element remains the same: the value of the  $k$ th element of a sorted list: e.g. for 1, 3, 3, 3, 3, 4, 4, 6, 6, 7, return 3 if  $k = 2, 3, 4$ , or 5, and return 4 if  $k = 6, 7$ .

- (a) The algorithm described in class (and in the text) will work correctly even if there are many duplicates, however it may be much slower. Analyze the running time of our algorithm to find the median element when the list consists of  $n$  copies of the same item. Obviously which element you choose as the split element doesn't matter in this case, so the best case, average case, and worst case are all the same.
- (b) Modify the randomized-select algorithm so it has good expected performance even when there may be many duplicates (HINT: modify the partition routine along with the method of deciding which sublist to continue searching).