# Final

December 14, 2009, 11:00–1:30 PM

There are six problems each worth five points for a total of 30 points. Show all your work, partial credit will be awarded. Space is provided on the test for your work; if you use a blue book for additional workspace, sign it and return it with the test. No notes, no collaboration.

Name: _____

| Problem | Credit |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| Total | |

1. Given a set of $n$ numbers $\{ a_1, a_2, \ldots, a_n \}$, find indices $i, j$ and $k$ such that $a_i + a_j = a_k$, or report that there are no such indices.

   Find the algorithm and give it's run time. Full credit given for an algorithm with the better run time.

2. Dynamic programing picking numbers game. (Thanks to Brian Dean)

   There is a row of coins with values $v_1, v_2, \ldots, v_n$, with $n$ even. You play a game against an opponent. You can pick either the leftmost coin $v_1$ or the rightmost coin $v_n$. The opponent can then pick the leftmost coin or the rightmost coin from those remaining. Example: if on your first move you took $v_1$, then the opponent can take either $v_2$ or $v_n$; if on your first move you took $v_n$ instead of $v_1$, then the opponent's response can be either $v_1$ or $v_{n-1}$. The came continues with players alternating. The winner is the player whose chosen coins totals to the highest value.

   Find a dynamic programming algorithm to determine what your highest guaranteed win of coins can be. Remember, you are the player the goes first.

   Analyze the run time of your algorithm.

   *Hint:* What are the self-similar sub-problems? After every two moves, you have some coins that you have taken and are faced with a row of coins $v_i, \ldots, v_j$. You might thing of this corresponding to a table entry $V(i, j)$ which gives the maximum guaranteed winnings for that situation.

   Working you way up, the smallest case would be $v_i, v_{i+1}$. Which coin would you choose of the two and which would you leave to your opponent? For $V(i, i+1)$ what is the number.

   *Note about maximum guaranteed winnings:* Between your move and the opponent's response there are four possibilities (think that through). Consider what will maximize for you given that the opponent might reply in order to minimize between the two remaining options.

This page for your work.

3. Consider the following graph: the nodes V are labeled $0, 1, \ldots, 9$. There is an edge from node $i$ to node $2i \pmod{10}$ for $i = 0, 1, 2, \ldots, 9$; and there is an edge from node $i$ to node $3i \pmod{10}$ for $i = 0, 1, 2, \ldots, 9$.

   Depth first search the graph. Give the $d$ and $f$ numbers. Label the egdes as tree, back, forward and cross.

4. This problem refers back to the previous problem.

   Run the strongly-connected components algorithm on the graph of the previous problem. You can take up where you left off in the problem.

   Show the second depth first search and indicate the strongly connected components.

5. Given two $n$ digit numbers, you can add them in time $O(n)$. This is just the familiar addition in which you work from lowest order digit to highest, using an $O(1)$ algorithm to add two one-digit numbers (you use a look-up table of memorized sums) and adding a possible carry. You update the carry and move to the next highest digit.

   Multiplication is $O(n^2)$ by the familiar technique. Each of the $n$ digits in the multiplier is multiplied against the multiplicand, and each of these takes $O(n)$, as it is a right to left scan of the multiplicand using a look-up table digit-wise, along with a possible carry. So in $O(n^2)$ time a bunch of $O(n)$ partial products results. The remaining summing down to a single result is a sequence of $n-1$ additions of $2n$ digit numbers.

   Since mutliplication is $O(n^2)$ then maybe divide and conquere will help make mutliplicaiton faster. It can, but it is a bit tricky, because you can break the problem into parts, but the parts have to intermingle when you put the parts back togehter.

   The idea is this. Split the number $a$ into the higher order digits $a_h$ and the low order digits $a_l$. The same with $b$. That is,

$$\begin{aligned} a &= a_h B^m + a_l \\ b &= b_h B^m + b_l. \end{aligned}$$

   where $B$ is the base, and $m = n/2$. Then,

$$\begin{aligned} a * b &= (a_h B^m + a_l) * (b_h B^m + b_l) \\ &= a_h * b_h B^{2m} + (a_h * b_l + a_l * b_h) B^m + a_l * b_l \end{aligned}$$

   We now have the one $n$ digit multiplication written as 4 $n/2$ digit multiplications, never mind the additions.

   If we were to recurse, how many multiplications are needed to complete the $n$ digit multiplciaton? Write down the recurrence and solve. Or (for partial credit) show me using a tree.

This page for your work.

6. Karatsuba multiplication: Referring back to the previous problem: Is this the best we can do?

   Notice:

   $$(a_h + a_l) * (b_h + b_l) = a_h * b_h + a_h * b_l + a_l * b_h + a_l * b_l$$

   Find 3 $n/2$ digit multiplications which can be used to create (by adding and subtracting among the three) the equivalent of the 4 products that were required by the previous problem's approach.

   Consider recursing on the halves. Again write down a recurence and solve. Is this faster?

   Can we do faster?