COMP 271 Design and Analysis of Algorithms
(Real) Fall 2009 Final Exam

1. **Quick-Answer Questions** ($3 \times 4 = 12$ pts)

   For each question below, write down the asymptotic (using Θ) result. You do not need to justify your answers.

   1.1 What is the solution of the recurrence $T(n) = T(n/1.1) + n, T(1) = 1$?

   1.2 What is $\sum_{i=1}^{n} \frac{1}{i \cdot \log n}$?

   1.3 Suppose we have an alphabet of m characters $A = \{a_1, \ldots, a_m\}$. We are given a text in which a_1 occurs n times while each of $a_2, \ldots, a_m$ appears only once. Assume $n > m$. What is the length (in terms of bits) of the message when it is encoded using the Huffman code? Express your result using both m and n.

   1.4 What are the *expected* and *worst-case* running times of Randomized Quicksort for sorting n numbers?

2. **[OMITTED FROM SYLLABUS]**
   Multiple Choice ($2 \times 4 = 8$ pts)

   For each of the following statements, indicate whether it is (a) true, (b) false, or (c) unknown based on our current scientific knowledge. *For any statements that you mark "false", please give a brief explanation why it is wrong.* You do not need to justify the "true" and "unknown" answers.

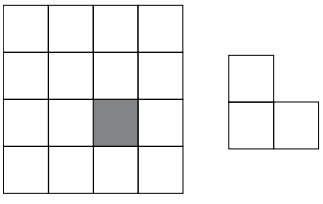
   1.1 $\mathcal{P} \subseteq \mathcal{NP}$.

   1.2 $\mathcal{P} = \mathcal{NP}$.

   1.3 If we can solve any $\mathcal{NP}$-complete problem in polynomial time, we can solve all the $\mathcal{NP}$-hard problems in polynomial time.

   1.4 If it is not possible to solve SAT in polynomial time, then it is not possible to solve DVC (decision version of vertex cover) in polynomial time.

3. (15 pts) In a square with $\sqrt{n} \times \sqrt{n} = n$ cells, where $\sqrt{n}$ is a power of 2, one of the cells has been removed creating a "hole" (see an example below). Now we have an unlimited number of "L"-shaped bricks that can cover three cells (see the figure below). Although it is not that obvious, it is always possible to cover the square with these bricks (without covering the hole). In fact, in this question you are asked to design an algorithm that finds such a covering. Note that in a valid covering, every cell (except the hole) is covered by exactly one brick, and no bricks can go outside the square. For full credits, your algorithm should run in $O(n)$ time. [Hint: Use divide-and-conquer.]



4. (20 pts) In the Partition problem, we are given a set A of positive integers $a_1, \ldots, a_n$ such that their sum $S = \sum_{i=1}^{n} a_i$ is even, and ask if there is a subset B of A such that $\sum_{a_i \in B} a_i = \sum_{a_i \in A-B} a_i = S/2$.

   (a) (15 pts) Design a dynamic programming algorithm that solves the Partition problem. Your algorithm only needs to output "yes" or "no". For full credits your algorithm should run in time $O(nS)$. You have to describe your algorithm from scratch, i.e., you cannot use other learned algorithms as "black boxes".

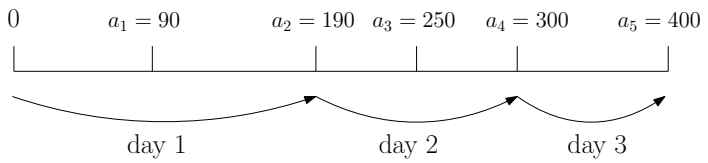
   (b) **[OMITTED FROM SYLLABUS]**
   (5 pts) In the last tutorial you proved that Partition is a $\mathcal{NPC}$ problem. Having solved part (a), some student concludes that he can solve Partition in polynomial time, and thus $\mathcal{P} = \mathcal{NP}$. Shall we give him the Turing award for his ground-breaking result? If not, what's wrong with his argument?

5. (15 pts) You are given an array A of n digits, i.e., $1 \leq A[i] \leq 9$ for each i. Now we want to insert m "+" into A ($m < n$), to make it into a summation. For example, if $A = [7, 9, 8, 4, 6]$ and $m = 2$, then $7 + 98 + 46$ and $79 + 8 + 46$ are two possible valid summations. Your task is to design an algorithm to find the summation with the minimum sum. Your algorithm only needs to output the value of the optimal summation. The output of the above example is 133, since $79 + 8 + 46$ is actually optimal.

   (a) (12 pts) For simplicity, you are allowed to use the function $atoi(i, j), 1 \leq i \leq n$, which converts $A[i...j]$ into an integer. For instance, $atoi(2, 4) = 984$ in the above example. Assume that any call to $atoi$ takes constant time. Your algorithm then should run in $O(n^2 m)$ time.

   (b) (3 pts) Assuming that $atoi$ runs in constant time is not realistic: it actually takes time $O(j - i + 1)$ to "assemble" these $j - i + 1$ digits. If so, can you still make your algorithm run in $O(n^2 m)$ time? [Hint: Don't spend too much time on this question if it appears difficult—it's just worth 3 pts!]

6. (15 pts) (This question sounds similar to the one in the fake exam, but they are NOT the same!) You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \ldots < a_n$, where a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n) which is your destination. If you can travel at most 200 miles between sleeping at two hotels, how many hotels you need to stop at in order to make the trip? Design an algorithm to find the minimum number of hotels you need to stop at. For example, 3 is the answer in the figure below. For full credits, your algorithm should run in $O(n)$ time. You have to prove the optimality of your algorithm unless it is obvious. [Hint: Use a greedy algorithm.]



7. **[OMITTED FROM SYLLABUS]**
   (15 pts) A *dominating set* for a directed graph $G = (V, E)$ is a subset $D \subseteq V$ such

that every vertex $u \in V - D$ is pointed to from at least one vertex in D by some edge. For example, the shaded vertices in the graph below form a dominating set of size 3. The decision dominating set problem (DDS) is the decision problem where we are given a directed graph G and an integer k, and the goal is to decide whether G contains a dominating set of size k. Prove that DDS $\in \mathcal{NPC}$ by reducing from *Set Cover* problem. Recall (from Homework 4) that the Set Cover problem is defined as follows: Given a finite set X and a collection of sets $F = \{S_1, \ldots, S_n\}$ whose elements are chosen from X, and given an integer k, determine if there exist k sets from F such that their union covers all elements in X. [Hint: Construct one vertex for each element in X and one vertex for each set S_i in F, and build appropriate edges among them.]