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

1. 1.1 $\Theta(n)$

   1.2 $\Theta(1)$

   1.3 $\Theta(n + m \log m)$

   1.4 expected: $\Theta(n \log n)$, worst-case: $\Theta(n^2)$

2. **[OMITTED FROM SYLLABUS]**

   2.1 True

   2.2 Unknown

   2.3 False. Some NP-hard problems are more difficult than NP-complete problems.

   2.4 True

3. Divide the $\sqrt{n} \times \sqrt{n}$ cells into 4 parts of size $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$. We firstly cover the 2×2 square in the center with an 'L' bricks and leave one cell uncovered if the corresponding $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$ part already has a hole. Then each of the four $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$ parts has a hole, and can be solved recursively.

   We stop the divide-and-conquer when we have a square of size 2×2 with a hole in it. It can be covered with an 'L' brick trivially.

   The running time $T(n)$ has the following recursion:

   $$T(n) = 4T(\frac{n}{4}) + O(1)$$

   so $T(n) = \Theta(n)$.

4. (a) This problem is equivalent to the subset sum problem covered in the tutorial.
   The solution is to construct a Boolean array $A[i, j], 0 \le i \le n$ and $0 \le j \le \frac{S}{2}$, defined as follows: $A[i, j] =$ true if there is a subset B from the first i numbers that sum to j, else $A[i, j] =$ false.

   $$A[i,j] = \begin{cases} \text{true} & \text{if } 0 \le i \le n \text{ and } j = 0 \\ \text{false} & \text{if } i = 0 \text{ and } 1 \le j \le \frac{S}{2} \\ A[i-1, j] & \text{if } i > 0 \text{ and } a_i > j \\ A[i-1, j-a_i] \text{ or } A[i-1, j] & \text{if } i > 0 \text{ and } j \ge a_i \end{cases}$$

   Computing from bottom to up, we traverse i from 0 to n and j from 0 to $\frac{S}{2}$. Therefore, the complexity is $O(nS)$.

   (b) **[OMITTED FROM SYLLABUS]**
   No. The input size is $O(n \log S)$, instead of $O(nS)$. The running time is exponential in the input size.

5. (a) Define $f(i, m)$ to be the minimum sum produced from the first i digits with m '+'. Then the recursive function is as

   $$f(i, m) = \min_{m \le k \le i-1} \{f(k, m-1) + \text{atoi}(k+1, i)\}$$

where we use index k to enumerate all the positions where to insert the last '+'. There are $O(nm)$ entries for the table $f(i, m)$. For each entry, we compute the 'min', which is chosen from at most n choices. So the total complexity of bottom-up computation is $O(n^2 m)$.

(b) We can improve the recursive function in (a) to compute $\text{atoi}(k+1, i)$ in constant time. We traverse the index k from $i-1$ to m in a decreasing order. Then we can compute $\text{atoi}(k+1, i)$ by $A[k+1] \times 10^{i-k-1} + \text{atoi}(k, i)$.

6. Algorithm: We use greedy algorithm to solve this problem. Every day, we go as much as possible if we can reach a hotel to stop in 200 miles.

   Optimality proof: Suppose an optimal solution stops at hotels at $s_1^*, s_2^*, \ldots, s_k^*$ and the greedy solution stops at $s_1, s_2, \ldots, s_l$. We find the first j where $s_j^* \neq s_j$. Then we have $s_j > s_j^*$ by the greedy nature of the greedy algorithm. Then we can move s_j^* forward to s_j. We notice that s_j^* won't overpass s_{j+1}^*, otherwise the optimal solution can be improved, contradicting that it is optimal. After this step the optimal solution has the same number of hotels and is more similar to the greedy solution. Iteratively applying this step we can convert the optimal solution to the greedy one while keeping the total number of stops.

7. **[OMITTED FROM SYLLABUS]**

   (a) $DDS \in NP$

   The certificate C is a set of k vertices that dominates all other vertices. Checking that every other vertex is dominated by at least one vertex in C can be easily done in polynomial time.

   (b) $DSC \leq_p DDS$

   Given an input to a Decision Set Cover Problem: $X = \{x_1, x_2, \cdots, x_m\}$, and a collection of sets $F = \{S_1, \cdots, S_n\}$, where $S_i \subseteq X$, we construct an input (G, k') to the DDS problem so that there exists a collection of k sets in F that cover X iff G has a dominating set of size k'.

   Transformation: We create a vertex v_i for each x_i, and create a vertex u_i for each S_i. For each $S_i \in F$, and for each $x_j \in S_i$, create an edge (u_i, v_j). For every two vertices u_i, u_j, create two edges (u_i, u_j) and (u_j, u_i). Set $k' = k$. Obviously the transformation could be done in polynomial time.

   Proof: Suppose the DSC input is a yes-input, i.e., there exists a collection of k sets in F that cover X. Then we can pick the corresponding vertices u_i in G. This will dominate all other vertices. Thus DDS is also "yes".

   Suppose G has a dominating set D of size k. If D contains any vertex v_i, because it has no outgoing edges and has at least one incoming edge, we can replace it with any vertex that points to v_i without leaving any vertex undominated. So we can transform D into D' with the same number of vertices. Then we pick the corresponding sets, which will cover the entire X.