Name:
Score:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>[100]</td>
</tr>
</tbody>
</table>

CRN 21723  
CIS 621  
Winter 2010

**Final Exam**  
(due by noon on Thursday, March 18)

This is the usual *open-everything, but no outside help take-home* test. Check "Class News", where I will post “frequently asked questions” about the test. Make sure that your answers are neat (preferably one problem per page) and legible. Please include this cover page with your submission.

1. **Loop Invariant**  
Consider the following algorithm that sorts a set $S$ of $n$ elements using a Priority Queue (PQ) $Q$:

$$
Q := S; \ i := 1;
\text{while (not } Q.\text{emptyPQ)} \ 
\text{do begin ouput}[i] := Q.\text{Min}; \ Q.\text{DeleteMin}; \ i++ \ \text{end}
$$

(i) Prove correctness of the algorithm by providing an invariant, a termination function and verifying the “check list”.

(a) the initialization establishes $Inv$;
(b) $Inv$ is maintained by a single execution of the body of the loop;
(c) upon exit from the loop, $output$ holds the sorted $S$.

(ii) Discuss the complexity (both worst-case and amortized) of the algorithm when PQ is implemented

(a) as a binary heap
(b) as a Binomial Heap (forest of binomial trees with unique ranks)
(c) as a Lazy Binomial Heap (forest of binomial trees)
(d) efficiently for $S \subseteq \{1..n\}$
2. Amortized Complexity  A mergeable priority queue of $n$ elements can be implemented by lazy binomial heaps with constant time $\text{Merge}$ and logarithmic amortized time $\text{DeleteMin}$ operations.

(i) Argue that infrequent $m$ $\text{DeleteMin}$ operations ($m \in o(n)$) can lead to faster performance of the priority queue than what is implied by the lower bound complexity of the comparison-based sorting problem.

(ii) What should $m$ be for a constant time amortized complexity of this implementation?

(iii) [Extra point] Marking an element deleted in constant time implements a “lazy” general delete in a priority queue. Show how this can improve the amortized performance of the data structure when $\text{DeleteMin}$ operations are infrequent.

3. Greedy Algorithms  To merge two ordered files of sizes $p$ and $q$, we move all their $p + q$ elements to a new, ordered file. Assume that a set of $n$ ordered files, $m_1, m_2, \ldots, m_n$ is given together with their sizes $s_1, s_2, \ldots, s_n$. The problem is how to create their sorted union through $n - 1$ merges so as to minimize the number of moved elements. Give a linear ($O(n)$ time) algorithm to optimally schedule the merges in the case when the files are given in order of their sizes, $s_1 \leq s_2 \leq \ldots \leq s_n$. (Note that this is a secondary problem: we are concerned with the order of merges, not with merges themselves.)

4. Greedy Algorithms vs. DP

(i) Both the prefix-free code problem (“Huffman’s code”) and the OBST problem (with data in the leaves) strive to minimize the weighted external path length in a tree. Why does a greedy algorithm work for the former but not for the latter?

(ii) D.E. Knuth observed that the optimal root $r(1, n)$ of an OBST that includes keys $1, \ldots, n$ lies between the optimal roots of the OBST for keys $1, \ldots, n - 1$ and the OBST for keys $2, \ldots, n$: $r(1, n - 1) \leq r(1, n) \leq r(2, n)$. What consequence, if any, does this observation have for the complexity of the DP construction algorithm?
5. **Dynamic Programming** Consider an $n$-gon on the plane (each of the $n$ vertices is given by a pair of coordinates) that is convex (the straight line joining any two interior points does not intersect its sides). A triangulation of the $n$-gon includes $n - 3$ diagonals that divide its interior into $n - 2$ triangular regions; the weight of a triangle is the total length of its sides.

(i) [A warm-up] Solve (ii) below when the weight of a triangle is defined as its area.

(ii) Design an efficient algorithm finding a triangulation of the minimum total weight.

6. **Polynomial-time reductions**

(i) Since the SCC (StronglyConnectedComponents) problem is efficiently solvable, so is the 2-SAT problem which reduces to SCC.

(a) Reduce the following instance of 2-SAT to an instance of SCC and decide its answer: $(\bar{x}_1 \lor \bar{x}_3) \land (x_2 \lor x_3) \land (\bar{x}_1 \lor x_3) \land (x_1 \lor x_3) \land (\bar{x}_4 \lor x_3) \land (\bar{x}_1 \lor x_4) \land (\bar{x}_2 \lor \bar{x}_3) \land (x_1 \lor \bar{x}_4) \land (x_1 \lor \bar{x}_2) \land (\bar{x}_4 \lor \bar{x}_1) \land (\bar{x}_2 \lor x_4)$

(b) Assuming an efficient reduction function, $f(B)$, and an efficient SCC solver, $A(G)$, give an algorithm for constructing a satisfying truth assignment if one exists.

(ii) Prove that the relation $\leq_p$ is transitive and reflexive. Is it symmetric?

(iii) Is a pseudopolynomial-time reduction (polynomial as a function of the maximum value of a number in a problem instance) sufficient to prove $\mathcal{NP}$-hardness?

(iv) Is a polynomial-time non-deterministic reduction sufficient?

7. **NP-completeness** Assume that there is a polynomial time algorithm CLQ to solve the MaximumClique decision problem:

**Instance:** graph $G$ and integer $K$

**Question:** Does $G$ have a completely connected set of $K$ vertices?

(i) Show how to use CLQ to determine the maximum clique size of a given graph in polynomial time.

(ii) Show how to use CLQ to find a maximum clique of a given graph in polynomial time.

(iii) Show that CLQ is NP-complete.