CS 373U: Combinatorial Algorithms, Spring 2004
Homework 2

Due Friday, February 20, 2004 at noon
(so you have the whole weekend to study for the midterm)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- Starting with this homework, we are changing the way we want you to submit solutions. For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to moreeasily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
- Unless specifically stated otherwise, you can use the fact that the following problems are NP-hard to prove that other problems are NP-hard: Circuit-SAT, 3SAT, Vertex Cover, Maximum Clique, Maximum Independent Set, Hamiltonian Path, Hamiltonian Cycle, k-Colorability for any $k \geq 3$, Traveling Salesman Path, Travelling Salesman Cycle, Subset Sum, Partition, 3Partition, Hitting Set, Minimum Steiner Tree, Minesweeper, Tetris, or any other NP-hard problem described in the lecture notes.
- This homework is a little harder than the last one. You might want to start early.

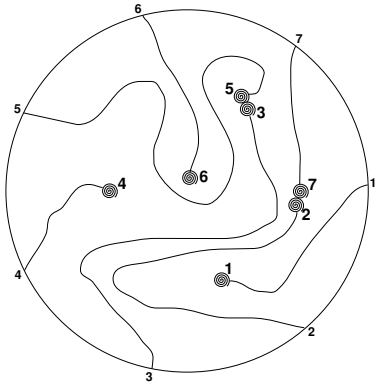
#	1	2	3	4	5	6*	Total
Score							
Grader							

1. In lecture on February 5, Jeff presented the following algorithm to compute the length of the longest increasing subsequence of an n-element array $A[1..n]$ in $O(n^2)$ time.

```
LengthOfLIS(A[1..n]):
  A[n+1] = ∞
  for i ← 1 to n+1
      L[i] ← 1
      for j ← 1 to i−1
          if A[j] < A[i] and 1 + L[j] < L[i]
              L[i] ← 1 + L[j]
  return L[n+1] − 1
```

   Describe another algorithm for this problem that runs in $O(n \log n)$ time. *[Hint: Use a data structure to replace the inner loop with something faster.]*

2. Every year, as part of its annual meeting, the Antarctican Snail Lovers of Union Glacier hold a Round Table Mating Race. A large number of high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to n. The snails wander around the table, each snail leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail (even their own). When any two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if n is even and the race goes on forever.



The end of an Antarctican SLUG race. Snails 1, 4, and 6 never find a mate.
The organizers must pay $M[3,5] + M[2,7]$.

   For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i,j] = M[j,i]$ is the reward if snails i and j meet.

   Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the $n \times n$ array M as input.

3. Describe and analyze a *polynomial-time* algorithm to determine whether a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is satisfiable.

4. This problem asks you to prove that four different variants of the minimum spanning tree problem are NP-hard. In each case, the input is a connected undirected graph G with weighted edges. Each problem considers a certain subset of the possible spanning trees of G, and asks you to compute the spanning tree with minimum total weight in that subset.

   (a) Prove that finding the minimum-weight *depth first search* tree is NP-hard. (To remind yourself what depth first search is, and why it computes a spanning tree, see Jeff's introductory notes on graphs or Chapter 22 in CLRS.)

   (b) Suppose a subset S of the nodes in the input graph are marked. Prove that it is NP-hard to compute the minimum spanning tree whose leaves are all in S. *[Hint: First consider the case $|S| = 2$.]*

   (c) Prove that for any integer $\ell \geq 2$, it is NP-hard to compute the minimum spanning tree with exactly ℓ leaves. *[Hint: First consider the case $\ell = 2$.]*

   (d) Prove that for any integer $d \geq 2$, it is NP-hard to compute the minimum spanning tree with maximum degree d. *[Hint: First consider the case $d = 2$. By now this should start to look familiar.]*

   You're welcome to use reductions among these four problems. For example, even if you can't solve part (d), if you can prove that (d) implies (b), you will get full credit for (b). Just don't argue circularly.

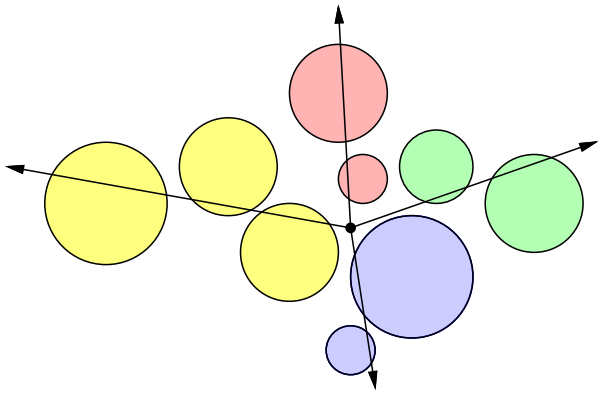
5. Consider a machine with a row of n processors numbered 1 through n. A *job* is some computational task that occupies a contiguous set of processors for some amount of time. Each processor can work on only one job at a time. Each job is represented by a pair $J_i = (n_i, t_i)$, where n_i is the number of processors required and t_i is the amount of processing time required to perform the job. A *schedule* for a set of jobs $\{J_1, \ldots, J_m\}$ assigns each job J_i to some set of n_i contiguous processors for an interval of t_i seconds, so that no processor works on more than one job at any time. The *make-span* of a schedule is the time from the start to the finish of all jobs.

   The *parallel scheduling problem* asks, given a set of jobs as input, to compute a schedule for those jobs with the smallest possible make-span.

   (a) Prove that the parallel scheduling problem is NP-hard.

   (b) Give an algorithm that computes a 3-approximation of the minimum make-span of a set of jobs in $O(m \log m)$ time. That is, if the minimum make-span is M, your algorithm should compute a schedule with make-span at most $3M$. You can assume that n is a power of 2.

*6. **[Extra credit]** Suppose you are standing in a field surrounded by several large balloons. You want to use your brand new Acme Brand Zap-O-Matic™ to pop all the balloons, without moving from your current location. The Zap-O-Matic™ shoots a high-powered laser beam, which pops all the balloons it hits. Since each shot requires enough energy to power a small country for a year, you want to fire as few shots as possible.



Nine balloons popped by 4 shots of the Zap-O-Matic™

The *minimum zap* problem can be stated more formally as follows. Given a set C of n circles in the plane, each specified by its radius and the (x, y) coordinates of its center, compute the minimum number of rays from the origin that intersect every circle in C. Your goal is to find an efficient algorithm for this problem.

(a) Describe and analyze a greedy algorithm whose output is within 1 of optimal. That is, if m is the minimum number of rays required to hit every circle in the input, then your greedy algorithm must output either m or $m + 1$. (Of course, you must prove this fact.)

(b) Describe an algorithm that solves the minimum zap problem in $O(n^2)$ time.

*(c) Describe an algorithm that solves the minimum zap problem in $O(n \log n)$ time.

Assume you have a subroutine INTERSECTS(r, c) that determines, in $O(1)$ time, whether a ray r intersects a circle c. It's not that hard to write this subroutine, but it's not the interesting part of the problem.