Advanced Algorithms 2011. Exam Questions

This take-home exam consists of a number of summarizing test questions on different levels.

Important: Motivate all your answers. An answer, even a correct one, without motivation does not count. (It could be just a lucky guess.)
On the other hand, each of the questions can be answered very briefly, in a few lines. Fully worked-out details are not expected. Write precisely to the point, without digressions.

Submission: Mail your answers to `ptr@chalmers.se` as plain text or PDF attachment strictly before the given deadline. Do not wait until the last minute! You may revise your submission arbitrarily often until the deadline, and only the last version is considered. Asking questions or revising your solutions does not count negatively. If you have any problems to stick to the deadline for an important reason, inform us in good time, in order to avoid failure.

Help: You must do the exam completely on your own. Neither group work nor external help is permitted. Used literature beyond the course material must be cited. Questions may be directed to the teachers only. Utmost academic honesty is expected. Cheating can lead to failure on the entire course and further consequences.

1. The hand-in exercise 1 addressed an algorithm for Load Balancing with an approximation ratio that gets arbitrarily close to 1. Is this a fully polynomial-time approximation scheme? Why, or why not?

2. Given is a set of points in the plane. Each point is colored with one of k colors. We wish to select k points, one of each color, such that the perimeter of the convex hull of the selected points is as small as possible. (The perimeter is the length of a belt laid tightly around the point set.) We propose the following algorithm: Fix some point p in the given set. For each color c, choose a point of color c that is closest to p. Do this (from scratch) for every point p, and eventually take the solution with minimum perimeter. We claim that this algorithm guarantees an approximation ratio at most $\pi = 3.14159\ldots$ Why is this true?

3. Consider the following algorithm for the unweighted Vertex Cover problem: Initially $S := \emptyset$ and $M := \emptyset$. Take an edge e that is disjoint to all edges that are already in M, and add e to M. Add the two end nodes of e to S. Repeat this step as long as possible.
Show that the final S is a vertex cover.

4. Show that the resulting S (in question 3) is at most twice as large as a minimum vertex cover.

5. Obviously, the edge set M (in question 3) is a matching. Is M always a matching of maximum size?

6. In the Weighted Hitting Set problem we are given a set A of n elements, each with a weight w_i, and a collection of m subsets $B_j \subset A$. A hitting set is a subset of A that intersects every B_j. We wish to determine a hitting set with minimum total weight. Show that a solution with a weight at most $H(m)$ times the optimum can be computed in polynomial time, through a simple reduction to Set Cover. It is enough to say how the reduction works.

7. The max-cut problem in graphs asks to find a partition of the node set in two parts such that as many as possible edges exist between these parts. "The max-cut problem can be reduced in polynomial time to min-cut, simply by replacing every edge of the given graph with a non-edge and vice versa. This turns the maximization into a minimization problem." True or not?

8. Bulb fiction: You want to test n light bulbs. In order to save time, you group them into k sets of m bulbs ($n = km$) and test each group in series connection: If some bulb in the group is defective, you get no light. Only if all bulbs in the group work properly, light is on. Assume that the bulbs are defective independently, each with the same probability p. What can you expect to observe?

9. Back to the max-cut problem: Show that the node set of every graph with m edges can be partitioned in two parts such that at least $m/2$ edges exist between these parts. More specifically, give a randomized algorithm that yields a cut with an expected number of $m/2$ edges.

10. Does your randomized algorithm for max-cut (in question 9) find a cut with at least $m/2$ edges in expected polynomial time?

11. Given a directed graph with m directed edges, we want to arrange the nodes in a linear order, such that as many as possible of the directed edges go from left to right. More formally, we want to name the nodes by $v_1, \ldots, v_n$ such that $i < j$ holds for as many as possible of the directed edges $v_i v_j$. Show that there exists a solution where at least $m/2$ edges get the desired orientation. (You may be tempted to use the probabilistic method again, however, this time there is also a trivial way ...)

12. Nice to know that an algorithm like Quicksort runs in $O(n \log n)$ expected time. It would be even nicer to know that large deviations from this expected time are unlikely. Can we use the powerful tool of Chernoff bounds to prove that? Why, or why not?

13. The 3-coloring problem asks to assign 3 colors to the nodes of a given graph, such that adjacent nodes always get different colors (or figure out that no such coloring is possible). The problem can be trivially solved in $O(3^n)$ time. With a little more thinking one can solve it in $O(2^n)$ time. How?

14. How much better is $O(2^n)$ time compared to $O(3^n)$ time? More precisely asked: Suppose you have a fixed time budget, where the $O(3^n)$ algorithm can solve instances with, say, N nodes. How large are the instances that the $O(2^n)$ algorithm can manage? (For simplicity you may assume that the time complexities are exactly 2^n and 3^n, neglecting polynomial factors.)

15. Let a graph G and an integer parameter k be given. Let c be the size of a minimum vertex cover in G. Devise an algorithm with the following properties:

- If $c \leq k$, it outputs a minimum vertex cover in $O^*(1.47^k)$ time.
- If $k < c \leq 2k$, it reports after $O^*(1.47^k)$ time that no vertex cover of size k exists.
- If $2k < c$, it reports already after polynomial time that no vertex cover of size k exists.

(Note carefully that c is not known in the beginning, hence you cannot simply use c as another input parameter.)

16. Consider any problem that is NP-complete but FPT, such as Vertex Cover (to be specific). You are offered a polynomial-time algorithm that takes input (G, k), where G is a graph with more than k nodes, and outputs a graph G' with the following properties: G' is *strictly* smaller than G, and some vertex cover of size k in G (if existing) is also a vertex cover of G'. (Note the resemblance to kernelization.) Would you buy it? That is, would you believe that this algorithm works?