Advanced Algorithms 2014. Exam Problems

Important: Motivate all your answers. An answer, even a correct one, without motivation does not count.
On the other hand, answer concisely and to the point, without digressions. Avoid wordy essays. Extensive writing does not necessarily add to clarity, but it can make it harder to retrieve the relevant statements and logical steps. As a rule of thumb: all sample solutions together will fit in less than 2 printed pages!

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. Feel free to ask questions and to revise your solutions. 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. (However, no solution hints will be given. Only questions about the interpretation of the exam problems will be answered.) Utmost academic honesty is expected. Cheating can lead to failure on the entire course and further consequences.

Problem 1.
In the Load Balancing problem with m machines and n jobs having processing times $t_1, \ldots, t_n$, preemptions were not allowed: Every job must be completed on one machine. Even for $m = 2$ machines it is NP-complete to decide whether we can get the best possible makespan $\sum_{j=1}^{n} t_j/2$.
Now we consider Load Balancing with Preemptions: There it is possible to interrupt a job and move it to another machine. We can even do portions of the same job simultaneously on several machines. Now we can always trivially achieve an optimal makespan $T^* = \sum_{j=1}^{n} t_j/m$. However, we want more: a makespan T^* and a minimum number p of interruptions.
Formulated as a decision problem: "Given $t_1, \ldots, t_n$, m and p, can the jobs be scheduled such that the makespan is $\sum_{j=1}^{n} t_j/m$, and at most p interruptions are used?"
Is this problem solvable in polynomial time or NP-complete?

Problem 2.
The unweighted Set Cover problem asks to cover a set U completely, by selecting a minimum number of sets from a given family of sets $S_i \subset U$, $i = 1, \ldots, m$. Now we "reverse" the problem: An integer k is also given, and the problem is to select exactly k sets that cover as many elements as possible.
Case $k = 1$ is trivial: The largest set, of size $n := \max |S_i|$, is also the best. Now let us consider $k = 2$. Cleerly, we can find the best pair of sets in $O(nm^2)$ time. But maybe m is already very large. We could do two steps of the well-known greedy Set Cover algorithm instead, which requires only $O(nm)$ time:
Let X be the largest S_i. Let Y be a set S_j with maximum $|S_j \setminus X|$. Output X and Y.
Your task: Prove that $X \cup Y$ has at least 3/4 as many elements as an optimal pair of sets would have. If you cannot manage the 3/4 ratio, which needs some detailed work, you may prove a weaker approximation ratio. (Here, seeing the way how you approach such analysis problems is more important than a perfect result.)

Problem 3.
We are given a huge database of sets $S_1, \ldots, S_m$, all of the same size n, and we get a new such set S. We are supposed to find the "most similar" sets in the database, that is, some sets S_i with maximum intersection sizes $|S_i \cap S|$. But due to the huge size, already a trivial $O(mn)$ time scan would be too slow. Therefore we only pick a small set R of random elements from S and check in which sets S_i they occur. If $R \cap S_i \neq \emptyset$ then we consider S_i a candidate for a large intersection. Then we compute $S_i \cap S$ for these candidates only, which saves much time.
Let S_j be a fixed set with intersection size $k := |S_j \cap S|$. We want to limit the probability to miss S_j to some fixed number $\epsilon > 0$. Which size $r = |R|$ is needed? To simplify the calculation you may assume that the elements for R are idependently sampled with replacement. (Pick any element of S with uniform probability, put it back to S, and so on. Thus R might contain the same element multiple times.)
In the previous question the set S_j was fixed. Now we want to limit the probability to miss *any* of the sets S_j with $|S_j \cap S| \geq k$ to some fixed number $\epsilon > 0$. Which size $r = |R|$ is needed for this stronger guarantee?

Problem 4.
In the confused voters problem (hand-in exercise 7), k people want to vote for candidate A, and $n-k$ people for candidate B, but everybody votes for the wrong candidate independently with some probability p. Let $a := k/n$ be the fraction of dedicated voters for A. Suppose that $p < 1/2$ and $a < 1/2$ are fixed. Clearly, $a < 1/2$ means that B should win.
Show that the failure probability (of A winning the election) decreases exponentially as a function of n. Note that you don't have to do a full calculation; you are only asked to prove the exponential decrease.
Moreover, for obtaining this conclusion, do you need the assumption that the voters' errors are independent?

Problem 5.
A hitting set intersects every set in a given set family. Suppose that each of the given sets has at most 3 elements. Then we can find find a hitting set of k elements (if some exists) by a branching algorithm in $O^*(3^k)$ time (we refer to hand-in exercise 11).
We can improve the time considerably with little effort: If all pairs of sets intersect in 0 or 2 elements, then the problem becomes polynomial. (This is not very hard to show, but you don't have to do this here.) It remains the case that some of the sets share exactly 1 element, such as $\{a, b, c\}$ and $\{c, d, e\}$.
Your task: Give a branching rule such that the number of leaves in the search tree is bounded by a function $T(k)$ that satisfies the recurrence $T(k) = T(k-1) + 4T(k-2)$. Show how to get this recurrence, then solve it, and write the resulting time bound in O^* notation.

Problem 6.
In an undirected graph, the distance between nodes u and v is the edge number of a shortest path connecting u and v. A d-independent set is a subset of nodes where all pairwise distances are larger than d. (Hence a 1-independent set is a usual independent set.)
Give a polynomial-time algorithm to compute a maximum-size 2-independent set in a tree. Nodes are not weighted.
Ideally it should be a greedy algorithm. (But do not forget to prove its correctness.)
Plan B if you cannot come up with a greedy algorithm: Use dynamic programming in the tree. Give and explain the recursive formula(s).