

Maximum Points: 45  
Extra Credit: 10 Points

### Instructions:

1. Although this exam is an open book exam, you are only allowed to consult standard textbooks or lecture notes that we used in the course. You cannot consult the WWW.
  - (a) Introduction to Algorithms (Cormen, Leiserson, Rivest, Stein)
  - (b) Algorithm Design by Kleinberg and Tardos
  - (c) Algorithms by Dasgupta, Papadimitriou and Vazirani
2. This test is designed to give you an opportunity to demonstrate *your* understanding of principles of algorithm design, analysis and application. Consequently, you are required to work by yourself in answering the questions. Group work or consultation with others is not allowed. You can check with the instructor or the TA for any clarification.
3. Guidelines for writing a solution:
  - (a) Write your solutions neatly. Proofread them before you submit.
  - (b) Clearly state the main ideas and strategies used in developing the algorithm.
  - (c) Write the algorithm using a high-level pseudo language providing important details.
  - (d) If you are using a modified version of a well-known algorithm, show clearly what your modifications are.
  - (e) If you are using a well-known algorithm without any modifications, it is enough to specify how you plan to apply the algorithm (for example, by specifying the input).
  - (f) Provide a detailed proof of correctness.
  - (g) Analyze the time complexity of your algorithm.
4. Well-written solutions will be rewarded. Characteristics of good solutions: correct solution, logical coherence, an appropriate balance between completeness and succinctness, and an orderly presentation of ideas. Excessively long solutions will be penalized.
5. There are five problems each carrying *nine* points towards regular credit and two problems each carrying *five* points towards extra credit.

### Problems:

1. A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence  
A, C, G, T, G, T, C, A, A, A, T, C, G  
has many palindromic subsequences including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic). Devise an algorithm that takes a sequence  $x[1 \dots n]$  and returns the length of the longest palindromic subsequence. Its running time should be  $O(n^2)$ .
2. Let  $T$  be rooted tree, not necessarily binary. There is a weight associated with each vertex, such that the weight of a vertex is greater than the weight of the vertex's parent (in other words, the weights satisfy the heap condition with the minimal weight on the top). Each vertex can be designated as either a *regular* vertex or a *pivot* vertex. The cost of a pivot vertex is the same as its weight. Regular vertices get discounts, however — their cost is their

weight minus the weight of the closest proper ancestor that is a pivot vertex if such a proper ancestor exists. If there is no proper ancestor that is a pivot element, then the cost is the weight itself. Thus, selecting a vertex as pivot vertex may increase its cost, but it will also decrease the costs of some of its descendants. There is no limit on the number of pivot vertices. Design an efficient algorithm to designate each vertex as either regular or pivot, such that the total cost of all vertices is minimized.

3. We are given  $n$  jobs  $J_1, J_2, \dots, J_n$  and  $k$  identical processors. Each job  $J_i$  requires total processing time  $p_i$  on any one of the processors. Moreover, *preemption* is allowed. That is,  $J_i$  need not be processed during an entire interval of length  $p_i$ , nor always on the same processor. However, it is processed on at most one processor at a given time, and it is subject to a release date  $r_i$  and a due date  $d_i$ . That is, the processing of  $J_i$  cannot begin before  $r_i$  and must be completed by  $d_i$ . Present a polynomial-time (in  $n$  and  $k$ ) algorithm to find a schedule that satisfies these requirements.
4. Design an efficient algorithm to find a minimum-size vertex cover for a given bipartite graph.
5. Problem 7, Chapter 11, Page 654 (Kleinberg and Tardos)

**Extra Credit Problems:** Each extra credit problem carries **5 points**.

1. Consider the following optimization version of the PARTITION problem:  
Given  $n$  integers  $c_1, \dots, c_n$ , find a partition of  $\{1, 2, \dots, n\}$  into two subsets  $S_1$  and  $S_2$  that minimizes the quantity  $\max(\sum_{i \in S_1} c_i, \sum_{i \in S_2} c_i)$ .  
Consider the following heuristic for some fixed integer  $k$ :
  1. Choose the  $k$  largest  $c_i$ 's
  2. Find the optimal partition of these  $k$  integers
  3. Complete this into a partition of  $\{1, 2, \dots, n\}$  by considering each of the remaining  $c_i$ 's and adding it to the partition which at the time has the smallest sum.Prove that this algorithm with  $O(2^k + n)$  time complexity outputs a sum that is within  $(k + 3)/(k + 1)$  fraction of the optimal output.
2. Let  $G = (V, E)$  be a connected weighted undirected graph. Let  $T$  be a minimum-cost spanning tree of  $G$ . Suppose that we now add a new vertex  $v$  to  $G$ , together with some weighted edges from  $v$  to vertices of  $G$ . Design a linear-time algorithm (time proportional to  $|V|$ ) to find a new minimum-cost spanning tree that includes  $v$ . Assume that the tree is rooted and that for each node, its parent and child pointers are already available.