Due: Thursday, October 5, 2017, 2PM on MarkUs

You will receive 20% of the points for any (sub)problem for which you write "I do not know how to answer this question." You will receive 10% if you leave a question blank. If instead you submit irrelevant or erroneous answers you will receive 0 points. You may receive partial credit for the work that is clearly "on the right track."

1. (20 pts) You are given an array A of n complex numbers. Some of these numbers might be identical. For $i \neq j \in [n]$ you can invoke a comparison procedure that returns whether the numbers $A[i]$ and $A[j]$ are identical or not. Design a divide and conquer algorithm to decide whether there is a number in the array that appears more than $\lceil n/3 \rceil$ times in A using $O(n \log n)$ invocations of the comparison procedure. (For simplicity you may assume $n = 2^k$ for some k and that is a hint.) Solutions using asymptotically more invocations of the comparison procedure will not receive any credit.

   (a) In English, state the high level idea of your algorithm.

   (b) Describe your algorithm in pseudocode.

   (c) Provide a correctness argument for your algorithm.

   (d) State the recurrence that describes the number of invocations of the comparison procedure (do not forget the base case).

2. (20 points) You are given two sorted lists A_1 and A_2, each with n real numbers. Assume all numbers in $A_1 \cup A_2$ are distinct. Design a divide and conquer algorithm that returns the median of $A_1 \cup A_2$ using $O(\log n)$ comparisons that test whether or not $x \leq y$. The median of m numbers $x_1 \leq x_2 \ldots \leq x_m$ is $x_{\lceil m/2 \rceil}$.

   (a) In English, state the high level idea of your algorithm.

   (b) Describe your algorithm in pseudocode.

   (c) Provide a correctness argument for your algorithm.

   (d) State the recurrence that describes the number of invocations of the comparison procedure (do not forget the base case).

3. (30 pts) Consider the following question concerning the MST problem where the input is a connected graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}$.

   (a) Let $e = (u, v) \in E$ be an arbitrary edge. We wish to compute a MST $T = (V, E')$ for G subject to the constraint that $e \in T$. Consider the following divide and conquer type algorithm.

   - Partition the vertices V into two sets V_1 and V_2 having sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ such that $u \in V_1$ and $v \in V_2$.
   - Compute a minimum spanning tree or minimum spanning forest $T_1 = (V_1, E_1)$ for the graph induced by V_1 and $T_2 = (V_2, E_2)$ for the graph induced by V_2.
     Note: by spanning forest we mean a collection of trees, with the fewest possible number of connected components. Kruskal's algorithm would compute a minimum spanning forest.

- Return $T = (V, E_1 \cup E_2 \cup \{e\})$

Is the solution T necessarily a spanning tree? If T is a spanning tree, is it necessarily a minimum spanning tree?

(b) Consider the constrained MST problem as above. Design a greedy algorithm for this problem. Give an informal but convincing argument as to why your algorithm computes an optimal spanning tree subject to the constraint.

(c) Returning to the standard (unconstrained) MST problem, consider the case when all the edge costs are distinct. Is the MST unique? If so, provide a brief argument; otherwise, provide a counter-example.

4. (20 points) Recall the EFT algorithm for interval scheduling on one machine. We wish to extend this algorithm to m machines. That is, when considering the i^{th} interval I_j we will schedule it if there is an available machine. But on which machine? That is, what is the tie breaking rule?

(a) Consider the First-Fit EFT greedy algorithm:

- Sort the intervals $\{I_j\}$ so that $f_1 \leq f_2 \ldots \leq f_n$
- For $j = 1 \ldots n$
  For $\ell = 1 \ldots m$
  If I_j fits on M_ℓ then schedule I_j on M_ℓ
  EndFor
  EndFor

Is First-Fit an optimal algorithm? If yes, provide a proof; otherwise provide a counter-example.

(b) Consider Best-Fit EFT greedy algorithm:

- Sort the intervals $\{I_j\}$ so that $f_1 \leq f_2 \ldots \leq f_n$
- For $j = 1 \ldots n$
  If there is a machine M_ℓ on which I_j can be scheduled, then
  schedule I_j on M_ℓ where $\ell = argmax_k\{f_k \leq s_j$ and I_k scheduled on $M_k\}$
  EndFor

Is Best-Fit an optimal algorithm? If yes, provide a proof; otherwise provide a counter-example.

5. (20 pts) You are given two arrays D (of positive integers) and P (of positive reals) of size n each. They describe n jobs. Job i is described by a deadline $D[i]$ and profit $P[i]$. Each job takes one unit of time to complete. Job i can be scheduled during any time interval $[t, t+1)$ with t being a positive integer as long as $t+1 \leq D[i]$ and no other job is scheduled during the same time interval. Your goal is to schedule a subset of jobs on a single machine to maximize the total profit - the sum of profits of all scheduled jobs. Design an efficient greedy algorithm for this problem.

(a) Describe your algorithm in plain English (maximum 5 short sentences).

(b) Describe your algorithm in pseudocode.

(c) Prove correctness of your greedy procedure. One possibile proof is to argue by induction that the partial solution constructed by the algorithm can be extended to an optimal solution. But you can use any type of valid proof argument.

(d) Analyze the running time of your algorithm (in terms of the total number of operations).

6. (20 points)

Consider the following $\{0, 1, 3\}$ knapsack problem. You are given a knapsack size bound S and n items $\{(s_1, v_1), \ldots (s_n, v_n)\}$. A feasible solution σ is a sequence $< c_1, \ldots, c_n >$ in $\{0, 1, 3\}^n$ such that $\sum_i c_i s_i \leq S$ and the value of such a solution is $V(\sigma) = \sum_i c_i v_i$. That is, every item in the knapsack occurs exactly once or three times. Give a polynomial time dynamic programming algorithm for optimally solving this problem assuming each size s_i is a positive integer and the knapsack bound $S \leq n^3$.

(a) Describe the semantic array.

(b) Describe the computational array. Don't forget the base case.

(c) Justify why the above two arrays are equivalent.

(d) What is the running time of your algorithm in terms of n. You can assume all operations take unit time and we only need to compute the optimum value (and not an optimum solution).

7. (20 pts) You are given an array A of $n \geq 3$ points in the Euclidean 2D space. The points $A[1], A[2], \ldots, A[n]$ form the vertices (in clockwise order) of the convex polygon P. A triangulation of P is a collection of $n - 3$ interior diagonals of P such that the diagonals do not intersect except possibly at the vertices. The total length of a triangulation of P is the sum of the lengths of the $n - 3$ interior diagonals used to form that triangulation. Design an efficient dynamic programming algorithm to find the total length of a triangulation with the minimum total length.

(a) Describe the semantic array.

(b) Describe the computational array. Don't forget the base case.

(c) Justify why the above two arrays are equivalent.

(d) What is the running time of your algorithm in terms of n. You can assume all operations take unit time and we only need to compute the optimum value (and not an optimum solution). .