Due: Thursday, February 6, 202 4:59PM 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."
You mnay choose to spend your time looking for solutions on the internet and may likely succeed in doing so but you probably won't understand the concepts that way and will then not do well on the quizzes, midterm and final. So at the very least try to do the assignment initially without searching the internet. If you obtain a solution directly from the internet, you must cite the link to avoid plagarizing.

1. (20 pts) You are writing a blog about restaurants in various cities. You have a list of the 16 top restaurants in Moscow but you don't trust the reviewer. You have a friend in Moscow is willing to help and all you want to do is provide the best of these 16 restaurants and the worst of these 16 restaurants. So you ask your friend to compare restaurants and give you his opinion on the best and the worst. How many retaurants does he/she have to compare?

   (a) (5 points)It is reasonably obvious how your friend can provide his/her opinion doing 30 comparisons. Say why?

   (b) (10 points) It is possible to provide the desired opinion in just 22 comparisons. Provide a divide and conquer algorithm that will achieve this bound. State your algorithm (for any arbitrary number n of restaurants) in pseudo code.

   (c) (5 points) State the recurrence (when there are $n = 2^k$ restaurants) that describes the number comparisons. (Do not forget the base case.)

2. (20 points) Let $a(x)$ be a polynomial of degree $n-1$ (say over the reals). We can evaluate $a(x)$ at a given point y in $O(n)$ $+, -, \times$ arithmetic operations. In fact, the precise number is $2(n-1)$ arithmetic operations using Horner's rule. Your goal is to provide an algorithm that will evaluate $a(x)$ at n distinct points $y_1, \ldots, y_n$. Evaluating at each y_i separately will result in a totasl of $O(n^2)$. operations. We are aiming for an algorithm that uses asymptotiallly much fewer arithmetic operations.

   Assume we can multiply two degree n polynomials in $O(n \log n)$ arithmetic operations. (Note: this can be done as we will indicate in class.) We also know that we can also do polynomial division in $O(n \log n)$ arithmetic operations. By polynomial division, we mean that for polynomials $a(x)$ and $b(x)$ we can compute polynomials $q(x)$ and $r(x)$ such that $a(x) = q(x)b(x) + r(x)$ with degree $r <$ degree b.

   For simplicity (but without loss of generality), let $n = 2^k$.

   (a) (5 points)
   With or without recursion, show how to compute the polynomial $\prod_{i=1}^{n}(x - y_i)$ where $y_1, \ldots, y_n$ are n real numbers. We suggest a recursive algorithm whose recursion will make it easier to estimate the complexity.

   (b) (10 points) Provide a divide a conquer algorithm for evaluating a degree $n-1$ polynomial $a(x)$ at n distinct points $y_1, \ldots, y_n$ using only $T(n) = O(n \log^2 n)$ airthmetic operations. Hint: Divide $a(x)$ by an appropriate $b(x)$ of degree $n/2$ that can be efficiently computed.

   (c) (5 points) Indicate the recursion and justfiy the bound on the number of arithmetic operations using the assumptions about the complexity of polynomial multiplication and division. What would the bound be if we only assume that polynomial multiplication and division can be done in $O(n^{\log_2 3})$.

3. (20 points) The maximum subarray sum problem

   Given an array A of n signed integers, design a divide and conquer algorithm in time $O(nlog(n))$ to find a subarray $A[i, ..., j]$ such that $A[i] + A[i + 1] + ... + A[j]$ is maximized $(1 \leq i \leq j \leq n)$. In other words, find a pair (i, j) such that $\forall(x, y)\Sigma_{k=i}^{j} A[k] \geq \Sigma_{k'=x}^{y} A[k']$.

   - (10 points)In English, describe your algorithm.
   - (5 points) Describe your algorithm in pseudocode.
   - (5) Using an appropriate recurrence, analyze the running time of your algorithm. (Do not forget the base case).

4. (15 points)

Consider the following greedy algorithm for graph colouring. Without loss of generality we will let the input $G = (V, E)$ be a connected graph with $V = \{v_1, v_2, \ldots, v_n\}$ $(n \geq 1)$ and let the colours be $C = \{1, 2, \ldots\}$. We also let $Nbhd(v)$ denote the neighbourhood (i.e. adjacent vertices) of node v.

GREEDY COLOURING ALGORITHM (using breadth first search)

Let $\chi(v_1) = 1; L(0) := \{v_1\}; i := 0$ % $\chi()$ is the colouring function
    % $L(i)$ will denote the nodes in the current level of the breadth first search
Let $A := \{v_1\}$ % A will be the nodes already coloured
Let $U := V - \{v_1\}$ % U will be the nodes not yet coloured
While $U \neq \emptyset$
    $L(i+1) := \emptyset$
    For $j := 1..n : v_j \in L(i)$
        For $k = 1..n : v_k \in Nbhd(v_j) \cap U$
          % colour the node being added to the next level
            $\chi(v_k) := min_{c \in C} : c \notin \cup_{v_h \in Nbhd(v_k) \cap A} \chi(v_h)$
            $U := U - \{v_k\}; A := A \cup \{v_k\}$
            $L(i+1) := L(i+1) \cup \{v_k\}$
        END For
    END For
    $i := i + 1$
END While

(a) (10 points) Give a short but convincing argument showing that the above greedy algorithm will colour every 2-colourable graph using 2 colours; that is, the greedy algorithm is optimal for 2-colourable graphs.

(b) (5 points) Give an example of a 3-colourable graph for which the above greedy algorithm will use more than 3 colours. You may label the verticess in any way (which then fixes the order in which the algorithm assigns colours,)

5. (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) (10 points) 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) (10 points) 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.

6. (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) (5 points) Describe your algorithm in plain English (maximum 5 short sentences).

   (b) (5 points) Describe your algorithm in pseudocode.

   (c) Prove (5 points) 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) (5 points) Analyze the running time of your algorithm (in terms of the total number of operations).

7. (30 points) A is an array of n integers. Design a dynamic programming algorithm to find a subset $S \subseteq \{1, 2, 3, ..., n\}$ that maximizes $\sum_{i \in S} A[i]$, subject to the constraint that no two members of S can point to two consecutive elements of A, i.e., if $i \in S$ then $(i + 1) \notin S$.

   - (15 points) Describe your algorithm by giving a semantic array V (i.e. what is in each entry of the array) and a recursively defined array V' that will be equal to V in each entry. Don't forget any base case(s) and explain how the desired set S is obtained from V. Provide a brief explanation that $V = V'$.
   - (10 points) Implement your algorithm as an iterative (non recursive) algorithm.
   - (5 points) What is the aysmptotic time comlexity of your algorithm whether executed iteratively or recursively (with memoization).