CSE 421 : Sample Midterm Exam 2

Name:	**NetID:** @uw.edu

Instructions

- This Sample Midterm was assembled from problems given in previous 421 exams.
- The exam here is approximately the length of an "in-class" (50 minute) exam. Yours may be of somewhat different length.
- No electronics are allowed at the exam. These include laptops, tablets, calculators, smartphones, smart watches, etc. If it has an on/off switch, it needs to be off. If it does not, it needs to be stored in your bag.
- You will receive a 1-page two-sided reference sheet, a sample of which is part of this sample exam.
- This is a *closed book* exam with the following exception:
  - Other than your writing implements, you are permitted one piece of 8.5x11 inch paper with handwritten notes (notes are allowed on both sides of the paper).
  - Your handwritten page must be clearly labelled with your name and Student number or UW Netid.
  - You must hand in both your handwritten page and your copy of the reference sheet along with your completed exam.

Advice

- Write your solutions in the appropriate spaces. The backs of pages are also available for solutions. if you use them to extend an answer to a question, please put a pointer from that question to the place on the back that you use.
- Move around the exam; if you get stuck on a problem, save it until the end.
- Proofs are not required unless otherwise stated.
- Remember to take deep breaths.

Question	**Max points**
Short Answer	0
k-wise merge	0
Product-Sum	0
Path in a DAG	0
Total	**???**

1. Short Answer

(a) True or False: Let $G = (V, E)$ be a weighted undirected graph. Suppose all edges in E have distinct weights except for two edges which share the same weight. The graph has exactly two minimum spanning trees.

(b) True or False: Let $G = (V, E)$ be a graph where its BFS tree contains at least 3 layers. Consider the cut $(L_0 \cup L_1, L_2)$ (i.e. all nodes that are 2 edges away from s form one side of the cut). All edges which cross the cut are tree edges.

(c) True or False: If $T(n) = 12T(n/6) + n^2, T(1) = 1$, then $T(n) = O(n^2 \log n)$. Briefly justify your answer.

(d) True or False: If $T(n) \leq 2T(n/2) + n \log n$, and $T(1) = 1$, then $T(n) = O(n^2)$. Briefly justify your answer.

(e) True or False: If we're given preference lists for there is a stable matching that gives no agent their first choice then there exists at least 3 stable matchings.

2. Celebrity

There is a party of n guests $A = [a_1, ..., a_n]$, with one celebrity in attendance. We define a "celebrity" as a guest whom all other guests know, yet who knows no other guests. Specifically, if guest a_i is the celebrity, $\forall a_j \neq a_i$, $knows(a_j, a_i) == true$ and $knows(a_i, a_j) == false$. Other guests may or may not know each other, as "normal" parties go.

(a) Describe an efficient divide and conquer algorithm `CELEBRITY`(A) which returns the celebrity. Suppose you have access to the $knows$ function above, and that it runs in constant time.

(b) What is the run time of your algorithm in terms of the number of guests, n?

3. Present prank

I want to play a prank on my brother for his birthday by putting his tiny gift in a bunch of progressively larger boxes, so that when he opens the large box there's a smaller box inside, which contains a smaller box, etc. until he's finally gotten to the tiny gift inside. Write a dynamic programming algorithm which, given a list of dimensions (length, width, and height) of n boxes, returns the maximum number of boxes I can nest (i.e. gives the count of the maximum number of boxes my brother must open). You may assume you have access to a method `FITS`(b_1, b_2) which indicates wether the box $b_1 = (x_1, y_1, z_1)$ fits within box $b_2 = (x_2, y_2, z_2)$.

(a) First, show that this greedy algorithm is not correct: First select the box with the smallest volume. Next select the box with the smallest volume which fits inside of it. Repeat.

(b) Give the optimization formula for computing the maximum depth nesting of boxes if box j is the outer-most box.

4. Stunt Planning

Jackie Chan is trying to plan a stunt to rapidly descend a tall building. Jackie will leap from balcony to balcony on the building until he reaches the ground. The building is n meters tall, and you have a list of the heights of the m balconies $h = [h_1, \ldots h_m]$ where h is sorted from highest to lowest balcony. The ground is at height 0. Jackie knows that any fall greater than k feet will cause injury.

(a) Describe an efficient greedy algorithm `SafeDescent`(n, h, k) that finds the shortest sub sequence of balconies which safely get Jackie to the ground.

(b) Argue the correctness of your algorithm using either a greedy stays ahead argument or else an exchange argument.

Reference Sheet

Unless explicitly stated otherwise, you may use any algorithm discussed in this class to solve a problem. You can also use any of the data structures from CSE 332.
In particular, you may use any of these functions as libraries (this list is not exhaustive).

Graph Search

- `TwoColor`(G) returns `True` if G can be 2-colored (i.e., is bipartite), `False` otherwise. Running time $\Theta(m+n)$
- `ConnectedComponents`(G) finds the connected components of an undirected graph G. You may assume you get any reasonable representation of this information. Running time $\Theta(m+n)$
- `StronglyConnectedComponents`(G) finds the strongly connected components of a directed graph G. You may assume you get any reasonable representation of the information. Running time $\Theta(m+n)$
- `TopologicalSort`(G) returns a `list` of vertices of a directed acyclic graph (DAG) G in topological order, or `null` if the graph has a cycle. Running time $\Theta(m+n)$
- `CondensationGraph`(G) given a directed graph G returns the DAG of strongly connected components of G, also known as the condensation of G. Running time $\Theta(m+n)$.

Other Graph Algorithms

- `Dijkstra`(G, s, t) finds the length of the shortest path from s to t in a (non-negative) weighted, directed graph G. If t is `null`, the distances from s are stored in every vertex. Running time $\Theta(m+n\log n)$
- `Prim`(G) finds the minimum spanning tree of a (weighted, undirected) graph G. Running time $min(\Theta(m\log n), \Theta(n^2))$
- `Kruskal`(G) finds the minimum spanning tree of a (weighted, undirected) graph G. Running time $\Theta(m\log n)$

Arrays

- `QuickSelect`(A, k) a randomized algorithm returns the value that would be at index k of A if A were sorted. Running time $\Theta(n)$
- `BFKRTSelect`(A, k) a deterministic algorithm returns the value that would be at index k of A if A were sorted. Running time $\Theta(n)$
- `MaxSubarraySum`(A) returns the sum of the maximum sum (contiguous) subarray of A. Running time $\Theta(n)$
- `MergeSort`(A) returns the sorted version of A. Running time $\Theta(n\log n)$

Other Algorithms

- `GaleShapley(proposerPrefs, receiverPrefs)` returns the proposer-optimal stable matching.
  Running time $\Theta(n^2)$ for n proposers and n receivers of proposals.
- `2dClosestPoints`(A) returns the distance between the two closest points of A (where A contains vectors in $\mathbb{R}^2$). Running time $\Theta(n\log n)$
- `LIS`(A) returns the longest increasing sub-sequence (LIS) in an array of integers A of length n. Running time $O(n^2)$.
- `EditDistance`(x, y) returns the edit distance between strings x and y.
  Running time $\Theta(mn)$ for strings of length m, n

There's more information on the back!

Greedy Proof Methods

- Greedy Stays Ahead: Show that for each k, the first k choices made by the greedy algorithm are better (according to some property) than the corresponding options from any other solution,.
- Exchange Argument: There is a finite series of local changes that allow one to transform any solution (including any optimal one) to the greedy solution without losing solution quality. (Alternatively, any solution that is not the greedy solution is either equivalent to greedy or can be strictly improved.)
- Structural: There is some basic property, independent of any algorithm, that limits the quality of any solution to the value produced by the greedy solution.

Divide and Conquer Master Theorem For a recurrence of the following form, where a, b, c, d are constants

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Where $f(n)$ is $\Theta\left(n^k\right)$ for $k \geq 0, a \in \mathbb{Z}^+, k \geq 0$

- If $a < b^k$ then $T(n) \in \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) \in \Theta\left(n^k \cdot \log(n)\right)$
- If $a > b^k$ then $T(n) \in \Theta\left(n^{\log_b(a)}\right)$

Dynamic Programming recurrence patterns

- Fibonacci: 1-dimensional, each element depends on $O(1)$ predecessors, $O(1)$ prior.
- Weighted Interval Scheduling: 1-dimensional each element depends on $O(1)$ predecessors, arbitrarily prior.
- Longest Increasing Subsequence: 1-dimensional, each element depends on all predecessors.
- Edit Distance/Sequence Alignment: 2-dimensional based on prefix indices of two different strings.