CSE 421 : Sample Midterm Exam

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: If all edges in a graph have weight 1, then there is an $O(m+n)$ time algorithm to find the minimum spanning tree, where m is the number of edges and n is the number of vertices. Briefly justify your answer.

(b) True or False: If an undirected connected graph G has a unique heaviest weight edge e, then e cannot be part of any minimum spanning tree. Briefly justify your answer.

(c) True or False: If $T(n) \leq 10T(n/3) + n^3, T(1) = 1$, then $T(n) = O(n^3)$. Briefly justify your answer.

(d) True or False: If the running time of an algorithm satisfies the recurrence $T(n) = 3T(n/6) + cn$, for some positive constant c, and $T(1) = 1$, then $T(n) = O(n)$. Briefly justify your answer.

(e) Give an example of preference lists for two pairs of agents, R and H, such that the R-optimal stable matching (where each agent in R paired with their best valid partner) and the H-optimal stable matching (where each agent in H is paired with their best valid partner) are different.

2. k-Wise Merge

A k-wise merge takes as input k sorted arrays, and constructs a single sorted array containing all of the elements of the input arrays.

(a) Describe an efficient divide and conquer algorithm `MultiMerge`$(k, A_1, ..., A_k)$ which computes a k-wise merge of its input arrays.

fontsize

(b) What is the run time of your algorithm with input of k arrays, each of length n.

3. Product-Sum

Given a list of n integers, $v_1, ..., v_n$, the **product-sum** of the list is the largest sum that can be formed by multiplying adjacent elements in the list.
Each element can be matched with at most one of its neighbors.

For example, given the list 1,2,3,1 the product-sum is $8 = 1 + (2 \times 3) + 1$.
Given the list 2,2,1,3,2,1,2,2,1,2, the product-sum is $19 = (2\times2)+1+(3\times2)+1+(2\times2)+1+2$.

(a) Compute the product-sum of 1,4,3,2,3,4,2.

(b) Give the optimization formula for computing the product-sum of the first j elements.

(c) Give pseudocode for the (iterative) dynamic program for computing the value of the product-sum of a list of integers. (You do not need to determine which pairs multiply together - just the number.)

4. Path in a DAG

Given a directed graph $G = (V, E)$ with no cycle, describe a $\mathcal{O}(|E| + |V|)$ time algorithm to check if there is a directed path that touches every vertex exactly one. Prove the correctness and the run time of the algorithm.

Hint: You can use the fact that topological sort can be done in $\mathcal{O}(|E| + |V|)$ time.

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.
- Knapsack: 2-dimensional, adding extra parameter (weight) to natural 1-dimensional input.
- RNA Secondary Structure (Dynamic Programming over Intervals): 2-dimensional: beginning and end of each inteval.
- Edit Distance/Sequence Alignment: 2-dimensional based on prefix indices of two different strings.