CSE 421 : Sample Midterm Exam Solutions

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.

Solution:

True. In this case, all spanning trees have the same weight, so we can use breadth first search to find a spanning tree.

(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.

Solution:

False. If the edge is the only edge that connects a particular vertex, it must be included in every spanning tree.

(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.

Solution:

True. By master theorem, since $3^3 > 10, T(n) = O(n^3)$.

(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.

Solution:

True. Since $3 < 6^1$, the master theorem says that the solution is $O(n)$

(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.

Solution:

r_1's preference list: (h_1, h_2)
r_2's preference list: (h_2, h_1)
h_1's preference list: (r_2, r_1)
h_2's preference list: (r_1, r_2)

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

Solution:

function MULTIMERGE($k, A_1, ..., A_k$)
  if $k = 2$ **then**:
    return Merge(A_1, A_2)
  else
    $B_1 :=$ MultiMerge$(\frac{k}{2}, A_1, ..., A_{\frac{k}{2}})$
    $B_2 :=$ MultiMerge$(\frac{k}{2}, A_{\frac{k}{2}+1}, ..., A_k)$
    return Merge(B_1, B_2)

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

Solution:

The run time of the algorithm is $O(kn \log k)$. One way to see this is to write the run time as a recurrence. Let cn be a bound on the cost of merging two arrays of length n. The recurrence for the run time is $T(k) = 2T(\frac{k}{2}) + ckn$, so the solution is $ckn \log k$.

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.

Solution:

$29 = 1 + (4 \times 3) + 2 + (3 \times 4) + 2$

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

Solution:

$$OPT[j] = \begin{cases} \max(OPT[j-1] + v_j, OPT[j-2] + v_j \cdot v_{j-1}) & \text{if } j \geq 2 \\ v_1 & \text{if } j = 1 \\ 0 & \text{if } j = 0 \end{cases}$$

(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.)

Solution:

```
function PROD-SUM(int[]v, n)
    if n == 0 then:
        return 0
    int [] OPT = new int[n + 1]
    OPT[0] = 0
    OPT[1] = v[1]
    for int j = 2 to n do
        OPT[j] = max(OPT[j − 1] + v[j], OPT[j − 2] + v[j] ∗ v[j − 1])
    return OPT[n]
```

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.

Solution:

Algorithm:

- Find the topological order $v_1, v_2, v_3, ..., v_n$.
- If there is an edge from $v_i \to v_{i+1}$ for all $i = 1, 2, ..., n-1$, output "YES"
- Else, output "NO"

Run time:
The run time is dominated by topological sort which is $O(|E| + |V|)$ time.

Correctness:
There are two cases:

Case 1: output "YES". In this case, the algorithm indeed finds a path $(v_1 \to v_2 \to ... \to v_n)$ that touches every vertex.

Case 2: output "NO". In this case, there is a i such that no edge goes from v_i to v_{i+1}. Therefore, any path touches v_i must go to v_j for some $j > i + 1$ (it cannot go to v_j with $j < i$ because of the topological order). However, after the path goes to v_j for $j > i + 1$, it cannot go back to v_{i+1} due to the topological order again. Hence, if the path touches v_i, it will miss v_{i+1}. Therefore, there is no path that touches every vertices.

In both cases, the algorithm is correct. (Note that the proof handled the disconnected case automatically).