NAME: ____________________________

CSE 421
Introduction to Algorithms
Midterm Exam Autumn 2021

Anup Rao
5 November 2021

DIRECTIONS:

• Answer the problems on the exam paper.
• No cheat sheets, no devices allowed.
• Justify all answers with proofs, unless the facts you need have been proved in class or in the book.
• If you need extra space use the back of a page
• You have 50 minutes to complete the exam.
• Please do not turn the exam over until you are instructed to do so.
• Good Luck!

<table>
<thead>
<tr>
<th>1</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>80</strong></td>
</tr>
</tbody>
</table>
1. (40 points, 5 each: 3 for the answer, 2 for the justification) For each of the following problems answer True or False and BRIEFLY JUSTIFY you answer.

(a) $n^{2.1} = O(n^2 \log n)$.
   False. $n^{0.1}$ grows faster than $\log n$, as we discussed in class.

(b) $2^{2n} = O(2^n)$.
   False. $2^{2n} = 2^n \cdot 2^n \gg c \cdot 2^n$ for any constant $c$.

(c) In the stable matching problem, if we have a company $C$ and an applicant $A$, and $A$ is last on $C$’s preference list, and $C$ is last on $A$’s preference list, then $C$ and $A$ can never be matched in any stable matching.
   False. E.g if there are applicants $A, B$ and companies $C, D$ and the prefs look like $D, C$ for every applicant, and $B, A$ for every company. Then $B - D$ and $A - C$ is stable.

(d) There is an $O(n^3)$ time algorithm to find the shortest path distance between every pair of vertices in a directed graph with non-negative edge weights.
   True. We discussed a dynamic programming algorithm with this performance for this problem.
(e) If the running time of an algorithm satisfies the recurrence $T(n) \leq T(n/10) + T(9n/10) + c(n)$, then $T(n) = O(n^{1.5})$.
True. In fact, $T(n) = O(n \log n)$. We discussed how to solve this when talking about the linear time median algorithm.

(f) Suppose an undirected graph has an edge $\{u, v\}$, and there is a vertex $w$ such that there is a path from $w$ to $u$ of length 11 and there is a path from $v$ to $w$ of length 19. Then the graph cannot be bipartite.
True. If the graph was bipartite, some pair, either $u, v$ or $w, u$ or $w, v$ must be on the same side of the bipartite partition. But the odd length paths imply that none of these three pairs lie on the same side.

(g) Suppose an undirected graph contains a 3-cycle whose edges have the weights 10, 10, 2. Then every MST must include the edge of weight 2.
False. Let $a, b, c$ be the three cycle, and suppose the graph has a verted $d$ that is connected to all of the other vertices by an edge of weight 1. Then the MST uses all of the edges touching $d$ and no other edges.

(h) There is an algorithm for multiplying two $n$-bit integers in time $O(n^{1.1})$.
True. This is a consequence of the FFT as we discussed in class. In fact, the running time for multiplying numbers is $O(n \log n \log \log n)$. 
2. (20 points) You are given $k$ sorted arrays, each with $n$ numbers in them. Give an algorithm for merging these arrays into a single sorted array of numbers that runs in time $O(nk \log k)$.

**Solution:**

In class we saw an algorithm to merge two lists of length $t$ in $O(t)$ time. We shall use that algorithm as a subroutine.

| **Input:** | $k$ sorted lists of $n$ numbers each |
| **Result:** | A single sorted list of the $n$ numbers |
|            | if $k = 1$ then |
|            | Output the single sorted list; |
|            | end |
|            | Recursively merge the first $k/2$ lists and the next $k/2$ lists; |
|            | Merge the final two lists using the algorithm from class and output it; |

**Algorithm 1: Merging**

The runtime of this algorithm satisfies $T(n, k) \leq 2T(n, k/2) + O(nk)$. Each level of the recursion costs $O(nk)$ and there are $O(\log k)$ levels of recursion. Thus the running time is $O(nk \log k)$. 
3. (20 points) Your friend wants to throw a party and is deciding whom to call. She has \( n \) people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they don’t know. Give a polynomial time algorithm for this task.

**Solution:** We give a greedy algorithm.

(a) Start with the set of all people \( S \).

(b) In each step if there is a person in \( S \) that is friends with less than 5 people in \( S \), or knows more than \(|S| - 5\) people, then remove that person from \( S \).

(c) Repeat this process until there are no such people, and output \( S \).

To see the correctness, we prove that the optimal solution \( T \) must always be contained in \( S \) at every stage of the execution. This is true initially, because \( S \) contains all people. At each step, we claim that if \( x \in S \) is about to be eliminated, then \( x \notin T \). This is because if \( x \in T \), then \( x \) must know 5 people in \( T \), and so must know 5 people in \( S \), and \( x \) must not know 5 people in \( T \), and so must not know 5 people in \( S \). So, the people in \( T \) will never be eliminated. Thus, the algorithm must terminate with \( S = T \).

To implement the algorithm efficiently, we can encode the input as the adjacency matrix of a graph. Then each check for a person takes \( O(n) \) time, so each iteration of the algorithm takes \( O(n^2) \) time, and the overall running time of the algorithm is at most \( O(n^3) \), since there can be at most \( n \) iterations before \( S \) becomes empty.