• This is a closed-book, closed-notes, open-brain exam. If you brought anything with you besides writing instruments and your handwritten $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

• Print your name, netid, and alias in the boxes above. Circle U if you are an undergrad, or G if you are a grad student. Print your name at the top of every page (in case the staple falls out!).

• **You should answer all the questions on the exam.**

• The last few pages of this booklet are blank. Use that for a scratch paper. Please let us know if you need more paper.

• If your cheat sheet if not hand written by yourself, or it is photocopied, please do not use it and leave it in front of the classroom.

• Submit your cheat sheet together with your exam. An exam without your cheat sheet attached to it will not be checked.

• If you are NOT using a cheat sheet you should indicate it in large friendly letters on this page.

• Time limit: 180 minutes.

• Relax. Breathe. It is almost over.

<table>
<thead>
<tr>
<th>#</th>
<th>Score</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>III</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
I. [20 Points]

A tournament is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A Hamiltonian path is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

A six-vertex tournament containing the Hamiltonian path $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

(a) [8 Points] Prove that every tournament contains at least one Hamiltonian path. (Give a short and concise proof - the answer will not be checked if it exceeds 200 words.)

(b) [8 Points] Describe an algorithm, as fast as possible, such that given as an input a tournament $G$ with $n$ vertices, represented using an adjacency matrix $A$, it outputs a Hamiltonian path of $G$. How fast is your algorithm?

(c) [4 Points] A random tournament $G$ is generated by randomly deciding, with probability half, for each pair of vertices $u, v$, whether to include the edge $u \rightarrow v$ in $G$, or the edge $v \rightarrow u$ in $G$.

Describe an algorithm, as fast as possible, such that given a random tournament $G$ with $n$ vertices, represented using an adjacency matrix $A$, it outputs a Hamiltonian path of $G$. What is the expected running time of your algorithm?
II. [20 Points] Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer $k$. Describe an algorithm, as fast as possible, to find the $k$th smallest element in the union of $A$ and $B$. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates.

How fast is your algorithm?
III. [20 Points] You are given a tree $T$ with $n$ nodes, and root $r$. In the following, you can assume that one can compute the depth of a node of $T$ in constant time, and that given a node $v$ and $k > 0$, one can compute the $\text{ancestor}(v, k) = \text{parent}^{(k)}(v)$ of $v$ in constant time.

(a) [5 Points] Show how to preprocess $T$, such that given a query which is made out of two nodes $u, v$ of $T$, one can decide, in $O(1)$ time, if $u$ is a child of $v$. (we denote this operation by $\text{IsChild}(u, v)$.)

(b) [5 Points] Using (a), describe an algorithm, as fast as possible, such that given a query made out of two nodes $u, v$ of $T$, it computes the LCA (least common ancestor) of $u$ and $v$ in $T$. How fast is your algorithm as a function of $n$? How fast is it in term of $h$ ($h$ is the height of $T$)?

(c) [3 Points] Assume that you are given an operation $\text{FindOne}(x)$, which returns the location of the first bit of $x$ ($x$ is a non-negative integer number) which is non-zero (starting from the least significant bit). For example $\text{FirstOne}(6) = \text{FirstOne}(110_2) = 1$ and $\text{FirstOne}(5) = \text{FirstOne}(101_2) = 0$, and $\text{FirstOne}(16) = \text{FirstOne}(10000_2) = 4$. Describe a function, such that given two positive integers $a, b$ it computes the first bit in which they differ. You can assume that you are allowed to use all standard bitwise operations, like $\text{AND}$, $\text{OR}$, $\text{XOR}$, and $\text{NOT}$.

(d) [6 Points] Assume that every node in $T$ has at most two children. Show how to preprocess $T$, so that given two nodes $u, v$ of $T$, one can compute their LCA in $O(1)$ time. How much time does the preprocessing takes?

(e) [1 Points] Why is the solution for (d) is unacceptable in practice?
IV. [20 Points] An instance of 2SAT is an instance of SATISFIABILITY, where each clause has at most 2 variables. For example $U = (a + b)(\overline{a} + c)d$ is a 2SAT formula.

(a) [5 Points] Describe an algorithm $\text{shrink}(F)$ that generates from a 2SAT formula $F$ with $n$ variables and $m$ clauses, a formula $F'$ (with at most $n$ variables), where each clause has exactly two distinct variables. Furthermore, $F$ is satisfiable if and only if $F'$ is satisfiable. (Hint: in the formula $U = (a + b)(\overline{a} + c)d$ the value of $d$ must be 1 in any satisfiable assignment for $U$, and as such $U' = (a + b)(\overline{a} + c)$ is a valid output.)

How fast is your algorithm?

(b) [5 Points] Given a 2SAT formula $F$ with $n$ variables $x_1, \ldots, x_n$. Describe a procedure $\text{reduce}(F, i, b)$ which assigns the variable $x_i$ the value $b$, and outputs a formula $F'$, with at most $n - 1$ variables, such that $F$ is satisfiable, if and only if, $F'$ is satisfiable. Prove the correctness of your algorithm.

For example, for $U = (x_1 + x_2)(\overline{x_1} + x_3)d$ $\text{reduce}(U, 3, 1)$ would return $U' = (x_1 + x_2)$. How fast is your algorithm?

(c) [10 Points] Present an algorithm, as fast as possible, such that given a 2SAT formula $F$ over $n$ variables with $m$ clauses, it outputs whether $F$ is satisfiable or not. How fast is your algorithm?