UNIVERSITY OF CALIFORNIA, RIVERSIDE
DEPARTMENT OF COMPUTER SCIENCE

2006 DEPTH EXAMINATION IN THEORY OF
OF COMPUTATION AND ALGORITHMS

- There are 10 problems on the test. Each problem is worth 10 points. The ordering of the problems is unrelated to their difficulty.
- Answer exactly 8 out of the 10 questions. Indicate below which two problems you do not want to have graded.
- Write legibly. What can’t be read won’t be credited.
- Algorithms can be described informally in pseudo-code.
- Good luck!

Name:

Skip Problems:
1.
2.
Problem 1: Consider the following procedure which builds a heap when given as input an array $A$ of $n$ integers.

$$\text{BuildHeap}(A : \text{array of int})$$
1. for $i \leftarrow \lceil n/2 \rceil$ downto 1 do
2. \quad $\text{Heapify}(A, i)$

$$\text{Heapify}(A : \text{array of int}, i : \text{int})$$
1. $left, right \leftarrow 2i, 2i + 1$
2. if $(left \leq n)$ and $(A[left] > A[i])$ then $max \leftarrow left$ else $max \leftarrow i$
3. if $(right \leq n)$ and $(A[right] > A[max])$ then $max \leftarrow right$
4. if $(max \neq i)$ then
5. \quad swap $A[i]$ with $A[max]$
6. \quad $\text{Heapify}(A, max)$

$\text{BuildHeap}()$ starts with the element at position $\lceil n/2 \rceil$ and works backwards. $\text{Heapify}(A, i)$ pushes the element at position $i$ down the heap as far as necessary. The elements in positions $\lceil n/2 \rceil + 1$ through $n$ are leaves, and so they cannot be pushed further down. The other nodes are either in the correct position, or larger than their children; thus, a push-down fixes the sub-heap rooted at $i$ without affecting the rest of the heap.

a. Express the time-complexity of $\text{BuildHeap}$ as the sum of the number of operations performed at each level of the heap.

b. Show that the overall time complexity of the procedure $\text{BuildHeap}$ is $O(n)$. 
Problem 2: We are given a tree $T$ with $n$ nodes and a set $X$ of distinct nodes in $T$ of cardinality $2k$ (where $k \leq n/2$). A matching of $X$ in $T$ is a partition of $X$ into $k$ pairs of nodes $(x_1, y_1)$, $(x_2, y_2), \ldots, (x_k, y_k)$. Given $T$ and $X$ we want to find a matching such that, for any two matched pairs $(x_i, y_i)$ and $(x_j, y_j)$, the simple paths $p_i$ from $x_i$ to $y_i$ and $p_j$ from $x_j$ to $y_j$ are edge disjoint. In the example illustrated below, the nodes in $X$ are marked with a square and dashed lines show the paths between the matched vertices.

a. Prove that for each $T$ and $X$ there exists such a matching.

b. Give a linear-time algorithm to compute the matching and justify its correctness.
Problem 3: Consider the following Reverse-Delete algorithm for the Minimum Spanning Tree (MST) problem. Start with a connected edge-weighted graph $G = (V, E)$. Consider edges in order of decreasing cost (breaking ties arbitrarily). When considering an edge $e$, delete $e$ from $E$ unless doing so would disconnect the current graph. Prove that Reverse-Delete computes a MST.
Problem 4: Consider the following problem. Given a set $L$ of $n$ (assume $n$ even) positive integers, partition the set into two subsets $A$ and $B$ each of size $n/2$, such that the absolute value of the difference between the sums of the integers in the two subsets is minimized. In other words, we want to minimize $|\sum_{x \in A} x - \sum_{y \in B} y|$ where $A \cup B = L$, $A \cap B = \emptyset$ and $|A| = |B| = n/2$. Give a pseudo-polynomial time algorithm for this problem and analyze its time complexity.
Problem 5: State and prove the max-flow/min-cut theorem (as presented in class).
**Problem 6:** Describe a sequence of languages $L_1, L_2, L_3, \ldots$ such that each $L_n$ is recognized by an $n$-state deterministic finite automaton, but $L_n$ is not recognized by any $(n - 1)$-state deterministic finite automaton. Prove that your languages have the desired property.
Problem 7: Let $M$ be a non-deterministic polynomial-time Turing machine. Define the *probability* that $M$ accepts an input $w$ to be the number of accepting computation paths, divided by the total number of computation paths, when $M$ is run on $w$. Define $L_{1/2}(M)$ to be the set of words that are accepted by $M$ with probability greater than $1/2$.

a. Show that SAT equals $L_{1/2}(M)$ for some non-deterministic, polynomial-time Turing machine $M$.

b. Define the complexity class $C = \{L_{1/2}(M) : M$ is a non-deterministic poly-time TM\}. Show that if $C = \text{NP}$, then $\text{NP} = \text{co-NP}$. 
Problem 8: Recall that the configuration of a Turing machine is defined by a triple \( (q, w, u) \), where \( q \) is the current state, \( w \in \Sigma^* \) is the string to the left of the cursor, and \( u \in \Sigma^* \) is the string to the right of cursor (including the symbol scanned by the cursor). Consider the RepeatsConfiguration problem defined below.

**Instance:** A description \( M \) of a Turing machine and input tape \( I \);

**Query:** Does the execution of \( M \) on the input \( I \) ever repeat a configuration?

a. Is RepeatsConfiguration r.e.?

b. Is RepeatsConfiguration undecidable?

Justify your answers.
Problem 9: Define \( \text{CLIP}(L_1, L_2) = \{ u : \exists w \in L_2 \text{ such that } uw \in L_1 \} \). If \( L_1, L_2 \) are Turing-recognizable (r.e.) then so is \( \text{CLIP}(L_1, L_2) \)? Justify your answer.
Problem 10: A wheel in a graph $G = (V, E)$ is a subgraph that consists of a cycle and a “center” node connected to all nodes in this cycle. Prove that finding the largest wheel in a graph $G$ is NP-hard.