Name: ___________________________ Student ID: ___________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

Academic Integrity Statement. I understand and acknowledge that I must NOT disclose ANY information related to this CSE 101 final exam, to ANY OTHER PERSON, before 2:30pm Pacific Time on Thursday, March 21, 2013. I further understand and acknowledge that ANY such disclosure is a violation of UCSD’s Academic Integrity policy and, if discovered, will be immediately referred to university administration.

Signature: ___________________________

INSTRUCTIONS: Be clear and concise. Write your answers in the space provided. Use the backs of pages, and/or the scratch page at the end, for your scratchwork. All graphs are assumed to be simple. Good luck!
1. (a) (5 points) **Shortest Paths, MST.** Consider the following undirected graph.

Draw the edges in the shortest-paths (Dijkstra's) tree when A is the source.

![Diagram of graph with labeled edges]

Draw the edges in an MST.

![Diagram of MST with labeled edges]
(b) (5 points) **Maximum Flow.** Consider the following directed graph representing a flow network.

Run the Ford-Fulkerson algorithm to find a maximum $s$-$t$ flow in the given network. Draw and label the final residual graph and clearly indicate the value of the maximum flow.

Maximum Flow =
2. (10 points) **Short answer**

(a) (3 points) The following function is invoked on an array \( A \) of size \( n \).

```plaintext
procedure OPT(A)
    n ← |A|
    if n = 1:
        return
    i = ⌊n/2⌋
    OPT(A[1...i])
    OPT(A[i+1...n])
    for j = 1 to n:
        print(n)
```

Write down a recurrence relation for the running time of this function call. Solve the recurrence and state the big-O running time in terms of \( n \).

(b) (3 points) Match each statement (in the left column) to what is implied by the statement (in the right column). (Use each item exactly once.)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>There exists an ( NP ) problem which is also in ( P ).</td>
<td>( P = NP )</td>
</tr>
<tr>
<td>There exists a problem in ( P ) which is ( NP )-hard.</td>
<td>( P \neq NP )</td>
</tr>
<tr>
<td>There exists a problem in ( NP ) which is neither in ( P ) nor ( NP )-complete.</td>
<td>Neither</td>
</tr>
</tbody>
</table>
(c) (4 points) There are $n$ tasks $T = \{t_1, \ldots, t_n\}$ which need to be scheduled on $m$ servers $S = \{s_1, \ldots, s_m\}$. Each task $t_i$ can only be scheduled on a subset of the servers $S_i \subseteq S$. Further, each server $s_j$ has a capacity and can handle at most $c_j$ tasks where $c_j$ is a positive integer.

Explain how maximum flow can be used to determine if there is a way to execute all $n$ tasks using the available $m$ servers.
3. (10 points) **Greedy Algorithms.** Consider the following Minimum Total Waiting Time problem. You are given a set of n jobs $j_1, j_2, \ldots, j_n$ with durations $t_1, t_2, \ldots, t_n$. You can process only one job at a time. A job's *waiting time* is the amount of time until that job is finished. Your goal is to minimize the sum of the waiting times of all jobs.

Example: If $j_1$ has duration 3, $j_2$ has duration 1, and $j_3$ has duration 7, then processing the jobs in order $j_3, j_1, j_2$ would have total waiting time $7 + 10 + 11 = 28$. Processing the jobs in order $j_1, j_2, j_3$ would have total waiting time $3 + 4 + 11 = 18$.

(a) (4 points) State an efficient algorithm that finds the order of jobs that minimizes total waiting time.

(b) (5 points) Justify your algorithm's correctness.

(c) (1 point) State your algorithm's running time.
4. (10 points) **Dynamic Programming.** You are given a numerical expression for a sum $a_1 + a_2 + \cdots + a_n$ where each of the $a_i$ is a positive integer. Your task is to maximize the value of this expression under the following rule. You can replace any of the addition operators + with a product operator $\times$, subject to the condition that there cannot be two product operators in a row. You may not change the order of the operands $a_i$. For example, consider the expression

$$4 + 7 + 9 + 2 + 7$$

The optimum solution achieves a total of 81:

$$4 + 7 \times 9 + 2 \times 7 = 4 + 63 + 14 = 81$$

Note that the $\times$ operator has precedence over the + operator. Note also that an expression such as $4 \times 7 \times 9 + 2 + 7 = 252 + 2 + 7 = 261$ is not a valid solution, because it has two consecutive $\times$ operators.

Provide an efficient DP algorithm that, given an expression, finds the maximum possible value for the expression under the above rule.

(a) (3 points) Define your subproblems and state which subproblem’s solution gives the desired answer.

(b) (5 points) Give base case(s) and the recurrence for your DP algorithm.

(c) (2 points) Give the running time of your algorithm in big-O notation.
5. (10 points) **Self-Reduction.** Recall that in a graph $G = (V, E)$, an *independent set* is a set of vertices $V' \subseteq V$ such that no two vertices in $V'$ are adjacent.

Now, recall the problem of **INDEPENDENT SET**.

**INDEPENDENT SET**
- **Input:** An undirected graph $G = (V, E)$ and a nonnegative integer $k$.
- **Decision Problem:** Does $G$ contain an independent set of size $k$?
- **Search Problem:** Find an independent set $S \subseteq V$ with $|S| = k$ if one exists.

Suppose that you have an oracle (i.e., a black box solver) for the INDEPENDENT SET decision problem. Explain how to use this oracle a polynomial number of times to solve the search problem.

(a) (4 points) Describe your reduction from the search problem to the decision problem.

(b) (2 points) Show that you use only a polynomial number of calls to the oracle.
(c) (4 points) Prove the correctness of your reduction.
SCRATCH PAGE. DO NOT REMOVE.