Name: ______________________________ **RCS ID:** ____________ @rpi.edu

$\bowtie \bowtie \bowtie$ CSCI 2300 — Introduction to Algorithms $\bowtie \bowtie \bowtie$
Spring 2020 Final Exam (May 6, 2020) — v1.2 — SOLUTIONS

Overview

- This exam is open book(s), open notes; given that you are working remotely, you may use any and all of the posted course materials, including all previous questions and answers posted in the Discussion Forum.
- **Please do not search the Web for answers.** While we cannot stop you from doing so, such searching will likely lead you down the wrong path, answering the given questions using techniques we have not covered in this class. Therefore, some "correct" answers will not receive credit or partial credit, so please follow the instructions carefully and only use the techniques taught in this course.
- This exam is designed to take at most 120 minutes (for 50% extra time, the expected time is 180 minutes), but you can make use of the **full seven hours from 5:00-11:59PM EDT**.
- Long answers are difficult to grade; **please be brief and exact in your answers**; the space provided should be sufficient for each question.
- **All work on this exam must be your own; do not even think of copying or communicating with others about the exam during or for 24 hours after the exam.**
- Once we have graded your final exam, solutions will be posted along with final course grades in Rainbow Grades. The grade inquiry window for the final exam will be 24 hours, after which final course grades will be posted in SIS. If need be, contact `goldschmidt@gmail.com` directly after that window is closed.

Submitting your Exam Answers

- You must submit your exam file(s) by 11:59PM EDT on Submitty.
- Please submit a single PDF file that includes this cover page and is called `upload.pdf`. This will help streamline the online grading process.
- If you are unable to submit everything as a single PDF file, submit files that have filenames that clearly describe which question(s) you are answering in each file. Do not submit a `README` or any other extraneous files.
- If you face any problems, please email `goldschmidt@gmail.com` directly with details.

Academic Integrity Confirmation

Please sign or indicate below to confirm that you will not copy or cheat on this exam, which in part means that you will not communicate with anyone under any circumstances about this exam:

Signature or Typed Name: ______________________________
Failure to submit this page will result in a grade of 0 on the final exam.

1. **(8 POINTS)**

   (a) **SOLUTION (2pts):** ii. $f = \Omega(g)$, i.e., $n! = \Omega(2^n)$

   (b) **SOLUTION (2pts):** iii. $f = \Theta(g)$, i.e., $2^{n+1} = \Theta(2^n)$

   (c) **SOLUTION (2pts):** ii. $f = \Omega(g)$, i.e., $5^{\log_2 n} = \Omega(n^{1/2})$

   (d) **SOLUTION (2pts):** i. $f = O(g)$, i.e., $n2^n = O(3^n)$

2. **(6 POINTS)**

   (a) **SOLUTION (2pts):** $T(n) = 2T(\frac{n}{2}) + 1$

   (b) **SOLUTION (2pts):** $T(n) = O(n)$ or $T(n) = \Theta(n)$
   (using the Master Theorem with $a = 2$, $b = 2$, and $d = 0$)

   (c) **SOLUTION (2pts):** $T(n) = O(n^{0.5}) = O(\sqrt{n})$ or $T(n) = \Theta(n^{0.5}) = O(\sqrt{n})$
   (using the Master Theorem with $a = 2$, $b = 4$, and $d = 0$)

3. **(6 POINTS)**

   SOLUTION:

   → **2pts:** The graph has two strongly connected components and at least six vertices

   → **4pts:** The graph consists of two disjoint cycles and it is impossible to add one edge to make the graph strongly connected.

   Each cycle is individually a strongly connected component; however, adding just one edge to connect them is not enough as it (at most) allows us to go from one component to another, but not back.

4. **(10 POINTS)**

   SOLUTION:

   → **(4pts):** Valid and clearly described algorithmic approach
   → **(3pts):** Approach is correct for any graph in which $V_{odd} \subset V$
   → **(3pts):** Approach is correct for any graph in which $V_{odd} = V$

   The number of vertices with an odd degree in an undirected graph must be even. Suppose $|V_{odd}| = 2k$. We arbitrarily pair up these vertices and add an edge between each pair, causing all vertices to have an even degree.

   Each connected component of this new graph must have an *Eulerian tour*, i.e., a cycle that uses each edge exactly once. Removing the k edges added above, we end up with k paths with the two endpoint vertices of each path being vertices of odd degree. All of these paths must be edge-disjoint since an Eulerian tour uses each edge exactly once.

   All of these endpoint vertices form the pairs of vertices required by the original problem statement.

5. **(8 POINTS)**

   SOLUTION:

   $\rightarrow$ **(1pt for each case):** There are two possible cases here:

   (i) $\texttt{pre}(u) < \texttt{pre}(v) < \texttt{post}(v) < \texttt{post}(u)$

   (ii) $\texttt{pre}(v) < \texttt{post}(v) < \texttt{pre}(u) < \texttt{post}(u)$

   $\rightarrow$ **(2pts):** In case (i), vertex u is an ancestor of vertex v.

   $\rightarrow$ **(2pts):** In case (ii), vertex v was popped off the stack without looking at vertex u; however, since there is an edge between them and we look at all neighbors of v, this cannot happen.

   $\rightarrow$ **(2pts):** Therefore, the given statement is true.

6. **(8 POINTS)**

   SOLUTION:

   $\rightarrow$ **(2pts):** If a graph has exactly one odd cycle, it can be colored using three colors.

   $\rightarrow$ **(6pts or 3pts for partial solution):** To obtain a 3-coloring, delete one edge from the odd cycle. The resulting graph then has no odd cycles and can be colored using two colors by alternating between them using DFS or BFS (with details described). Finally, add back the deleted edge and (re)assign a third color to one of its endpoint vertices.

   Other variations here are fine, though partial (half) credit is given if only the odd-length cycle is considered. Full credit is given if a graph with exactly one odd-length is considered.

7. **(8 POINTS)**

SOLUTION:

→ **(1pt):** algorithm inputs, including additional output

```
procedure dijkstra(G,l,s)
Input:  Graph G=(V,E), directed or undirected;
        positive edge lengths l for each e in E;
        vertex s in V
Output: For all vertices u reachable from s,
        dist(u) is set to the distance from s to u
        and best(u) is set to minimum number of edges    ****
        traversed in a shortest path from s to u         ****
```

→ **(1pt):** algorithm initialization
→ **(1pt):** initialization of best() to infinity and best(s) to zero

```
for all u in V:
   dist(u) = infinity
   prev(u) = nil
   best(u) = infinity    ****
dist(s) = 0
best(s) = 0    **************

H = makequeue(V)   (using dist-values as keys)
```

→ **(2pts):** best(v) is updated as necessary
→ **(2pts):** prev(v) is updated as necessary

```
while H is not empty:
   u = deletemin(H)
   for all edges (u,v) in E:
      if dist(v) > dist(u) + l(u,v):
         dist(v) = dist(u) + l(u,v)
         prev(v) = u
         best(v) = best(u) + 1        ****
         decreasekey(H,v)
      if dist(v) = dist(u) + l(u,v):  ****
         if best(v) < best(u) + 1:    ****
            best(v) = best(u) + 1     ****
            prev(v) = u               ****
```

→ **(1pt):** The runtime is the same as Dijkstra's

8. **(8 POINTS)**

   SOLUTION:

   → **(2pts):** Since the given graph is connected, it will have an edge that can be removed that still leaves the graph connected if and only if the graph is *not* a tree. More specifically, for graph G, we require $|E| > |V| - 1$.

   → **(1pt):** The input is connected undirected graph $G = (V, E)$.

   → **(2pts):** We perform a DFS on G until we see $|V|$ edges. If we can find $|V|$ edges, then we return true; otherwise, we return false.

   → **(1pt):** The output is either true or false.

   → **(2pts):** The runtime of the algorithm is linear. For DFS, the runtime is $O(|V| + |E|)$. We could also state here that we can stop once we find $|V|$ edges.

9. **(12 POINTS)**

   (a) **SOLUTION:**

   → **(2pts):** This claim is false.

   → **(4pts):** To show this, consider a graph in which the shortest path between u and v is edge $\{u, v\}$ with weight 11, but edges $\{u, w\}$ and $\{v, w\}$ have weights 10 and 2, respectively. Other counterexamples certainly are fine here.

   (b) **SOLUTION:**

   → **(2pts):** This claim is true.

   → **(4pts):** Suppose that the path from vertex s to vertex t has an edge weight greater than m. Then one of the edges of the m-path must be absent (or else there would be a cycle). Including this absent edge and removing the longer edge would yield a better tree, which forms a contradiction.

10. **(18 POINTS)**

SOLUTION:

$\rightarrow$ **(4pts):** *Subproblems:* For $1 \leq i \leq H$ and $1 \leq j \leq V$, let $C(i,j)$ represent the maximum attainable cost from a rectangle of sheet metal of size $i \times j$.

$\rightarrow$ **(4pts):** Also define function $R()$ as follows:

$$R(i,j) = \begin{cases} \max_k c_k \text{ for all products } k \text{ with } h_k = i \text{ and } v_k = j \\ 0 \text{ if no such product exists} \end{cases}$$

$\rightarrow$ **(4pts):** We initialize the smallest subproblems as follows:

$$C(1,j) = \max\{0, R(1,j)\}$$

$$C(i,1) = \max\{0, R(i,1)\}$$

$\rightarrow$ **(4pts):** *Algorithm and Recursion:* Given both $C()$ and $R()$ above, an $i \times j$ rectangle can be cut in the $(i-1)+(j-1)$ ways shown in the recursion below or used in its entirety. Using the recursion below, we go in increasing order of $i+j$.

$$C(i,j) = \max \begin{cases} \max_{1 \leq k < i}\{C(k,j) + C(i-k,j)\} \\ \max_{1 \leq h < j}\{C(i,h) + C(i,j-h)\} \\ R(i,j) \end{cases}$$

Our solution is then $C(H,V)$.

$\rightarrow$ **(2pts):** *Runtime:* The runtime is rather messy. There are $H \times V$ subproblems, with each subproblem requiring $O(H+V+n)$ time to evaluate. Therefore, runtime is $O(HV(H+V+n))$.

11. **(8 POINTS)**

SOLUTION:

$\rightarrow$ **(2pts):** To show the IES Problem is in class NP-complete, we must show a reduction from a known NP-complete problem.

$\rightarrow$ **(2pts):** A good NP-complete problem to use is the Vertex Cover Problem. Other NP-complete problems may be possible here, too.

$\rightarrow$ **(2pts):** Construct graph $G = (V,E)$ in which V contains all elements of all sets $\{S_1, \ldots, S_n\}$ and E contains edge $\{u,v\}$ if elements u and v are both in the same set.

$\rightarrow$ **(2pts):** In the Vertex Cover Problem, we wish to find budget b vertices that cover (or touch) each edge in E. Here, b is given in the IES Problem.