Name: ________________________________

Algorithms
Fall 2018
FINAL EXAM

This is a 3-hour, closed-notes exam. **Please put away all notes, phones, laptops, etc.** There are twelve pages and six problems. Good luck!

Problem	**Score**
1	/10
2	/20
3	/20
4	/20
5	/15
6	/15
Total	/100

1 Short answer.

(a) Let $T_W(n)$ be the worst-case response time for my algorithm, and let $T_B(n)$ be the best-case runtime for my algorithm. For each statement, give a big-O, big-Ω, or Θ class for one of $T_W(n)$ or $T_B(n)$ that means the same thing as the statement.

i. There's some input to my algorithm that requires at least time n^2

ii. All inputs to my algorithm require at least time $n \lg n$

iii. When run on the worst possible input, my algorithm takes at most time n

(b) Suppose I have two problems, X and Y, and I've shown that $X \leq_P Y$. If I then show that X cannot be solved in polynomial time, what (if anything) do I learn about Y? Give a brief explanation of your answer.

(c) Suppose I show that Graph Coloring $\leq_P$ Max-Flow. What are the implications of this result? Give a brief explanation of your answer.

2 Cliques.

Suppose you have an undirected, unweighted graph $G = (V, E)$. A *k-clique* is a set of k vertices such that each pair of vertices in the set is connected by an edge. For example, any pair of vertices with an edge between them forms a 2-clique. Any "triangle" of vertices forms a 3-clique.

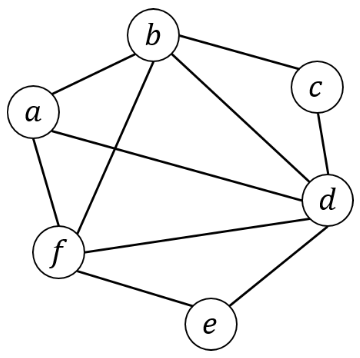
The *k-clique* problem is defined as follows:

Input:

- an undirected, unweighted graph $G = (V, E)$
- a positive integer k

Output: A yes/no answer to the question: does G contain a k-clique?

(a) Here is a graph. Find a 4-clique in this graph.



(b) What does it mean for a problem X to be NP-complete?

(c) Prove that k-clique is NP-complete (hint: use Independent Set).

3 Subsequences.

The *Longest Common Subsequence* problem is defined as follows:

Input: string S, consisting of n characters, and string T, consisting of m characters

Output: the length of the longest sequence of characters that appear in order from left to right (but not necessarily contiguously) in both S and T

For example, suppose $S =$ `algorithm` and $T =$ `great`. Then then longest common subsequence is `GRT`, which has length 3: `alGoRiThm` and `GReaT`.

(a) Let $L(i, j)$ denote the length of the longest common subsequence of the first i characters in S and the first j characters in T (where the first character has index 1). Give a recurrence that expresses $L(i, j)$ in terms of solutions to smaller subproblems.

(b) What does it mean for a problem to have *overlapping subproblems*? How do we use this property when designing a dynamic programming algorithm?

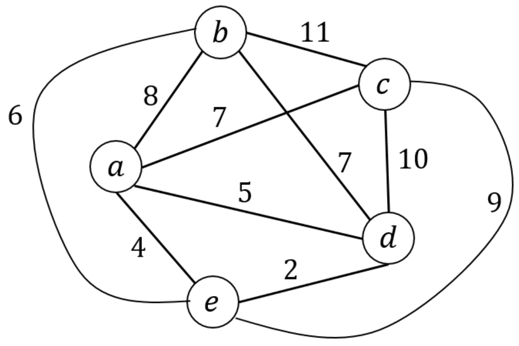
4 Traveling Salesman.

Recall the Traveling Salesman problem: given a complete, undirected, weighted graph and a positive number D, is there a cycle that visits all vertices exactly once, ends where it starts, and has total weight $\leq D$?

And recall the Hamiltonian Cycle problem: given a directed, unweighted graph, is there a cycle that visits all vertices exactly once and ends where it starts?

(a) We said in class that we could prove that TSP is NP-hard by giving a polynomial-time reduction from Hamiltonian Cycle to TSP. We didn't do the reduction in class. Do so now: show that Ham-Cycle $\leq_P$ TSP.

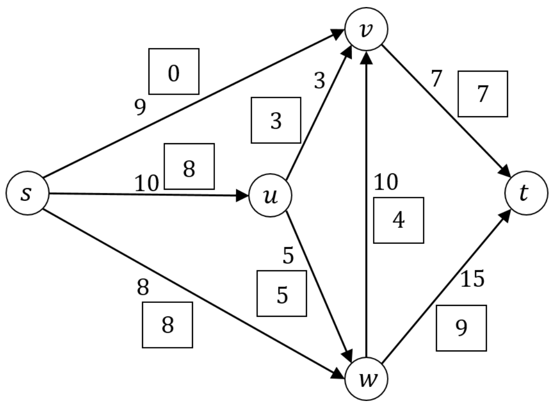
(b) Here's a graph:



Run the TSP approximation algorithm we discussed in class to find a "pretty good" Traveling Salesman tour. (Show the steps that this algorithm takes; from your solution it should be clear that you know what the algorithm does.)

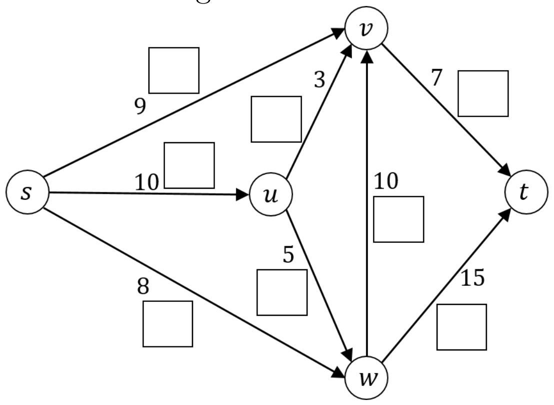
5 Network flow.

Here is a graph, where non-boxed numbers represent edge capacities and boxed numbers represent the current flow along the edge.



(a) Update the flow from s to t by running one iteration of Ford-Fulkerson. (You should show what happens during this iteration, not just the resulting flow; your answer should make clear that you understand what Ford-Fulkerson does.)

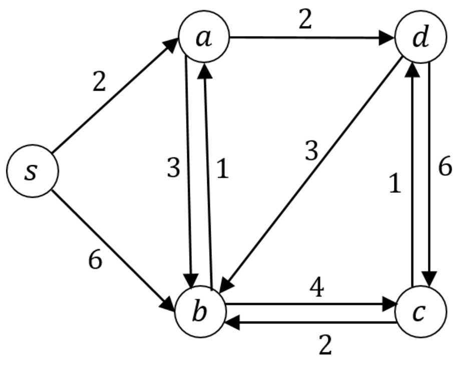
Your resulting flow:



(b) At this point, do you have a max flow? Explain why or why not (your answer should be general: explain how, given *any* graph and a flow in that graph, you can determine if it's a max flow).

6 Shortest paths.

(a) Here's a graph:



Run Dijkstra's algorithm on this graph to find the shortest path from s to every other node in the graph. Show what happens during the execution of the algorithm (from your solution, it should be clear that you understand what steps Dijkstra's algorithm follows).

(b) What algorithmic paradigm does Dijkstra's algorithm use? Why is this paradigm no longer a good choice when the graph contains negative edge weights? (Your answer should discuss what properties a problem should satisfy in order for the paradigm to hold, and why shortest paths with negative edge weights violates one or more of those properties.)

EXTRA CREDIT: For up to 2 points extra credit, write me an algorithms-themed joke.