Final

Name:

TA:

Section Time:

Course Login:

Person on Left:

Person on Right:

Answer all questions. Read them carefully first. Be precise and concise. Write your answers in the space provided. We will provide separate scratch paper. Do NOT turn in the scratch paper!

The exam has a total of 180 points.

Good luck!

1 True/False [2 points each]

Write either True or False to the *left* of the number.

1. If an undirected connected graph has the property that between any two nodes u and v, there is *exactly* one path between u and v, then that graph is a tree.

   True. This ensures that the graph is acyclic. Since it is also connected, it must be a tree.

2. Modular exponentiation is known to take polynomial time. (Modular exponentiation takes numbers a, b, c—all in binary—as input and returns $a^b \pmod{c}$ in binary as output).

   True.

3. Finding all solutions to the equation $x^2 = a \pmod{N}$ is known to be solvable in polynomial time.

   False. We discussed that this is known to be equivalent to factoring.

4. Dijkstra's algorithm will find the correct shortest distances from a given starting node on a directed graph with negative edges provided there are no negative cycles.

   False. (Unmodified) Djikstra is guaranteed to work only when all edge weights are non-negative.

5. Let $T(n)$ be the time taken by some algorithm on inputs of size n. If $T(2n) \leq (T(n))^2$ for all positive n, then the algorithm runs in polynomial time.

   False. Consider $T(n) = 2^n$. Then $T(2n) = 2^{2n} = T(n)^2$, but $T(n)$ is not polynomially bounded.

6. Suppose that the running time $T(n)$ of an algorithm on inputs of size n satisfies the recurrence $T(n) = T(n/2)\log n + O(n)$. Then $T(n) = O(n^c)$ for some finite positive constant c.

   False. The number of sub-problems at the last level is $\Omega\left(\prod_{i=1}^{\log n}(\log n - i)\right) = e^{\Omega(\log n \log\log n)}$, which is not polynomially bounded.

7. Given a problem B, if there exists an NP-complete problem that can be reduced to B in polynomial time, then B is NP-complete.

   False. B may not be in NP itself.

8. If P $\neq$ NP, then any search problem in NP can be reduced to any other problem in NP in polynomial time.

   False. If P $\neq$ NP, we cannot do such reduction from an NP-complete problem to a problem in P.

9. Given any connected undirected graph G, there exists a maximum independent set of G that is also a vertex cover of G.

   False. Consider the triangle graph: the max independent set is of size 1 but the minimum vertex cover is of size 2.

10. Given any NP-complete optimization problem, a b-approximate solution can be found in polynomial time for some positive finite constant b.

    False. There are NP-complete problems which cannot be approximated to any constant factor unless P = NP (such as general TSP).

11. Any acyclic undirected graph is 2-colorable.

    True. Since it is acyclic, it is also bipartite (recall an undirected graph is bipartite if and only if it does not have an odd cycle), and thus it is 2-colorable.

12. Given an undirected, unweighted, connected graph G, suppose we run a DFS on G starting on some node s. We find that the DFS tree has the property that for any vertex v, the path in the DFS tree from s to v is a shortest (fewest number of edges) path from s to v in G. G is a tree.

    True. If the graph had a cycle, then the DFS tree would not have the shortest path for the last node it visits in the cycle.

13. For integers $a, N > 2$, if $a^{N-1} \neq 1 \pmod{N}$ then N is not prime.

    False. Consider $a = N = 3$.

14. For positive distinct integers a and b, if for an $x \neq 0 \pmod{ab}$, $x^{(a-1)(b-1)} \neq 1 \pmod{ab}$ then either a or b is not prime.

    False. Even if a and b are prime, we would have $a^{(a-1)(b-1)} \neq 1 \pmod{ab}$. Consider $a = 2, b = 3$, and $x = 2$.

15. 8 has a multiplicative inverse mod 99.

    True. 8 is co-prime to 99.

16. If the maximum flow problem is not NP-complete, then P $\neq$ NP.

    True. If P = NP, then all non-trivial problems are NP-complete. See also Homework 11 Problem 1.

17. Consider a DFS in a directed graph (which has an edge (u, v)) where pre$(u) = 5$, post$(u) = 8$, and pre$(v) = 3$ and post$(v) = 10$. This graph must have a cycle.

    True. The pre and post numbers imply that (u, v) is a back-edge and hence part of a cycle.

18. The node with the highest post order number in a depth first search of an undirected connected graph is always the starting node.

    True. Since the graph is connected, all vertices are reachable from the start node, and so they will all be visited before the postvisit of the starting node.

19. The node with the highest post order number in a depth first search of a directed graph is in a sink strongly connected component.

    False. It lies in a *source* component. Consider the graph $\{(A, B), (A, C)\}$ with DFS started at A. The vertex with the highest post number is A which is not in a sink strongly connected component.

20. One can always make a directed graph with vertices V strongly connected by adding at most $|V|$ edges.

    True. Just add the edges $(v_1, v_2), (v_2, v_3), \cdots (v_{n-1}, v_n), (v_n, v_1)$ (if not already present).

21. Given two points $\mathbf{v_1}$ and $\mathbf{v_2}$ that each satisfy a set S of linear constraints, the point $\frac{\mathbf{v_1}}{2} + \frac{\mathbf{v_2}}{2}$ also satisfies all the constraints in S.

    True. This follows from the fact that the feasible region is convex. Alternatively, if $\vec{a} \cdot \vec{v_1} \leq b$ and $\vec{a} \cdot \vec{v_2} \leq b$, then adding the inequalities and dividing by 2 gives $\vec{a} \cdot (\frac{\vec{v_1}}{2} + \frac{\vec{v_2}}{2}) \leq b$.

22. Given a linear program on n non-negative variables, with more than n constraints, the linear program has a bounded optimal value.

    False. Consider $x \geq 1$, $x \geq 2$, etc. Then if we want to maximize x, there is no bounded optimal value.

23. The linear program for the maximum flow problem on a graph with integer capacities has an optimal solution where all the variables have integer values.

    True. We saw in class that max-flow with integer capacities always has at least one integral optimum.

24. Consider the 8 node directed graph on the professor and the course staff with edges from older to younger (assuming no ties), where each edge has capacity 1. To clarify, note that there is a directed edge (a, b) from a to b if a is older than b. There is a flow of value 7 from the oldest (the professor) to the youngest.

True. Send one unit from the professor to everyone else. Everyone apart from the youngest person routes this one unit to the youngest person.

25. Chapter 9 suggests that backtracking can be used to solve all problems in NP in polynomial time.

False. This would be blasphemy. More seriously, we do not have any serious candidate approaches for solving NP-complete problems in polynomial time.

26. The cost of a TSP tour with a given set of distances is never less than the cost of a minimum spanning tree of the complete graph where the costs of the edges are the distances in the TSP instance.

True. The TSP tour contains a spanning tree, and hence its cost must be at least that of any MST.

27. There is a graph with 12 nodes which has a vertex cover of size 5 and a matching of size 6.

False. Any vertex cover must include at least one vertex from each edge in any given matching.

28. A valid superposition for two qubits is $\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle$.

False. This is not a valid superposition because the squares of the amplitudes do not sum up to 1.

2 Let's work some examples. [32 points]

No need to justify.

1. What is $5^{49} \mod 35$?

   Solution: 5. We cannot use Euler's theorem directly. So we use Chinese Remaindering. We first observe that $5^{49} = 0 \pmod 5$ and $5^{49} = \left(5^6\right)^8 \cdot 5 = 5 \pmod 7$. Thus, we have to solve $x = 0 \pmod 5$ and $x = 5 \pmod 7$, to which $x = 5 \pmod{35}$ is a solution.

2. What is x, when $8x = 7 \pmod{99}$?

   Solution: 38. Many ways to solve this one:

   Find $8^{-1} = 62 \pmod{99}$ and so $x = 7 * 62 = 434 = 38 \pmod{99}$.

   Multiplying both sides by 13 (which is co-prime to 99), we have $5x = -8 \pmod{99}$. Multiplying both sides by 20 (which is again co-prime to 99), we get $x = 38 \pmod{99}$.

   Notice that it must be that $8x = 99k + 7$ for some integer k. Try values of k from 1 onwards. The first number that is divisible by 8 is 304, which gives $x = 304/8 = 38$.

3. What is the FFT of $[0, 1, 0, 0]$?

   Solution: We need to use $\omega = e^{2\pi i/4} = i$. The FFT is then seen to be simply $[\omega^0, \omega^1, \omega^2, \omega^3] = [1, i, -1, -i]$.

4. If one is to use FFT to multiply two degree-5 polynomials. What value of ω should you use? (Please express it in exponential form.)

   Solution: The product can be a degree 10 polynomial, which can have 11 coefficients. We need to use the next higher power of 2, which is 16. We therefore need $\omega = e^{2\pi i/16}$.

5. Given a positive integer d, what is a set of frequencies for 2^d characters where the optimal prefix code has an associated tree of depth exactly d? (The depth of a tree is the length of the path from the root to the deepest leaf.)

   Solution: If all characters had the same frequency $1/2^d$, then by symmetry, they will all be at leaves of a complete binary tree of depth d.

6. Give an example of a set cover instance with at most 5 sets and 10 elements where the greedy algorithm always produces a sub optimal solution.

   Solution: Suppose there are 6 elements, and the sets are $S_1 = \{1, 2, 3\}$, $S_2 = \{4, 5, 6\}$ and $S_3 = \{2, 3, 4, 5\}$. There is a solution which chooses only the 2 sets S_1 and S_2. However, the greedy solution will first pick S_3 and then also have to pick S_1 and S_2, which is suboptimal.

7. Express the problem of finding an optimal row strategy for the following game as a linear program. The row player wants to maximize the payoff here. $\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$

   Solution:

$$\begin{aligned} \max z \\ z &\leq x + 3y \\ z &\leq 2x + y \\ x + y &= 1 \\ x, y &\geq 0 \end{aligned}$$

8. Describe (give the coordinates of) one vertex of the resulting linear program from the previous problem.

   Solution: $(x, y, z) = (0, 1, 1)$ or $(1, 0, 1)$ or $(2/3, 1/3, 5/3)$

3 Where have I seen this before? [18 points]

These problems should have very short answers given knowledge of the material. You may provide a brief justification, but it is not necessary.

1. Given a directed graph, explain how to find all edges that are not part of a cycle.

   Solution: Find the SCCs. Return all edges that are between two SCCs.

2. We wish to determine the most efficient way to deliver power to a network of cities. Initially, we have n cities that are all disconnected. It costs $c_i \geq 0$ to open up a power plant at city i. It costs $r_{ij} \geq 0$ to build a cable between cities i and j. A city is said to have power if either it has a power plant, or it is connected by a series of cables to some other city with a power plant (in other words, the cables act as undirected edges). Give an efficient method for finding the minimum cost to power all the cities.

   Solution: Consider a graph with the cities at the vertices and with weight r_{ij} on the (undirected) edge between city i and city j. Add a new vertex s and add an edge with weight c_i from s to the ith city, for each i. The minimum spanning tree on this graph gives the best solution (power stations need to be opened at those cities i for which (s, i) is part of the MST.

   The original problem can be seen as finding a collection of disjoint forests of minimum total cost (the fact that the solution has to look like forests follows from the non-negativity of the cable costs), and then an additional cost for opening a power plant at one of the vertices in each component of the forest. The construction of adding the new vertex takes care of both these requirements.

3. We are given a set of n lists of sizes n_i. A merge takes as input two lists. If the lists are of size ℓ and r, the merge takes $\ell + r$ operations to complete. How should we merge the lists into a single list using pairwise merges while minimizing the total number of operations?

   Solution: Suppose we arrange the merge schedule as a binary tree (with the original lists at the leaves and the internal nodes representing the intermediate lists produced). It is then easy to see that the total cost of the operation is precisely the length of the prefix tree encoding if the original lists were characters with frequencies equal to their lengths (this is because the length of each non-leaf list in the tree is simply the sum of the lengths of its children, and the total cost of the operation is the sum of lengths of all lists in the tree, except the one at the leaf). Thus, to minimize the merge cost, we simply need to find the Huffman encoding treating the lists as characters with frequencies equal to their length.

4 How low can you go? [24 points]

1. (Warm up.) Consider an undirected *weighted* connected graph. We wish to find the shortest path between nodes s and t that takes *exactly* K edges (the path can visit each node and edge any number of times). We will give an efficient algorithm for this problem. Let $L(v,k)$ be the length of the shortest path from s to v using k edges. What are the base cases and the recurrence to compute $L(v,k)$?

   Solution: The base case is $L(s,0) = 0$ and $L(v,0) = \infty$ for $v \neq s$. Let w_{uv} denote the length of the edge $\{u,v\}$. The recursion is

   $$L(v,k) = \min\{L(u,k-1) + w_{uv} | u \text{ is a neighbor of } v\}.$$

2. Given a row of n houses that can each be painted red, green, or blue (with a cost $P(i,c)$ for painting house i with color c), devise an algorithm to find a minimum cost coloring of the entire row of houses such that no two adjacent houses are the same color.

   Solution: Let $C(i,c)$ be the minimum possible cost of a valid coloring of the first i houses in which the last house gets color c. We then have the base case:

   $$C(1,c) = P(1,c), \text{ for all colors } c.$$

   The recurrence is

   $$C(i,c) = \min_{c' \neq c}\{C(i-1,c') + P(i,c)\}.$$

   In order to compute the colors, we work backwards from the nth house. To compute the best color c_n of the nth house, we simply set $c_n = \arg min_c C(n,c)$. Then, given the colors $c_{i+1}, c_{i+2}, \ldots c_n$ of houses $i+1$ through n, we compute the color c_i by choosing it to ensure that $C(i+1, c_{i+1}) = C(i,c_i) + P(i+1,c_{i+1})$ (this equation always has a solution for c_i because of the way $C(i+1,\cdot)$ was defined.

   Correctness and runtime: The base cases are trivially correct. Suppose that for for all c and $i \leq k$, $C(i,c)$ has the best possible costs among coloring of the first i houses where the ith house gets color c. Consider an optimal coloring of the first $i+1$ houses with the last house getting color d. The optimal solution then must choose the cheapest solution of cost $C(i,c')$ for $c \neq d$ for the first i houses, which is what the solution here does. This completes the proof by induction. The runtime is $O(n)$ since there are n subproblems each of which takes $O(1)$ time.

3. Given a string s, we wish to do a series of operations to change s into a palindrome. More specifically, we know that adding a character c at any position costs $\texttt{add}(c)$ and deleting a character c at any position costs $\texttt{rem}(c)$ (all these costs are non negative and can change depending on the character). Give an efficient algorithm to determine the minimum cost needed to change s into a palindrome.

Solution: Define $D(i, j)$ to be the minimum cost we need to change the substring $s[i : j]$ to a palindrome. Our answer will just be $D(1, |s|)$. Define $Q(c) = \min(rem(c), add(c))$. We have base cases

$$D(i, i) = 0, \text{ for all } 1 \leq i \leq |s|$$

$$D(i, i+1) = \begin{cases} 0 & \text{if } s_i = s_{i+1} \\ \min(Q(s_i), Q(s_{i+1})) & \text{otherwise} \end{cases}$$

Our recurrence is

$$D(i, j) = \begin{cases} D(i+1, j-1) \text{ if } s_i = s_j \\ \min(D(i+1, j) + Q(s_i), D(i, j-1) + Q(s_j)) \text{ otherwise} \end{cases}$$

Notice because of non negative costs, if the two ends of a substring are equal, we can always just add those in for free. Otherwise, we have to consider either adding s_i to the right end of the string, removing s_i, adding s_j to the beginning, or removing s_j. Thus, our recurrence above takes care of all these four cases, so this shows that this works.
We run this recurrence for all length 1 substrings, then length 2 substrings, and so on until we reach $D(1, |s|)$ at which point, we return our result.

The runtime is $O(n^2)$ since there are $O(n^2)$ problems each taking $O(1)$ time.

5 Half-way there [20 points]

Recall that the knapsack problem takes in a set of n items with profit p_i and size s_i with size bound B. We want to find a subset I of the items which maximizes profit given that the size is less than the bound (we can only use each item once). You may also assume that all $p_i \geq 0$, and $0 < s_i \leq B$

We know a dynamic programming solution exists which unfortunately takes time $O(Bn)$ (exponential in terms of the input), so we create some greedy approximations:

1. **Greedy:** Assume items are sorted according to decreasing $\frac{p_i}{s_i}$ (breaking ties arbitrarily). While we can add items, greedily pick items in the given order.

   Show that this greedy algorithm has no nonzero constant approximation ratio.

   Solution: Consider an item with size 1 and profit 2, and another with size B and profit B. Our greedy algorithm above will take the first item, while the optimal solution is taking the second item, so this algorithm has approximation ratio $\frac{2}{B}$ which can go arbitrarily close to zero.

2. We make one very small adjustment to our greedy algorithm from the previous part

   Almost Greedy: Assume items are sorted according to decreasing $\frac{p_i}{s_i}$ (breaking ties arbitrarily). Greedily add items until we hit an item a_i that is too big (can't fit in the knapsack). Pick the better set of $\{a_1, \ldots, a_{i-1}\}$ and $\{a_i\}$

   Show that this algorithm has an approximation ratio of $\frac{1}{2}$.

   Solution: Let the profit of the first set be S_1 and the profit of the second be S_2. Suppose for a moment, that our knapsack expanded to exactly hold a_i. Now, this must be the optimal solution for knapsack for this size, since we are consider the highest "profit per unit size" ratio for all the items, and we are taking as many as we can (i.e. no other solution will have a higher profit per unit size). Additionally, a knapsack with larger capacity will have a better solution than a knapsack with smaller capacity, thus, we must have $S_1 + S_2 \geq OPT$. This implies that at least one of them is bigger than $\frac{1}{2}OPT$, so we can conclude that this algorithm gives us a $\frac{1}{2}$ approximation ratio.

6 Reductions [30 points]

In this problem, you may use the NP-completeness results described in the book.

1. In the 4D matching problem, one has equal-sized sets of boys, girls, pets and houses, and a set of consistent 4-tuples, each consisting of a boy, a girl, a pet and a house. One then needs to find a subset S of the consistent tuples such that every boy, girl, pet, and house is a member of exactly one tuple in S. Show that this problem is NP-complete.

   Solution: First we must show that the 4D matching problem is in NP, and can be verified in polynomial time. Given a solution of 4-tuples to a 4D matching problem, we have to verify that every boy, girl, pet and house are used exactly once and that within each 4 tuple, all the nodes are strongly connected (signifying the the boy is compatible with the girl, pet and house, etc.). Given the original problem, we can check whether all the required edges within a 4-tuple exist in $O(n)$ time as each of the n 4-tuples has 6 edges to check, and at the same time we can keep track of which nodes we have included in 4-tuples. If we check all n 4-tuples and no node is visited twice, then we have also verified that no node appears twice. Thus we have checked that our solution is correct in polynomial time.

   Then for the reduction from 3D matching to 4D matching: Given any instance of 3D matching with n triplets, which we know to be NP-complete, we add n houses to the problem, with every house being connected to every boy, girl, and pet. The solution to this 4D matching problem will also be a solution to the 3D matching problem if we ignore the houses completely because within each 4-tuple we know that the choice of boy, girl, and pet will all have edges between them from the original 3D matching graph. Additionally, if there is a solution to the 3D matching problem there must also be a solution to the corresponding 4D matching problem as every house can be added to any triple, they will not restrict any solutions. Because the number of houses and edges we are adding to the 3D matching problem are bounded polynomially by n and $3n^2$, the reduction is polynomial.

2. We will produce a reduction from 3SAT to node disjoint paths; given a directed graph $G = (V, E)$ and a set of terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$ find a path connecting each terminal pair where no two paths have a node in common.

   Given a formula ϕ, we make a graph consisting of a pair of terminals (s_x, t_x) for each variable x in ϕ and a pair of terminal (s_c, t_c) for each clause c in ϕ. We further make a node v_ℓ for each literal, ℓ, in the formula (recall a literal is an occurrence of a variable, negated or not, in a clause). We also make a pair of paths between the terminal pair (s_x, t_x) for variable x, as follows. One path goes through all literals of form x and one path goes through all the literals of from $\overline{x}$. Note that for a variable x, s_x and t_x can only be connected using one of the two paths; one of these corresponds to True, and the other corresponds to False.

   Given a clause c, $(\ell_i \vee \ell_j \vee \ell_k)$ where ℓ_i, ℓ_j and ℓ_k are literals, describe which nodes the clause terminals s_c and t_c should be connected to or from so that the resulting graph has a set of node-disjoint paths connecting all the terminal pairs if and only if the formula, ϕ, is satisfiable. (No need to prove correctness for this part.)

   Solution: For the clause c we add the edges from s_c and t_c to every literal that appears in the clause. In this example that would mean the edges (s_c, ℓ_i), (t_c, ℓ_i), (s_c, ℓ_j), (t_c, ℓ_j), (s_c, ℓ_k), and (t_c, ℓ_k). Thus, any path from s_c to t_c must pass through one of the literals that are found in the clause, signifying that that literal was chosen as part of the satisfying assignment. If every clause has at least one of its literals chosen as part of an assignment, then we know the formula has been satisfied. We also know that for each variable, it will be always true or always false because of the paths we constructed for each variable. If the variable uses the path which goes through all occurences of the true literals, then no clause's path has used a node corresponding to that variable's true literal, and we can safely assign that variable to be false without invalidating any clauses.

3. Show that given a graph G and a positive integer k, the problem of finding a connected spanning subgraph where every node has degree at most k is NP-complete. A spanning subgraph of $G = (V, E)$ is another graph $G' = (V, E')$ where $E' \subseteq E$ (notice that the set of vertices is the same).

   Solution: This is a generalization of Rudrata Path. If we set $k = 2$, we will get either a Rudrata Path or a Rudrata Cycle, and it is easy to get a Rudrata Path from a Rudrata Cycle (delete one of the edges). Conversely, if there exists a Rudrata Path, that will be a connected spanning subgraph where every node has degree at most 2.