Final

- **The exam has 7 questions (you can choose one of the last 5 to skip), is worth 100 points, and will last 180 minutes.**
- We indicated how points are allocated to different questions. Not all parts of a problem are weighted equally.
- Read the instructions and the questions carefully first.
- Begin each problem on a new page.
- Be precise and concise.
- The questions start with true/false, then short answer, then algorithm design. The problems may **not** necessarily follow the order of increasing difficulty.
- Good luck!

Part A

1 True/False + short justification

(2 pts per item) Are the following **true** or **false**? **Provide a brief justification (1-3 sentences) for each answer.**

(a) If the weights of all the edges in an undirected graph are increased by 1, then the shortest path from s to every vertex (also called the shortest path tree from s) remains unchanged.

(b) If the weights of all the edges in an undirected graph are increased by 1, then the minimum spanning tree remains unchanged.

(c) There exists a linear program $\min c \cdot x$ subject to $Ax \geq b$ for which there is a solution with objective value 5, and for which the dual linear program has a solution with objective value 10.

(d) If we can reduce a size-n instance of problem A to a size n^2 instance of problem B in $O(n^2)$ time (including preprocessing and postprocessing) and problem B has a linear-time algorithm, then problem A has a quadratic-time algorithm.

For the remaining parts there are *four* possible answers:

- True,
- False
- True iff P = NP
- True iff P $\neq$ NP.

In each case, choose the right one.

(e) Consider s-t connectivity, the problem of deciding if there is a path from s to t in a graph G. There is a polynomial-time reduction from 3-SAT to s-t connectivity.

(f) There is a polynomial-time reduction from s-t connectivity to 3-SAT.

2 Short Answer: Solve 4 out of the 5 subparts

(5 pts per item) Clearly identify which questions you solved. In case no such identification is given, we will check the first 4 questions.

(a) You run the multiplicative weights algorithm on n experts. Recall that you are guaranteed to achieve low regret comparable to the best expert. Suppose you have n competitors, and you happen to find out that on day i, competitor k follows the advice of expert $(i+k) \mod n$. Are you also guaranteed to achieve low regret comparable to your best competitor? If yes, explain why. If no, describe how to modify your algorithm to achieve the same regret bound against your competitors.

(b) Consider the following algorithm for set cover: While there is some uncovered element, repeatedly choose an element x that hasn't been covered yet, and add all sets containing x to the set cover. Suppose that your set system has the property that no element appears in more than d sets. Show that the algorithm described above is a d-approximation algorithm.

(c) Fix a prime m, and recall that $[m] := \{0, 1, 2, \ldots, m-1\}$. Let $h_{a_1,a_2} : [m] \times [m] \to [m]$ be the function $a_1 x_1^2 + a_2 x_2 \mod m$. Is the hash function family $\{h_{a_1,a_2} : a_1, a_2 \in [m]\}$ universal? If yes, briefly justify why. If no, give an example of two inputs $(x_1, x_2) \neq (y_1, y_2)$ such that $h(x_1, x_2) = h(y_1, y_2)$ with probability $> 1/m$.

(d) Suppose we have an algorithm A using $O(\log n)$ bits of memory that streams a n-element list L and outputs a random number k with expected value $\mathbf{E}[k]$ equal to the number of distinct elements in L.

Describe an algorithm using $O(\log n)$ bits of memory that streams an n-element list L_1, then streams an n-element list L_2, and outputs a random number m with expected value $\mathbf{E}[m]$ equal to the number of distinct elements that appear in both L_1 and L_2, i.e., $|\{x : x \in L_1 \text{ and } x \in L_2\}|$.

Example: $L_1 = 5, 3, 5, 1, 1, 3$ has 3 distinct elements, $L_2 = 1, 1, 2, 2, 3, 1, 4, 2, 6$ has 5 distinct elements, while there are only 2 distinct elements, namely $1, 3$ that appear in both L_1 and L_2.

Hint: For any two sets S, T, $|S| + |T| = |S \cap T| + |S \cup T|$.

(e) You want to multiply the two polynomials $f(x) = x$ and $g(x) = x^3$ using FFT. However, you forgot to pad appropriately, and have used vectors of length 4. What would be the resulting product of $f(x) \cdot g(x)$ according to your algorithm? Briefly justify your answer.

Part B : Solve 4 out of the next 5 questions

(17 pts per question) Clearly identify which questions you solved. In case no such identification is given, we will check the first 4 questions.

3 Road Trip Redux

You want to drive from San Francisco to New York City on I-80. Your car holds C gallons of gas, but is a gas guzzler and gets 1 mile per gallon. There are n gas stations along I-80, with gas station i at distance d_i miles from San Francisco (in sorted order), with cost per gallon c_i. You start with a full tank at station 1 in San Francisco, and your goal is to reach gas station n in NYC, while spending as little as possible on gas. You may assume there is some feasible trip, i.e., that $\max_i[d_i - d_{i-1}] \leq C$.

(a) Suppose the ith gas station has the cheapest price (i.e. the smallest value of c_j) among stations $1, \ldots, n-1$. Show that if we spend as little money as possible, then either the ith gas station is the first one we buy gas at or we arrive at the ith gas station with an empty tank. Similarly, argue that either the ith gas station is the last one we buy gas at or we leave the ith gas station with a full tank. You may assume the c_i values are distinct. (Hint: use an exchange argument)

(b) Using part a, give an $O(n^2)$-time divide-and-conquer algorithm to compute the minimum amount of money you can spend on gas. Just the algorithm description and runtime analysis are needed.

4 LP+Dijkstra alternative to Bellman-Ford

Let $G = (V, E)$ be a directed weighted graph with possibly negative edge weights w_{ij} for edge (i, j). Our goal is to efficiently compute a new set of non-negative weights w'_{ij} such that the shortest paths between any s and t remains unchanged under the change of weights — this means that we can use Dijkstra under weights w' to compute the shortest path from s to t.

Specifically, for each vertex i we increase the weight of all outgoing edges by some amount c_i and decrease the weight of all incoming edges by c_i. This means that $w'_{ij} = w_{ij} + c_i - c_j$

(a) Argue that the shortest paths from s to t under w' are the same as the shortest paths from s to t under w. Note that the path lengths may not be the same under w and w'.

(b) We know that the problem of finding shortest paths in a graph is only well-defined if the graph has no negative weight cycles. Show that if there is some choice of $\{c_i\}_{i \in V}$ such that all w'_{ij} are non-negative, then G has no negative weight cycles.

* Note that it is also true that if G has no negative weight cycles then there is some choice of $\{c_i\}_{i \in V}$ such that all w'_{ij} are non-negative. But we are **not** asking you to prove it for this question.

(c) We want to choose the amounts, c_i, so the adjusted weight w'_{ij} of every edge in the modified graph is non-negative. Suppose we also want to modify the weights as little as possible, i.e., we want to minimize the sum of the absolute values of the c_i's. Write this as a linear program.

5 Small Vertex Cover

Recall that $S \subseteq V$ is a vertex cover for graph $G = (V, E)$, if every edge in E has at least one endpoint in S. The vertex cover problem asks on input $G = (V, E)$ and k whether G has a vertex cover of size at most k. While Vertex-Cover is NP-complete, we will give an algorithm for this problem that is efficient when k is small.

(a) Give an algorithm to decide if there's a vertex cover of size 1 or less in $O(|V| + |E|)$ time. Just the algorithm description is needed.

(b) Give a recursive algorithm to decide if there's a vertex cover of size at most k in $O(2^k \cdot (|V| + |E|))$ time. Give the algorithm description, a brief justification, and runtime analysis. (Hint: to solve the problem with parameter k, solve two problems with parameter $k-1$, and apply recursion.)

For half credit you may instead give an algorithm to decide if there's a vertex cover of size at most 2 in $O(|V| + |E|)$-time. Give the algorithm description, a brief justification, and runtime analysis.

6 Is-Different

Consider the following problem called **Is-Different**. You are given two 3-CNF (3-SAT) formulas[1] Φ_1, Φ_2 over the same set of variables $x_1, \ldots, x_n$. You want to decide if there is some assignment of values to $x_1, \ldots, x_n$ such that $\Phi_1 \neq \Phi_2$, i.e. one of the formulas evaluates to true and the other to false.

For example, for $\Phi_1 = (x_1 \vee x_2 \vee x_3)$ and $\Phi_2 = (x_1 \vee x_2 \vee \overline{x_3})$ the assignment $(0, 0, 0)$ causes Φ_1 to be false but Φ_2 to be true, so the answer would be **yes**. However, for $\Phi_1 = (x_1 \vee x_2) \wedge (\overline{x_2})$ and $\Phi_2 = (x_1) \wedge (\overline{x_2})$, we always have $\Phi_1 = \Phi_2$ so the answer is **no**.

(a) Show that Is-Different is in **NP**.

(b) Show that Is-Different is **NP**-hard.

(c) Φ_1 and Φ_2 are called equivalent if they have the same satisfying assignments, i.e., they have the same truth value for all assignments to the variables (in other words, if the answer to Is-Different is no). Consider now the problem of 3-CNF-minimization: Given a 3-CNF formula Φ, find an equivalent 3-CNF formula with the smallest number of clauses.

Show that if 3-CNF-minimization can be solved in polynomial time, then 3-SAT can be decided in polynomial time.[2]

7 Local 3-SAT

Consider the following variant of 3-SAT, called k-local 3-SAT. We are given a 3-SAT instance $\Phi(x_1, x_2, \ldots, x_n)$ with m clauses, with the guarantee that x_i, x_j appear in the same clause only if $|i - j| < k$.

(a) We will give a dynamic programming algorithm for this problem that is efficient when k is small – the target running time is $O(2^k nm)$.

In the j-th iteration of our algorithm we wish to restrict our attention to the variables $x_1, x_2, \ldots, x_j$. Denote by $\Phi_j(x_1, x_2, \ldots, x_j)$ the restriction of $\Phi(x_1, x_2, \ldots, x_n)$ to clauses containing only variables $x_1, x_2, \ldots, x_j$. For example, if $\Phi(x_1, x_2, \ldots, x_4) = (x_1) \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_3 \vee x_4)$ then $\Phi_1(x_1) = (x_1)$ and $\Phi_3(x_1, x_2, x_3) = (x_1) \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \overline{x_3})$.

Define the subproblem $f(j, a_1, \ldots, a_k)$ for every $j \in \{k, \ldots, n\}$, and $a_1, \ldots, a_k$ Boolean. $f(j, a_1, \ldots, a_k)$ is true if $\Phi_j(x_1, x_2, \ldots, x_j)$ has a satisfying assignment with $x_{j-k+1} = a_1, \ldots, x_j = a_k$.

Note that as base cases we can brute force compute $f(k, a_1, \ldots, a_k)$. Give a recurrence relation for solving these subproblems for $j > k$. Analyze the runtime of your recurrence relation. Also describe how to determine if the k-local 3-SAT formula is satisfiable given the solutions to all these subproblems.

(b) Give a polynomial-time reduction from 3-SAT to $\sqrt{n}$-local 3-SAT.

[1] Recall that a 3-CNF formula is the AND of some number of clauses, and each clause is the OR of between 1 to 3 literals, e.g., $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee x_4) \wedge (\overline{x_1})$.

[2] By polynomial time, we mean polynomial in the number of clauses and variables.