CS 170
Spring 2020

Sample Exam.

Final Exam

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

○ You must choose either this option

○ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

Welcome to the CS 170 final!

(a) **There is no negative scoring for this exam.**
(b) All questions in this exam have the same weight.
(c) This tool shows you when you have unsaved questions. Make sure to save all your work before finishing.
(d) Make sure to check your email. We have sent a PDF of the exam for you to use, in case you encounter technical difficulties with this tool. **Only use the exam PDF in your email as a last resort.**

You can complete and submit the name and SID questions before the exam starts.

(a) (1 pt) What is your full name?

(b) (1 pt) What is your student ID number?

1. Exam Problems

(a) **i.** **(1 pt)** Suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Select all that **must** be true.

- ☐ $f(n) = O(g(n))$
- ■ $f(n) = \Omega(g(n))$
- ☐ $f(n) = \Theta(g(n))$
- ☐ None of the other options

ii. **(1 pt)** Suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Select all that **could possibly** be true.

- ☐ $f(n) = O(g(n))$
- ■ $f(n) = \Omega(g(n))$
- ☐ $f(n) = \Theta(g(n))$
- ☐ None of the other options

iii. **(1 pt)** Suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. Select all that **must** be true.

- ☐ $f(n) = O(g(n))$
- ☐ $f(n) = \Omega(g(n))$
- ☐ $f(n) = \Theta(g(n))$
- ■ None of the other options

iv. **(1 pt)** Suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. Select all that **could possibly** be true.

- ■ $f(n) = O(g(n))$
- ■ $f(n) = \Omega(g(n))$
- ■ $f(n) = \Theta(g(n))$
- ☐ None of the other options

v. **(1 pt)** Choose all statements which are true, if $f(n) = n^3$.

- ☐ $f = O(27^{\log_3(\sqrt{n})})$
- ☐ $f = o(10n^3)$
- ■ $f = \omega(n)$
- ■ $f = \Omega(n^3)$
- ☐ None of the other options are true

vi. **(1 pt)** Choose all statements which are **guaranteed** to be true, if $f = O(\log n)$

☐ $f = \Omega(1)$

[✓ $f = O(\log^2 n)$]

[✓ $f = O(n^{0.01})$]

☐ $f = \Theta(\log n)$

☐ None of the other options are true

vii. **(1 pt)** Choose all statements which are **guaranteed** to be true if $f = n(\sin(n) + 2)$

[✓ $f = \Omega(1)$]

[✓ $f = \Theta(n)$]

[✓ $f = O(n^2)$]

☐ None of the other options are true

viii. **(1 pt)** Suppose $f(n) = n^3$ if $n^3 < 2^{20}$ and n otherwise. State the theta bound of f(n) or write 'undefined' if it does not exist.

theta (n)

ix. **(1 pt)** Suppose $f(n)$ and $g(n)$ are both $\Theta(n^d)$ and $f(n) \neq g(n)$. If $h(n) = f(n) - g(n)$ and $h(n) > 0$ for all n, what is $h(n)$? Your answer should be in omega, theta, or O.

O (n^d)

x. **(1 pt)** What is the runtime for the following? Answer must be in Θ.

```
for $i$ from 1 to $n$:
    do $\Theta(3^i)$ work
```

theta (3^n)

xi. **(1 pt)** What is the runtime for the following? Answer must be in Θ.

```
for $i$ from 1 to $n$:
    for $k$ from $i$ to $n$:
        do $\Theta(1)$ work
```

theta (n^2)

(b) **i.** **(1 pt)** We want to multiply the polynomials $A(x) = 1 + 5x + 3x^2 + 7x^3$, and $B(x) = 3 + x$. How many roots of unity should polynomial A be evaluated at to successfully compute the product of A and B?

8

ii. **(1 pt)** We want to multiply the polynomials $A(x) = 1 + 5x + 3x^2 + 7x^3$, and $B(x) = 3 + x$. How many roots of unity should polynomial B be evaluated at to successfully compute the product of A and B?

8

iii. **(1 pt)** Suppose ω is a primitive $2n$-th root of unity. What are the n-th roots of unity in terms of ω? Be sure to describe all nth roots.

1, w^2, w^4, ..., w^{2(n - 1)}

iv. **(1 pt)** Consider Fake Fiveier Transform, which is similar to the Fast Fourier Transform, though each polynomial is instead evaluated at arbitrary values (roots of disunity) naively. What is the runtime of polynomial multiplication using this algorithm, if the polynomials we are multiplying are of degree n?

$\mathcal{O}(n^2)$ **if the numbers are random there is no way to divide and conquer.**

v. **(1 pt)** Recall that a degree n polynomial can be written in its point-value form, by giving its values at n distinct points.

Given two degree n polynomials in point-value form (with the same set of n dinstict points), what is runtime of computing their product in point-value form? Give your answer in Θ notation.

theta (n)

vi. **(1 pt)** Given two arrays $[a_0, a_1, \dots, a_{n-1}]$ and $[b_0, b_1, \dots, b_{n-1}]$, it is possible to compute $\sum_{i=0}^{k} a_i b_{k-i}$ for all $k = 0, 1, \dots, n$ in $\Theta(n \log n)$ time.

[✓ True]

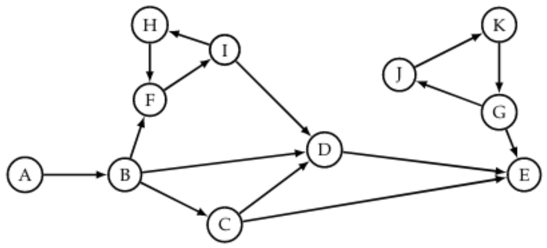
False

(c) **i.** **(1 pt)** Suppose on a directed graph G there exists at least one DFS tree (starting from a fixed vertex s) with at least one forward edge. For all DFS trees on this graph starting from s, there must be at least one forward edge.

○ True

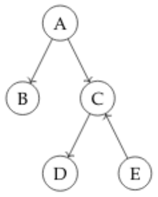
[✓ False]

ii. **(1 pt)** For the following graph, write out the order of a DFS traversal in which the smallest post number is **not** in a sink strongly connected component. For your ordering, only write a vertex when it's prenumber has been assigned.



K G J E A B C D F I H.

iii. **(1 pt)** Consider the following directed graph:



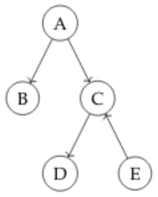
Suppose we add the directed edge (E, D). If we run DFS, (E, D) will be a cross edge.

○ Always

[✓ Sometimes]

○ Never

iv. **(1 pt)** Consider the following directed graph:



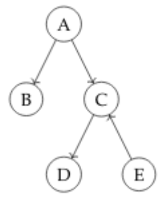
Suppose we add the directed edge (D, B). If we run DFS, (D, B) will be a back edge.

○ Always

○ Sometimes

[✓ Never]

v. (1 pt) Consider the following directed graph:

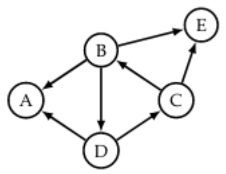


Suppose we add the directed edge (D, A). If we run DFS, (D, A) will be a tree edge.

○ Always

[✓ Sometimes]

○ Never



vi. (1 pt)

We are using DFS to find the strongly connected components in the graph. Calling explore on which of the following vertices will correctly mark **exactly** one SCC as visited?

[✓ A]

☐ B

☐ C

☐ D

[✓ E]

vii. (1 pt) Adding an edge to a DAG creates a cycle.

○ Always

[✓ Sometimes]

○ Never

viii. (1 pt) In a topological ordering of a DAG where every edge has length 1, vertex a comes before vertex b. This means that the length of the longest path that ends at vertex b must be greater than or equal to the longest path that ends at vertex a.

○ True

[✓ False]

ix. (1 pt) If G is a directed graph which has at least 2 SCCs, it is possible to **decrease** the number of SCCs of G by adding additional vertices and/or edges.

[✓ Always]

○ Sometimes

○ Never

x. (1 pt) If G is a directed graph, it is possible to **increase** the number of SCCs of G by adding additional vertices and/or edges.

[✓ Always]
○ Sometimes
○ Never

xi. (1 pt) If G is a directed graph, it is possible to **decrease** the number of SCCs of G by adding a single edge of your choosing.

○ Always
[✓ Sometimes]
○ Never

xii. (1 pt) If G is a directed graph, it is possible to **increase** the number of SCCs of G by adding a single edge of your choosing.

○ Always
○ Sometimes
[✓ Never]

(d) i. (1 pt) In a disjoint sets data structure with union by rank and path compression with n items, the time for any single call to $find(x)$ is at most $O(\log^* n)$ in the worst case.

○ True

[✓ False]

ii. (1 pt) In a disjoint sets data structure with union by rank and path compression, it is possible for the rank of a (non-root) node to be equal to that of one of its ancestors (parents, parents of parents, etc).

○ True

[✓ False]

iii. (1 pt) Consider a disjoint sets data structure with union-by-rank and **no path compression**, with the elements $\{1, 2, 3, 4, 5\}$. Starting with all elements in their own set, we make the calls:

A. $union(1, 2)$
B. $union(2, 3)$

Afterwards, some arbitrary number of union and find operations are performed. Give the tightest possible upper bound on the rank of element 3 after all operations have been performed.

0

iv. (1 pt) In a disjoint sets data structure with no union-by-rank or path compression, what is the maximum height of a tree in the data structure if there are n elements? *Note: a root node has height 0.*

n - 1

(e) **i.** **(1 pt)** Choose all bounds which are **guaranteed** to be true, if $T(n) = 2T(\frac{n}{5}) + O(n)$.

☐ $T(n) = O(n^{\log_5 2})$

☐ $T(n) = \Theta(n^{\log_5 2})$

■ $T(n) = O(n)$

☐ $T(n) = \Theta(n)$

☐ None of the other options

ii. **(1 pt)** Choose all bounds which are **guaranteed** to be true, if $T(n) = 12T(\frac{n}{7}) + O(n)$.

■ $T(n) = O(n^{\log_7(12)})$

☐ $T(n) = \Theta(n^{\log_7(12)})$

☐ $T(n) = O(n)$

☐ $T(n) = \Theta(n)$

☐ None of the other options

iii. **(1 pt)** Choose all bounds which are **guaranteed** to be true, if $T(n) = 25T(\frac{n}{5}) + \Theta(n^2)$.

■ $T(n) = O(n^2 \log n)$

■ $T(n) = \Theta(n^2 \log n)$

■ $T(n) = O(n^3)$

☐ $T(n) = \Theta(n^3)$

☐ None of the other options

iv. **(1 pt)** Give an O bound for the following recurrence: $T(n) = 9T(\frac{n}{3}) + n^2(\log_3 n)$

nn(log n)*(log n)

(f) **i.** **(1 pt)** If a directed graph has exactly one negative cycle, Dijkstra's will always compute the correct shortest paths.

○ True

[✓ False]

ii. **(1 pt)** If a directed graph has negative edges but **no** negative cycles, Dijkstra's will compute the correct shortest paths.

○ Always

[✓ Sometimes]

○ Never

iii. **(1 pt)** Running Dijkstra's on a **tree** with negative edge weights will correctly compute shortest paths.

[✓ Always]

○ Sometimes

○ Never

iv. **(1 pt)** Running Dijkstra's on a DAG with negative edge weights will correctly compute shortest paths.

○ Always

[✓ Sometimes]

○ Never

(g) **i.** **(1 pt)** Consider the following algorithm as an attempt to solve the maximum independent set problem. Each iteration, the algorithm takes the lowest-degree vertex into the solution, removes its neighbors from the graph, and repeats. Which one(s) of the following statements about the algorithm is true?

- ☐ The algorithm does not necessarily produce a valid independent set.
- ■ The algorithm may produce an independent set that is not the maximum one.
- ■ The algorithm may produce the maximum independent set.
- ■ The algorithm always produces a valid independent set.

ii. **(1 pt)** Consider the following algorithm as an attempt to solve the maximum clique problem. Each iteration, the algorithm takes the highest-degree vertex into the solution, removes all vertices that are not its neighbors from the graph, and repeats. Which one(s) of the following statements about the algorithm is true?

- ☐ The algorithm does not necessarily produce a valid clique.
- ■ The algorithm may produce a clique that is not the maximum one.
- ■ The algorithm may produce the maximum clique.
- ■ The algorithm always produces a valid clique.

(h) **i.** **(1 pt)** Sid is in charge of organizing a Pokemon tournament with n players. The tournament is a single-elimination tournament: if players i and j play, the player that loses is out of the tournament and cannot play any more games. There are no ties.

Sid has obtained the following insider information: if players i and j play, then they will score a combined total of $f(i, j) \geq 0$ points in that game, and furthermore as the organizer, he can rig the match so that the player of his choice wins. Sid wishes to find a tournament schedule which maximizes the number of points scored throughout the tournament. For that, he needs to specify what the matches are and which player he chooses to be the winner for each match.

Briefly describe an efficient algorithm to solve this problem.

Let $G = (V, E)$ be the complete graph on n vertices with edge weights $\ell(i, j) = f(i, j)$. Find the maximum spanning tree. Done.

(i) **i.** **(1 pt)** Let G be an undirected weighted graph with positive edge weights, and let T be a MST of G. If every weight w in G is replaced with w^2, T will still be a MST of G.

- ○ Always
- [✓ Sometimes]
- ○ Never

ii. **(1 pt)** Let G be an undirected weighted graph with positive edge weights, and let T be a MST of G. If every weight w in G is replaced with 2^w, T will still be a MST of G.

- [✓ Always]
- ○ Sometimes
- ○ Never

iii. **(1 pt)** The unique heaviest edge across some cut is guaranteed **not** to be in a MST.

- ○ True
- [✓ False]

iv. **(1 pt)** An edge that is the unique lightest edge among all edges incident to a vertex is guaranteed to be in a MST of the graph.

- [✓ True]
- ○ False

(j) **i.** **(1 pt)** Let G_k be the complete graph of k vertices. Edges are undirected and unweighted. What is the value of a cut of G_k with n nodes on the left side of the cut and $k - n$ nodes on the right side of the cut?

n * (k - n)

ii. **(1 pt)** Let G_4 be the complete graph of 4 vertices. Edges are undirected and unweighted. What is the minimum number of edges one has to remove on G_4 so that for all s, t, the value of maximum s-t flow is reduced?

2 edges. If you only remove one edge, there will be a vertex with a min cut of 3 still.

iii. **(1 pt)** In a directed weighted graph, for fixed s, t, removing the edge with the smallest capacity would always reduce the value of maximum s-t flow.

True [✓ False]

iv. **(1 pt)** In a directed weighted graph, for fixed s, t, removing the edge with the largest capacity would always reduce the value of maximum s-t flow.

True [✓ False]

v. **(1 pt)** Deciding if a bipartite graph has a perfect matching can be reduced to computing maximum flow in a directed graph.

[✓ True] False

vi. **(1 pt)** Let C be the min s-t cut value of a directed graph, where each edge has unit capacity. Then increasing each edge capacity by 1 would always increase the value of max s-t flow by exactly C.

[✓ True] False

(k) Zero Sum Games

i. (1 pt) Consider a two-player zero-sum game. If the row player has announced and fixed a pure strategy, there exists an optimal strategy for the column player which is *pure* (always choose the same move).

[✓ True]

False

ii. (1 pt) Consider a two-player zero-sum game. If the column player plays a non-optimal strategy, the row player's expected payoff could be higher than the value of the game.

[✓ True]

False

iii. (1 pt) For each player in a two-player zero-sum game, the optimal strategy is always unique.

True

[✓ False]

iv. (1 pt) The value of a two-player zero-sum game is unique.

[✓ True]

False

v. (1 pt) Let V be the value of a two-player zero-sum game. If the row player plays according to their optimal strategy, it is possible for the column player to play in a way that causes the row player's expected payoff to be less than V.

True

[✓ False]

vi. (1 pt) Specify the entries of the 2x2 payoff matrix of a two-player zero-sum game such that the optimal strategy for the row player is not unique, but the optimal strategy for the column player is:

```
+----+----+
| a  | b  |
+----+----+
| c  | d  |
+----+----+
```

One possible solution: $a = 1, b = 0, c = 1, d = 0$. **Any solution where each column has all the same value.**

vii. **(1 pt)** Consider the following payoff matrix for a zero-sum game (each entry represents the row player's payoff):

```
+----+----+
| 1  | -3 |
+----+----+
| -2 | 5  |
+----+----+
```

Which one of these optimization problems finds the optimal strategy (x_1, x_2) for the **row** player?

○ $\min_{x_1,x_2} \max\{x_1 - 3x_2, -2x_1 + 5x_2\}$

○ $\max_{x_1,x_2} \min\{x_1 - 3x_2, -2x_1 + 5x_2\}$

○ $\min_{x_1,x_2} \max\{x_1 - 2x_2, -3x_1 + 5x_2\}$

[✓ $\max_{x_1,x_2} \min\{x_1 - 2x_2, -3x_1 + 5x_2\}$]

viii. **(1 pt)** Consider the following LP:

$\max z$ such that \$z $\leq 2x - yz \leq -5x + 3y$ $x, y, z \geq 0$

Fill in the optimization problem so that its optimal value is equal to this LP's optimal value:

`(a) (b) { (c), (d) }`

where you may specify either $\max_{x,y}$ or $\min_{x,y}$ for (a), max or min for (b) and write an expression in terms of x and y for (c) and (d).

max_{x,y} min {2x - y, -5x + 3y}

ix. **(1 pt)** In a two-player zero-sum game, suppose both players are restricted to playing *pure* strategies (each must always choose the same move). And recall that if they are allowed to use mixed strategies, the optimal strategies can be found via primal and dual LPs. Explain how to modify the LP to solve for the optimal pure strategies. You may use non-linear or integral constraints.

Change $x_i \geq 0$ **to** $x_i \in \{0, 1\}$

x. **(1 pt)** In a two-player zero-sum game, suppose both players are restricted to playing *pure* strategies (each must always choose the same move). Strong duality (still) always holds in this situation, i.e., the payoffs of the row and column player still coincide, when both play their optimal pure strategies.

○ True

[✓ False]

xi. **(1 pt)** Consider the following zero sum game. (Each entry represents the row player's payoff. The row player is trying to maximize the payoff.)

```
+---+----+
| 3 | -1 |
+---+----+
| 5 | 5  |
+---+----+
```

If the row player plays (3/4, 1/4) and the column player plays optimally, What is the expected payoff of this game?

1/2

xii. **(1 pt)** Consider the following zero sum game. (Each entry represents the row player's payoff. The row player is trying to maximize the payoff.)

```
+---+----+
| 3 | -1 |
+---+----+
| 5 | 5  |
+---+----+
```

The expected payoff is positive if both players play optimally.

[✓ True]

False

xiii. **(1 pt)** Specify the values for x and y in the following payoff matrix of the row player in a zero-sum game such that row player has a dominating strategy (a move that achieves equal or higher payoff than any other one, regardless of the strategy of the other player), but column player does not.

```
+---+---+---+
| 2 | x | 3 |
+---+---+---+
| 3 | 1 | 4 |
+---+---+---+
| y | 2 | 1 |
+---+---+---+
```

According to your values of x, y, what is row's dominating strategy (row number)?

Wasn't possible. Free point for this problem.

(l) **i.** **(1 pt)** In the experts problem with n experts and losses on each day in $[0,1]$, what is the tightest bound the multiplicative weights algorithm guarantees about your total loss over T days?

T

ii. **(1 pt)** In the experts problem, even if the losses can be infinite, the regret bound of the multiplicative weights update algorithm still holds.

○ True

[✓ False]

iii. **(1 pt)** In the experts problem, if we knew all losses of all experts on all days in advance (before day 0), we may be able to incur less total loss (over T days) than the offline optimum. (Recall that the offline optimum is defined as the total loss incurred by the best expert.)

[✓ True]

○ False

iv. **(1 pt)** In the experts problem, there always exists a strategy that chooses the same expert on every day (and no other experts) that achieves the offline optimum total loss.

[✓ True]

○ False

v. **(1 pt)** We are running the multiplicative weights algorithm with 20 experts and every loss is in $[0,1]$. Let's say we run the algorithm for 80 days, we believe the best expert has a total cost 15 over the 80 days, and epsilon is $\frac{1}{4}$. What is the maximum that the total loss incurred by the algorithm can be? Provide your answer in terms of a mathematical expression.

total regret $=$ **loss of algorithm - offline optimum** $\leq \epsilon T + \frac{\ln(n)}{\epsilon}$**.** \

$$\textbf{loss of algorithm} - 15 \leq \frac{1}{4}(80) + \frac{\ln(20)}{\frac{1}{4}}$$

\

$$\textbf{loss of algorithm} \leq 20 + 4\ln(20) + 15$$

\ **The maximum loss is roughly 46.98.**

vi. **(1 pt)** In the experts problem, multiplicative weights update (MWU) algorithm and offline optimum can incur the same total loss over T days.

[✓ True]

○ False

(m) Hashing

i. (1 pt) Consider the hash family $\mathcal{H} = \{h_1, h_2\}$, where $h_1(x) = 1$ and $h_2(x) = x$ for $x \in \{1, \ldots, n\}$. That is, we are hashing n values to n buckets. Instead of choosing a hash function uniformly at random from the hash family, we'll choose h_1 with $\frac{1}{n}$ probability and h_2 with $\frac{n-1}{n}$ probability. For any pair of distinct elements $x, y \in \{1, \ldots, n\}$ is the probability that $h(x) = h(y)$ bounded by $1/n$?

[✓ Yes]

○ No

ii. (1 pt) Let $\mathcal{H}$ be a universal family of hash functions mapping a set A to a set B. Consider $a, b \in A$ and $a \neq b$. Give an upper bound on the number of functions h in $\mathcal{H}$ s.t. $h(a) = h(b)$ is at most:

○ $\frac{|A|}{|B|}$

○ $|\mathcal{H}| \cdot \frac{|A|}{|B|}$

[✓ $\frac{|\mathcal{H}|}{|B|}$.]

○ $\frac{|B| \cdot |A|}{|\mathcal{H}|}$

iii. (1 pt) Let n be a multiple of 3. Consider the hash family $\mathcal{H} = \{h_1, h_2\}$, where $h_1(x) = x$ and $h_2(x) = x \mod \frac{n}{3}$ for $x \in [n]$. Is this hash family universal?

○ Yes

[✓ No]

iv. (1 pt) If x and y are distinct elements from a universe U, and $h : U \to \{0, 1\}^k$ is a hash function drawn from a universal hash family $\mathcal{H}$. Select all of the following that are upper bounds on the probability that $h(x) = h(y)$?

[✓ $1/k$]

[✓ $1/2^{k-1}$]

[✓ $1/\log k$]

☐ $1/2^{k+1}$

v. (1 pt) Let $\mathcal{H}$ be a universal hash family of hash functions mapping U to $[m]$. Let S be any subset of U such that $|S| = n$. For any k in S, select all of the following which are upper bounds on the expected number of collisions k has with elements in S for h sampled uniformly from $\mathcal{H}$.

[✓ $\frac{n-1}{m}$]

[✓ $\frac{n}{m}$]

☐ $\frac{n-2}{m}$

[✓ $\frac{n+1}{m}$]

vi. (1 pt) Let S be the set of p roots of unity where p is a fixed prime. Consider the hash family $\mathcal{H} = \{h_{a_1,a_2} : S^2 \to S, a_1, a_2 \in \{0, 1, \ldots, p-1\}\}$, where $h_{a_1,a_2}(e^{2\pi i \frac{x_1}{p}}, e^{2\pi i \frac{x_2}{p}}) = (e^{2\pi i \frac{x_1}{p}})^{a_1} \cdot (e^{2\pi i \frac{x_2}{p}})^{a_2}$.

Is this hash family universal?

[✓ Yes]

○ No

(n) Lower Bounds

i. (1 pt) Choose which of the following are necessarily true if $P = NP$?

- [✓ ■] There exists a polynomial-time reduction from 3SAT to feasibility of linear programs.
- ☐ There do not exist any problems that are not in NP.
- ☐ There exists an n^2 time algorithm for the Traveling Salesman Problem (TSP).
- ☐ None of the other options.

ii. (1 pt) Let A be a decision problem. Suppose that given any instance I of the problem A, we can, in polynomial time, construct a 3SAT formula that is satisfiable if and only if the solution to I is YES. Which one of the following must be true?

- ○ A is NP-hard.
- ○ A is NP-complete.
- [✓ ●] A reduces to 3SAT.
- ○ 3SAT reduces to A.

iii. (1 pt) Let n be an even number. In the fantasy of Sosiland, people believe that it is NP-Hard to find a cut of size at least 0.91 of the maximum cut. Under this belief, show that it is also NP-Hard to find a balanced cut of size at least 0.91 of the Balanced Max Cut.

Recall that in the Max Cut problem, for a given graph $G = (V, E)$, you are asked to find a set $S \subseteq [n]$ that maximizes the number of edges between S and $V \setminus S$. In the Balanced Max Cut problem, you are asked to find S of size $\frac{n}{2}$ that maximizes the number of edges between S and $V \setminus S$.

Take the input for Max Cut, and duplicate it.

iv. (1 pt) Given a positive integer x, we wish to give an algorithm that returns 2^x which runs in polynomial time in the size of our input. Is this possible?

- ○ Yes
- [✓ ●] No

v. (1 pt) Suppose we have an algorithm for solving a decision problem Q by applying a polynomial-time reduction to a decision problem Q' and then running a procedure A that solves Q' in time $\Theta(ne^n)$. Assume that the reduction preserves the solution and algorithm A is correct. Which one of the following is definitely true about Q?

- ○ The problem is NP-hard.
- ○ The problem is NP-complete.
- ○ The problem is in NP.
- ○ The problem is in P.
- [✓ ●] None of the other options.

(o) Complexity

i. (1 pt) Recall that a path where no vertex is visited more than once is called a *simple* path. Also, recall that the Hamiltonian path problem asks if an undirected graph contains a path that visits every vertex exactly once, and that this problem is NP-complete. Which one of the following statements is *false*?

☐ Computing the longest (simple) path in an unweighted undirected graph is NP-hard.

☐ The Hamiltonian cycle problem, which asks if an undirected graph contains a cycle that visits every vertex exactly once, is NP-hard.

☐ Computing the longest (simple) path in an unweighted DAG is solvable in polynomial time.

[✓ Computing a path that visits every edge exactly once (allowing revisiting vertices) in an undirected graph is NP-hard.]

ii. (1 pt) If $P \neq NP$, select all of the following problems that are solvable in polynomial time?

☐ 170-SAT

☐ Compute the longest path in an unweighted undirected graph

[✓ Compute the longest common subsequence of 3 binary strings]

☐ Decide if an (undirected) graph is 170-colorable

iii. (1 pt) Given a 0-1 matrix A of size n by n, we call an index set $S \subseteq [n]$ *dense* (with respect to A) if for all $i \in S, j \in S$, we have $A_{ij} = 1$. Given a matrix A and positive integer t, the *Dense Subset* problem asks if there exists a dense index set (with respect to A) of size at least t. Assuming $P \neq NP$, which one of the following statements is true?

☐ The Dense Subset problem can be solved in $O(n)$ time.

☐ The Dense Subset problem can be solved in time polynomial in n, but not in $O(n)$ time.

[✓ The Dense Subset problem is NP-Complete.]

☐ The Dense Subset problem is not in NP.

iv. (1 pt) All problems in NP can be solved in polynomial space.

[✓ True]

○ False

v. (1 pt) Recall that a path where no vertex is visited more than once is called a *simple path*. Consider the following problem: ''Given a weighted graph G, and vertices s and t, find a simple path from s to t of length greater than k." This problem is NP-complete.

[✓ True]

○ False

vi. (1 pt) Given a graph G, and a vertex s and t. Find a path from s to t of length greater than k (vertex and edge repetitions are allowed). This problem is NP-complete.

○ True

[✓ False]

vii. **(1 pt)** A problem X is NP-complete if there is a polynomial time reduction from every problem in NP to X.

○ True

[✓ False]

viii. **(1 pt)** For a given problem X, if there is a polynomial time reduction from X to every problem in NP, then X is NP-hard.

○ True

[✓ False]

ix. **(1 pt)** We know Problem X is NP-hard, and wish to prove Problem Y is also NP-hard. Giving a polynomial time reduction from Problem __ to Problem __ proves that Y is indeed NP-hard. (Fill in X or Y in each blank.)

○ Problem X to Problem X

[✓ Problem X to Problem Y]

○ Problem Y to Problem X

○ Problem Y to Problem Y

x. **(1 pt)** X is NP-complete if there is a polynomial time reduction from 3SAT to X.

○ True

[✓ False]

xi. **(1 pt)** There is a polynomial time reduction from the problem of finding a shortest path in a graph to Maximum Set Cover.

[✓ True]

○ False

xii. **(1 pt)** Let A and B be two problems in a complexity class $\mathcal{C}$. If there exists a polynomial time reduction from A to B, then there exists a polynomial time reduction from B to A.

○ True

[✓ False]

(p) Approximation

i. (1 pt) Recall that in the Trident Bubble Gum problem, we are given an undirected graph G, and wish to output the minimum-sized set of vertices S such that every vertex in G is either in S or adjacent to a vertex in the set. In this problem, we call a vertex '"covered" by S if it is in S or has a neighbor in S. Now consider the following algorithm that takes in a graph and finds a set S:

A. Initialize $S := \{\}$.
B. Add the vertex of highest degree in G to the set.
C. While there is a vertex not covered by S, add the vertex with the most uncovered neighbors to S.

What is the smallest approximation ratio that this algorithm can guarantee?

☐ 2

[✓ $\ln n$]

☐ $\sqrt{n}$

☐ Unbounded

ii. (1 pt) Consider the following LP relaxation for the minimum vertex cover problem, which, given an unweighted undirected graph $G = (V, E)$, asks for the minimum set of vertices that includes at least one endpoint of every edge of G.

$$\begin{aligned} \min \quad & \sum_{u \in V} x_u \\ \text{such that} \quad & x_u + x_v \geq 1 \text{ for all } (u, v) \in E \\ & x_u \geq 0 \text{ for all } u \in V \end{aligned}$$

We would like to approximate the vertex cover problem. We first solve the LP and obtain a set of fractional values $\{x_u\}_{u \in V}$, one for each vertex. Then we take u into our solution if and only if $x_u \geq 1/2$. Which one of the following statements is true about the algorithm?

☐ The algorithm does not necessarily produce a valid vertex cover.

[✓ The algorithm always produces a valid vertex cover, and the smallest approximation factor it guarantees is 2.]

☐ The algorithm always produces a valid vertex cover, and the smallest approximation factor it guarantees is $\ln n$.

☐ The algorithm always produces a valid vertex cover, but the approximation factor can be unbounded.

iii. (1 pt) Consider the following LP relaxation for the max cut problem, which, given a connected, unweighted, and undirected graph $G = (V, E)$ on $n \geq 2$ vertices, asks for set of vertices $S \subseteq V$ that maximizes the number of edges between S and $V \setminus S$.

$$\begin{aligned} \max \quad & \frac{1}{|E|} \sum_{(u,v) \in E} y_{(u,v)} \\ \text{such that} \quad & x_u + x_v \geq y_{(u,v)} \text{ for all } (u,v) \in E \\ & 2 - x_u - x_v \geq y_{(u,v)} \text{ for all } (u,v) \in E \\ & 1 \geq x_u \geq 0 \text{ for all } u \in V \\ & 1 \geq y_{(u,v)} \geq 0 \text{ for all } (u,v) \in E \end{aligned}$$

We would like to approximate the max cut problem. Let's say we solve the LP and obtain a set of possibly fractional values $\{x_u\}_{u \in V}$. Consider the following two algorithms to obtain a set of vertices S:

A. **Algorithm 1.** Let S be the set of all vertices u such that $x_u \geq 1/2$.
B. **Algorithm 2.** Choose S randomly by placing each u in S independently with probability x_u.

Which one of the following statements are true about the algorithms? For any S, we define its cut value to be $\frac{E(S,\bar{S})}{|E|}$, where $\bar{S} = V \setminus S$.

☐ Algorithm 1 always produces the maximum cut.

☐ There exists a sequence of graphs where the true max cut value is exactly $\frac{1}{2}$ while the objective value of the LP is 1.

[✓] There exists a graph and an optimal LP solution $\{x_u^*\}_{u \in V}$ where the true max cut value is equal to the cut value given by Algorithm 1.

[✓] On the complete graph on 3 vertices, the expected cut value produced by Algorithm 2 is strictly greater than the cut value produced by Algorithm 1.

iv. (1 pt) We would like a $\frac{1}{2}$-approximation for the maximum independent set problem. Recall that S is a vertex cover iff $V \setminus S$ is an independent set and that we have an efficient 2-approximation greedy algorithm for the minimum vertex cover problem. Suppose we are given graph G as input. Now consider the following algorithm:

A. We run our 2-approximate greedy algorithm for Minimum Vertex Cover on G and obtain S.
B. We output $V \setminus S$.

Is the above algorithm a valid $\frac{1}{2}$-approximation for the maximum independent set problem?

○ Yes.

[✓ No.]

v. (1 pt) Given a graph G on n vertices where we are promised that the minimum vertex cover has size at most $n/3$, we wish to output a $\frac{1}{3}$-approximation of the maximum independent set. Recall that S is a vertex cover iff $V \setminus S$ is an independent set and that we have an efficient 2-approximation greedy algorithm for the minimum vertex cover problem. Suppose we are given graph G as input. Now consider the following algorithm:

A. We run our 2-approximate greedy algorithm for Minimum Vertex Cover on G and obtain S.
B. We output $V \setminus S$.

Is the above algorithm a valid $\frac{1}{3}$-approximation for the maximum independent set problem?

[✓ Yes.]

○ No.

vi. (1 pt) Consider the following algorithm for a knapsack problem of n items with values: $p_1, \dots, p_n$, weights: $w_1, \dots, w_n$, and a budget for total weight of W. Assume that for all i in $[n]$, $w_i \le W$.

A. Define ratio $r_i = \frac{p_i}{w_i}, \forall i \in [n]$.
B. Greedily pick items with the highest ratio until picking the next item exceeds budget W. Call this next item k.
C. Compute total price of the items selected by the greedy algorithm, p_G.
D. Return $\arg\max\{p_G, p_k\}$.

What's the highest approximation ratio guaranteed by the output? (Note that this is a maximization problem so the approximation ratio is at most 1.)

$\frac{1}{2}$**, as OPT** $\le p_G + p_k$**.**

vii. (1 pt) Given any NP-Complete problem, we have an α-approximation algorithm in polynomial time for some constant $\alpha > 0$.

○ True

[✓ ● False]

(q) Randomized Algorithms

i. (1 pt) Given G, a bipartite graph promised to contain a perfect matching, suppose algorithm A finds a perfect matching with probability $\frac{1}{2}$. Consider algorithm B, which runs A independently T times, and returns the output of an arbitrary run if one of the T runs produces an output, and returns ''failure" otherwise.

How large should T be to guarantee that the probability that algorithm B outputs ''failure" is less than $\delta = 0.05$?

$T \geq \log \frac{1}{0.05} = \log_2 20$, **so** $T \geq 5$.

ii. (1 pt) Suppose you have a Las Vegas algorithm **A** that solves a computational problem on instances of size n with expected running time $T(n)$.

Describe how to get a Monte Carlo algorithm **B** for the same problem which runs in time at most $2T(n)$, such that on every input **B** outputs a correct output with probability at least $1/2$, and otherwise outputs ''failure" (it never returns a wrong answer).

We simply run algorithm {A}. If it returns an answer before time $2T(n)$**, we return the answer, which is correct by assumption. Otherwise we cut stop the algorithm at time** $2T(n)$**, and declare "failure". By Markov's inequality, wp** $\geq 1/2$ **{A} will terminate with the correct answer in time at most** $2T(n)$**.**

iii. (1 pt) Suppose you have a Monte Carlo algorithm **B** that returns a correct answer with probability at least $1/2$ on every input or otherwise it returns ''failure", and suppose that its running time on an input of size n is at most $T(n)$.

Construct a Las Vegas algorithm **C** from it that runs in expected time at most $2T(n)$.

We just run algorithm {B} again and again, with independent random choices in each run, until it returns an answer (not "FAIL"). In this case, we always get the right answer at the end. Since each run will success with probability $\frac{1}{2}$**, and the runs are independent, on expectation we need to run {B} twice to get the answer, thus the expected running time is** $2T(n)$**.**

iv. (1 pt) Consider a probabilistic test for whether $AB = C$. Given three $n \times n$ matrices A, B, C, we want to verify if $AB = C$. The algorithm chooses random vectors x, y uniformly at random from $\{0,1\}^n$, and computes $y^T ABx$, $y^T Cx$. If they are not the same, the algorithm declares $AB \neq C$, otherwise it declares $AB = C$. Which of the following are valid upper bounds of the error probability of this test.

- ☐ 0
- [✓ 1]
- [✓ 3/4]
- ☐ 1/2

v. (1 pt) Let **X** be an integer valued random variable. Then $\mathbf{E}[\mathbf{X}] = \sum_{i\geq 1} \Pr[\mathbf{X} \geq i]$.

- ○ True
- [✓ False]

vi. (1 pt) Let $\mathbf{X}$ be a positive integer valued random variable. Then $\mathbf{E}[\mathbf{X}] = \sum_{i \geq 1} \Pr[\mathbf{X} \geq i]$.

[✓ True]

False

(r) Streaming

i. (1 pt) Consider a stream of m elements, all of which are in $[n]$, and further assume that each element in $[n]$ occurs at least once in the stream. Consider the following algorithm to sample an element from the stream: using a random hash function as in the FM algorithm $h\colon [n] \to [0, 1]$, store the element with minimum hash value in the stream. Take this element as your sample.

Give a big-O bound on the space required to run this algorithm (ignoring the space needed to read each element in the stream and the space needed to store the hash function h)?

O(log n)

ii. (1 pt) Consider a stream of m elements, all of which are in $[n]$, and further assume that each element in $[n]$ occurs at least once in the stream. Consider the following algorithm to sample an element from the stream: using a random hash function as in the FM algorithm $h\colon [n] \to [0, 1]$, store the element with minimum hash value in the stream. Take this element as your sample. Let $i \in [n]$.

What is the probability that an element with value i is selected?

1/n

iii. (1 pt) Consider a stream of elements in $[n]$, where each element shows up at most once. At the end of the stream, we are given a query $q \in [n]$ and asked if q appeared in the stream. To solve this problem, we create a bit array A of length m, initialized to be all 0, and sample k independent random hash functions $h_1, \ldots, h_k$ where each $h_i : [n] \to [m]$.

A. During the stream, when an element $e \in [n]$ appears, we set the $h_i(e)$th bit of the array A to 1, for all i.
B. To answer the query, we simply check the $h_i(q)$th bit of the array for each i, and claim that q appeared in the stream if and only if all these bits are 1.

Note that except at the initialization step the algorithm never sets a bit to 0. If q appeared in the stream, the algorithm will always correctly claim that it did.

[✓ True]

False

iv. (1 pt) Consider a stream of elements in $[n]$, where each element shows up at most once. At the end of the stream, we are given a query $q \in [n]$ and asked if q appeared in the stream. To solve this problem, we create a bit array A of length m, initialized to be all 0, and sample k independent random hash functions $h_1, \ldots, h_k$ where each $h_i : [n] \to [m]$.

A. During the stream, when an element $e \in [n]$ appears, we set the $h_i(e)$th bit of the array A to 1, for all i.
B. To answer the query, we simply check the $h_i(q)$th bit of the array for each i, and claim that q appeared in the stream if and only if all these bits are 1.

Note that except at the initialization step the algorithm never sets a bit to 0. For fixed i in $[k]$, element e in $[n]$ and ℓ in $[m]$, what is the probability that the $h_i(e) \neq \ell$?

● [✓ $1 - 1/m$]

○ $(1 - 1/m)^k$

○ $1 - (1 - 1/m)^{k^2}$

○ $\left(1 - (1 - 1/m)^{k \log k}\right)^k$

v. (1 pt) Consider a stream of elements in $[n]$, where each element shows up at most once. At the end of the stream, we are given a query $q \in [n]$ and asked if q appeared in the stream. To solve this problem, we create a bit array A of length m, initialized to be all 0, and sample k independent random hash functions $h_1, \ldots, h_k$ where each $h_i : [n] \to [m]$.

A. During the stream, when an element $e \in [n]$ appears, we set the $h_i(e)$th bit of the array A to 1, for all i.
B. To answer the query, we simply check the $h_i(q)$th bit of the array for each i, and claim that q appeared in the stream if and only if all these bits are 1.

Note that except at the initialization step the algorithm never sets a bit to 0. For element e in $[n]$ and ℓ in $[m]$, what is the probability that for all $i \in [k]$, $h_i(e) \neq \ell$?

○ $1 - 1/m$

● [✓ $(1 - 1/m)^k$]

○ $1 - (1 - 1/m)^{k^2}$

○ $\left(1 - (1 - 1/m)^{k \log k}\right)^k$

vi. (1 pt) Consider a stream of elements in $[n]$, where each element shows up at most once. At the end of the stream, we are given a query $q \in [n]$ and asked if q appeared in the stream. To solve this problem, we create a bit array A of length m, initialized to be all 0, and sample k independent random hash functions $h_1, \ldots, h_k$ where each $h_i : [n] \to [m]$.

A. During the stream, when an element $e \in [n]$ appears, we set the $h_i(e)$th bit of the array A to 1, for all i.
B. To answer the query, we simply check the $h_i(q)$th bit of the array for each i, and claim that q appeared in the stream if and only if all these bits are 1.

Note that except at the initialization step the algorithm never sets a bit to 0. Let ℓ be a fixed number in $[m]$. If C distinct elements in $[n]$ have appeared in the stream what is the probability that the ℓth bit is 1?

○ $1 - 1/m$

○ $(1 - 1/m)^k$

[✓ $1 - (1 - 1/m)^{Ck}$]

○ $\left(1 - (1 - 1/m)^{Ck}\right)^k$

vii. (1 pt) Consider a stream of elements in $[n]$, where each element shows up at most once. At the end of the stream, we are given a query $q \in [n]$ and asked if q appeared in the stream. To solve this problem, we create a bit array A of length m, initialized to be all 0, and sample k independent random hash functions $h_1, \ldots, h_k$ where each $h_i : [n] \to [m]$.

A. During the stream, when an element $e \in [n]$ appears, we set the $h_i(e)$th bit of the array A to 1, for all i.
B. To answer the query, we simply check the $h_i(q)$th bit of the array for each i, and claim that q appeared in the stream if and only if all these bits are 1.

Note that except at the initialization step the algorithm never sets a bit to 0. Let C be the number of distinct elements that appeared in the stream and let $q \in [n]$ be an element that did not appear in the stream. Upon being queried q, what is the probability that the algorithm erroneously claims that q appeared in the stream even though it did not?

○ $1 - 1/m$

○ $(1 - 1/m)^k$

○ $1 - (1 - 1/m)^{Ck}$

[✓ $\left(1 - (1 - 1/m)^{Ck}\right)^k$]

viii. (1 pt) Imagine we implement the FM algorithm for distinct elements from class, but accidentally store the maximum hash value instead of the minimum. How can we make a simple modification to estimate the number of distinct elements from the maximum value?

Max val can be interpreted as a min value by taking 1-max. Then apply usual logic/math.

ix. (1 pt) Rishi receives a length-n stream of 0's, 1's, and 2's. Let L be the product of all elements of the stream. He wishes to output the exact value of $\log_2 L$, where we set it to -1 if $L = 0$.

Describe a simple algorithm to do so in $O(\log n)$ space.

Just count the number of 2's. If we encounter a zero, set counter to negative one and don't change. If we encounter a one, ignore. If we encounter 2, increment. Note that it does not suffice to keep track of the entire product as that would require O(n) memory.

x. (1 pt) We wish to estimate the number of distinct elements in a stream and modify the FM algorithm from class in the following way: we keep track of the average hash value of all elements that appeared in the stream instead of the minimum. Let t be the value stored by the algorithm.

Does there exist a choice of constants a and b such that $\mathbf{E}[at + b]$ is equal to the number of distinct elements in the stream?

Yes. [✓ No.]

xi. (1 pt) Recall Morris's algorithm to estimate n, the length of a stream, in $O(\log \log n)$ space from lecture: 1. Initialize $X = 0$. 2. For each element in the stream, increment X with probability $\frac{1}{2^X}$. 3. For a query output $\widetilde{n} = 2^X - 1$. Suppose your only source of randomness is a ''fair coin", i.e., a random bit that is 1 with probability $\frac{1}{2}$ and 0 with probability $\frac{1}{2}$. State how to implement the second step of the above algorithm in $O(\log X)$ space.

Let ℓ be a counter initially set to X. While $\ell > 0$, sample a uniform bit $\boldsymbol{b}$ and if $\boldsymbol{b} = 0$, set $\ell = -1$, otherwise decrement ℓ by 1. If $\ell = 0$, increment X, and if $\ell = -1$, do not increment X.

(s) **i. (1 pt)** Consider the recurrence

$$M[a, b] = \max(a \cdot M[a-1, b-1], b \cdot M[a-2, b-2])$$

where $0 \leq a \leq n, 0 \leq b \leq n$. Let $M[0, i] = M[i, 0] = 1$ for all $0 \leq i \leq n$, and let $M[i, j] = 0$ if either i or j are negative. Give the tightest possible asymptotic bound on the runtime to compute $M[n, n]$ using the recurrence.

Θ(n) as we only care about diagonal entries

ii. (1 pt) Let g be some integer valued function with a lookup table given to you, so it takes constant time to compute. Consider the recurrence

$$M[a, b, c] = \max(M[a-1, b, c-1] + g(a-1, b, c-1), M[a-2, b, c-1] + g(a-2, b, c-1))$$

where $0 \leq a, b, c, \leq n$. Let $M[i, 0, 0] = M[0, i, 0] = [0, 0, i] = 1$ for all $0 \leq i \leq n$, and $M[i, j, k] = 0$ if any of i, j, k are negative. Give the tightest possible asymptotic bound on the runtime to compute $M[n, n, n]$ using the recurrence.

$\Theta(n \log^2(n))$

iii. (1 pt) Let g be some integer valued function with a lookup table given to you, so it takes constant time to compute. Consider the recurrence $M[a, b] = \min_{0 \leq i < b} M[a-1, i] + g(a, i)$, defined for $1 \leq a, b \leq n$. As base cases, $M[i, j] = 0$ for any i, j not in the range 1 to n. You would like to compute $M[a, b]$ for all a, b.

Select all valid orders which minimize the required **concrete** space requirement (if multiple orders achieve the smallest possible space, select all of those orders). Concrete here means that leading constants matter, e.g. $2n^2$ space is considered less than $3n^2$ space.

- ☐ For $a = 1$ to n, for $b = 1$ to n, compute $M[a, b]$ using recurrence relation
- [✓] For $a = 1$ to n, for $b = n$ to 1, compute $M[a, b]$ using recurrence relation
- ☐ For $a = n$ to 1, for $b = 1$ to n, compute $M[a, b]$ using recurrence relation
- ☐ For $a = n$ to 1, for $b = n$ to 1, compute $M[a, b]$ using recurrence relation
- ☐ None of the other answers

iv. (1 pt) Consider the following pseudocode. Which one of the following recurrence relations could be computed in this order? Ignore the base cases for each relation.

```
for s = 1 to n-1:
  for i = 1 to n-s:
    j = i + s
    compute M(i, j)
```

- ○ $M(i,j) = \min_{k\in[n]}\{M(i-k,j) + M(i,j-k) + f(i,j,k)\}$
- ○ $M(i,j) = \min_{i\le k\le j}\{M(i-k,j) + M(i,j-k) + f(i,j,k)\}$
- ○ $M(i,j) = \min_{i\le k\le j}\{M(i,k) + M(j,k+1) + f(i,j,k)\}$
- ○ $M(i,j) = \min_{k\in[n]}\{M(i,k) + M(j,k+1) + f(i,j,k)\}$
- [✓ $M(i,j) = \min_{i\le k\le j}\{M(i,k) + M(k+1,j) + f(i,j,k)\}$]

v. (1 pt) Neha is trying transform her current document x into a desired document y. It takes her I seconds to insert a character, R seconds to remove a character, and S seconds to substitute a character.

Define a recurrence relation $T(i,j)$ as the time it takes to transform the first i characters of x into the first j characters of y. The recurrence is of this form:

$$T(i,j) = \min\{\dots\}$$

Select all expressions that should appear in the min.

- ☐ $T(i,j-1)$
- ☐ $T(i,j-1)+R$
- ☐ $T(i,j-1)+S$
- [✓ $T(i,j-1)+I$]
- ☐ $T(i-1,j)$
- [✓ $T(i-1,j)+R$]
- ☐ $T(i-1,j)+S$
- ☐ $T(i-1,j)+I$
- ☐ $T(i-1,j-1)$ if$(x[i]\neq y[j])$
- ☐ $T(i-1,j-1)+R$ if$(x[i]\neq y[j])$
- [✓ $T(i-1,j-1)+S$ if$(x[i]\neq y[j])$]
- ☐ $T(i-1,j-1)+I$ if$(x[i]\neq y[j])$
- [✓ $T(i-1,j-1)$ if$(x[i]= y[j])$]
- ☐ $T(i-1,j-1)+R$ if$(x[i]= y[j])$
- ☐ $T(i-1,j-1)+S$ if$(x[i]= y[j])$
- ☐ $T(i-1,j-1)+I$ if$(x[i]= y[j])$

vi. (1 pt) Let $M[a, b, i, j]$ be a matrix of subproblems, with $0 \le a \le 3, 0 \le b \le n, 0 \le i \le \log n, 0 \le j \le 5$. Let $f(i, j, k)$ be an arbitrary integer-valued function that is fixed ahead of time, so computing it for any particular input takes constant time. Consider the following recurrence:

$$M[a, b, i, j] = \max_{0 \le x < i, 0 \le y < j} M[a-1, b-1, x, y]^2 + f(i-x, y-j, a+b)$$

Assume all base cases $M[0, 0, i, j]$ have been computed. Give the tightest possible asymptotic bound on the runtime to compute all entries in M.

Θ(n log^2(n))

(t) Linear Programming

i. (1 pt) It's possible for an optimal solution to a linear program to be on an edge of the feasible region, as opposed to just a vertex.

[✓ True]

○ False

ii. (1 pt) Consider a linear program whose feasible region is a cube in d-dimensional space. What is the smallest number of variables that could be in such a program?

d

iii. (1 pt) Consider a linear program whose feasible region is a cube in d-dimensional space. What is the smallest number of constraints that could be in such a program (do not include the non-negativity constraints)?

d

iv. (1 pt) Consider a linear program whose feasible region is a cube in d-dimensional space. What is the maximum number of iterations that the Simplex method would need to find the optimal solution?

d

v. (1 pt) Consider the following LP:

$$\begin{aligned} &\max x + y \\ \text{s.t.}\quad & x \le 10 \\ & y \le 20 \\ & \alpha \cdot x + \beta \cdot y \le C \\ & x, y \ge 0 \end{aligned}$$

where α, β, and C are non-negative real numbers. For what values of α, β, and C does the LP have an optimum *that is not unique*?

alpha = beta and c/alpha< 30

vi. (1 pt) x^* and y^* are both optimal points of some LP. Let $z = \frac{x^*+y^*}{2}$.

[✓ z is always in the feasible region]

☐ z is sometimes in the feasible region

☐ z is never in the feasible region

[✓ z is always an optimal solution as well]

☐ z is sometimes be an optimal solution as well

☐ z is never an optimal solution as well

(u) Duality

i. (1 pt) Let A be an $m \times n$ matrix of real numbers. Consider $x = \min_i \max_j A_{i,j}$ and $y = \max_j \min_i A_{i,j}$. Select the tightest comparison that holds in general (e.g. $x < y$, $x \geq y$, or no relation holds in general).

- ○ $x < y$
- ○ $x > y$
- ○ $x \leq y$
- [✓ $x \geq y$]
- ○ $x = y$
- ○ no relation between x and y holds in general

ii. (1 pt) Consider the linear program $\max c^T x$ subject to $Ax \leq b$. Assume the LP is feasible and that its objective value is bounded, and let x^* and y^* be optimal solutions to the primal and dual problems, respectively. It is required that $c^T x^* = b^T y^*$.

- [✓ True]
- False

iii. (1 pt) Fill in the blank to complete the proof for weak duality and briefly justify.

$$\begin{array}{cc} \textit{Primal}: & \textit{Dual}: \\ \max \mathbf{c}^T\mathbf{x} & \min \mathbf{y}^T\mathbf{b} \\ \mathbf{Ax} \leq \mathbf{b} & \mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T \\ \mathbf{x} \geq 0 & \mathbf{y} \geq 0 \end{array}$$

Proof: $c^T x \leq$ ________ $\leq y^T b$

y^(T) A x. Multiply the first dual constraint by x and combine with the first primal constraint

iv. (1 pt) For LPs, if the primal problem is a minimization problem, weak duality states that any feasible solution to the dual problem upper bounds the optimal value of the primal problem.

- ○ True
- [✓ False]

No more questions.