COS 226 **Algorithms and Data Structures** **Fall 2023**

Final

This exam has 13 questions worth a total of 100 points. You have 180 minutes.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated spaces. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and redo.

Resources. The exam is closed book, except that you are allowed to use a one page reference sheet (8.5-by-11 paper, both sides, in your own handwriting). No electronic devices are permitted.

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before the solutions are posted is a violation of the Honor Code.

Please complete the following information now.

Name:

NetID:

Exam room: ◯ McCosh 46 ◯ McCosh 50 ◯ McCosh 60 ◯ Other

Precept:

P01	P02	P03	P04	P05	P06	P07	P08	P09
◯	◯	◯	◯	◯	◯	◯	◯	◯

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

1. **Initialization. (1 point)**

   In the spaces provided on the front of the exam, write your name and NetID; fill in the bubble for your exam room and the precept in which you are officially registered; write and sign the Honor Code pledge.

2. **Empirical running time. (6 points)**

   Suppose that you observe the following running times (in seconds) for a program on graphs with V vertices and E edges.

		E			
		100	400	1600	6400
V	100	0.3	1.0	4.0	16.0
	200	1.0	4.0	16.0	64.0
	400	4.0	16.0	64.0	256.0
	800	16.0	64.0	256.0	1024.0

   (a) Estimate the running time of the program (in seconds) for a graph with V = 1,600 vertices and E = 25,600 edges. *Fill in the best-matching bubble.*

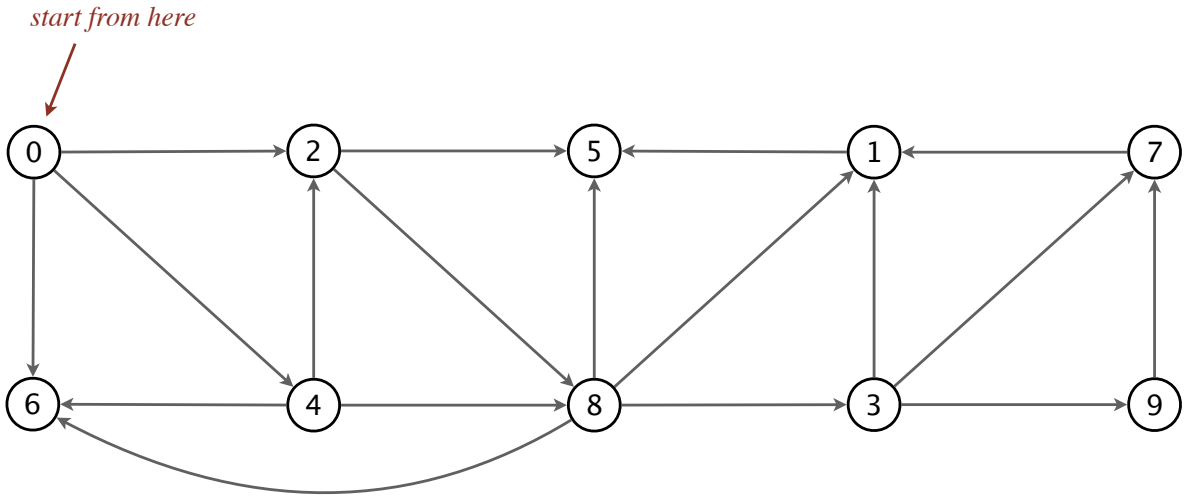
   ◯ 2,000 ◯ 4,000 ◯ 8,000 ◯ 16,000 ◯ 32,000

   (b) Estimate the *order of growth* of the running time as a function of both V and E. *Fill in the best-matching bubble.*

   ◯ $\Theta(V^2 + E^2)$ ◯ $\Theta(E + V^2)$ ◯ $\Theta(V^2 E)$ ◯ $\Theta(V E^2)$ ◯ $\Theta(V^2 E^2)$

3. **Depth-first search. (9 points)**

Run *depth-first search* on the following digraph, starting from vertex 0. Assume the adjacency lists are in sorted order: for example, when iterating over the edges leaving vertex 0, consider the edge 0→2 before either 0→4 or 0→6.



(a) List the 10 vertices in *DFS preorder*.

0 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

(b) List the 10 vertices in *DFS postorder*.

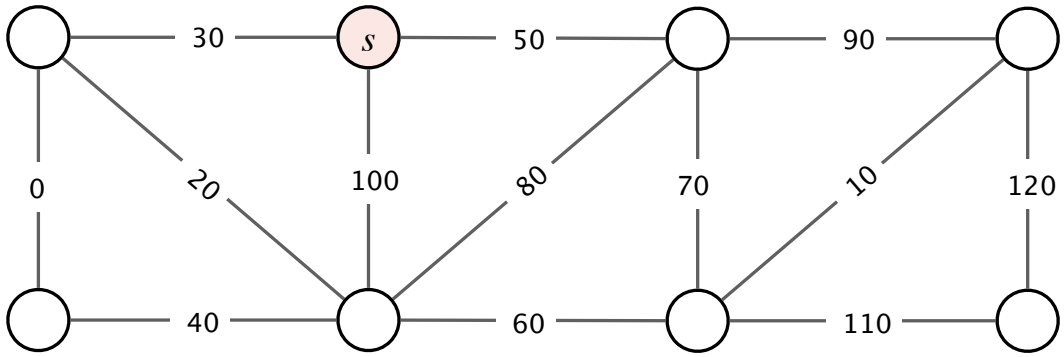
___ ___ ___ ___ ___ ___ ___ ___ ___ 0

(c) Is the *reverse* of the DFS postorder in (b) a *topological order* for this digraph?

◯ *yes* ◯ *no*

4. **Minimum spanning trees. (8 points)**

Consider the following edge-weighted graph.



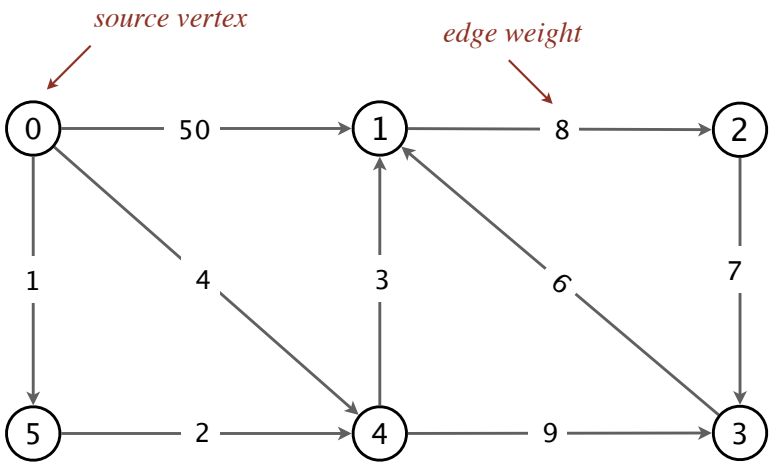
(a) List the weights of the MST edges in the order that *Kruskal's algorithm* adds them to the MST.

(b) List the weights of the MST edges in the order that *Prim's algorithm* adds them to the MST. Start Prim's algorithm from vertex s.

5. **Shortest paths. (8 points)**

Consider running the *Bellman–Ford algorithm* in the following edge-weighted digraph, with source vertex $s = 0$. Assume that, within a pass, the edges are relaxed in sorted order:

$$0\to1,\ 0\to4,\ 0\to5,\ 1\to2,\ 2\to3,\ 3\to1,\ 4\to1,\ 4\to3,\ 5\to4$$



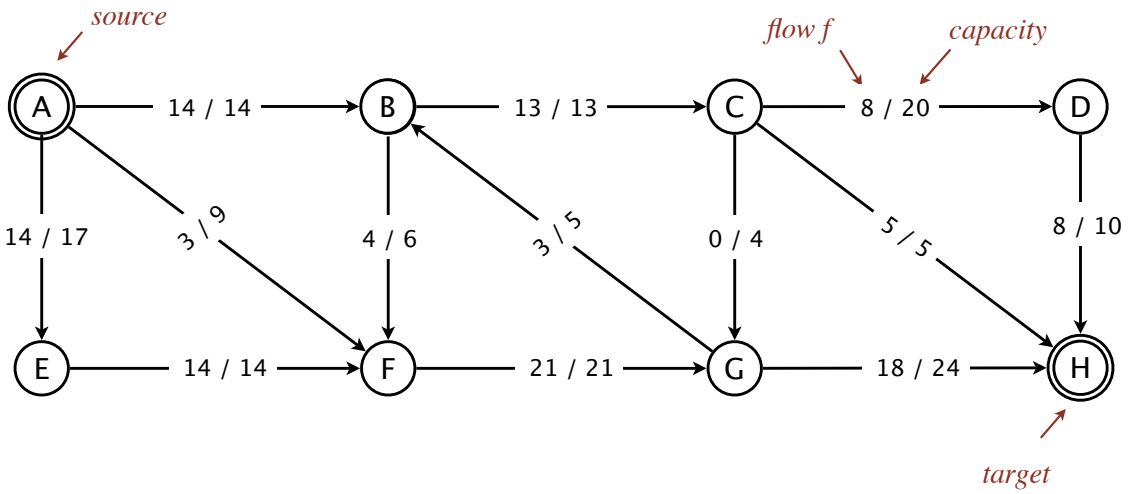
(a) Immediately *after the first pass*, what are the values of `distTo[v]` for each vertex v? *Write the values in the corresponding boxes.*

0					
`distTo[0]`	`distTo[1]`	`distTo[2]`	`distTo[3]`	`distTo[4]`	`distTo[5]`

(b) Immediately *after the first pass*, for which vertices v is `distTo[v]` the length of the *shortest* path from s to v? *Mark all vertices that apply.*

■ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

6. **Maxflows and mincuts. (10 points)**
Consider the following flow network and a *flow* f.



(a) What is the *value* of the flow f?

◯ 29 ◯ 31 ◯ 34 ◯ 37 ◯ 39

(b) What is the *capacity* of the cut $\{A, B, E, F\}$?

◯ 29 ◯ 31 ◯ 34 ◯ 37 ◯ 39

(c) What is the *net flow* across the cut $\{A, B, E, F\}$?

◯ 29 ◯ 31 ◯ 34 ◯ 37 ◯ 39

(d) Find an *augmenting path* with respect to f. Write the sequence of vertices in the path.

$A \rightarrow$

(e) What is the *bottleneck capacity* of the augmenting path found in part (d)?

◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5

7. **Data structures. (10 points)**

(a) Suppose that the following keys are inserted into an initially empty *linear-probing hash table*, but *not* necessarily in the order given:

key	*hash*
A	1
B	1
C	4
D	3
E	2

Which of the following could be the contents of the underlying array? Assume that the length of the array is 6 and that it neither grows nor shrinks.

Fill in all checkboxes that apply.

☐

0	1	2	3	4	5
–	A	B	C	D	E

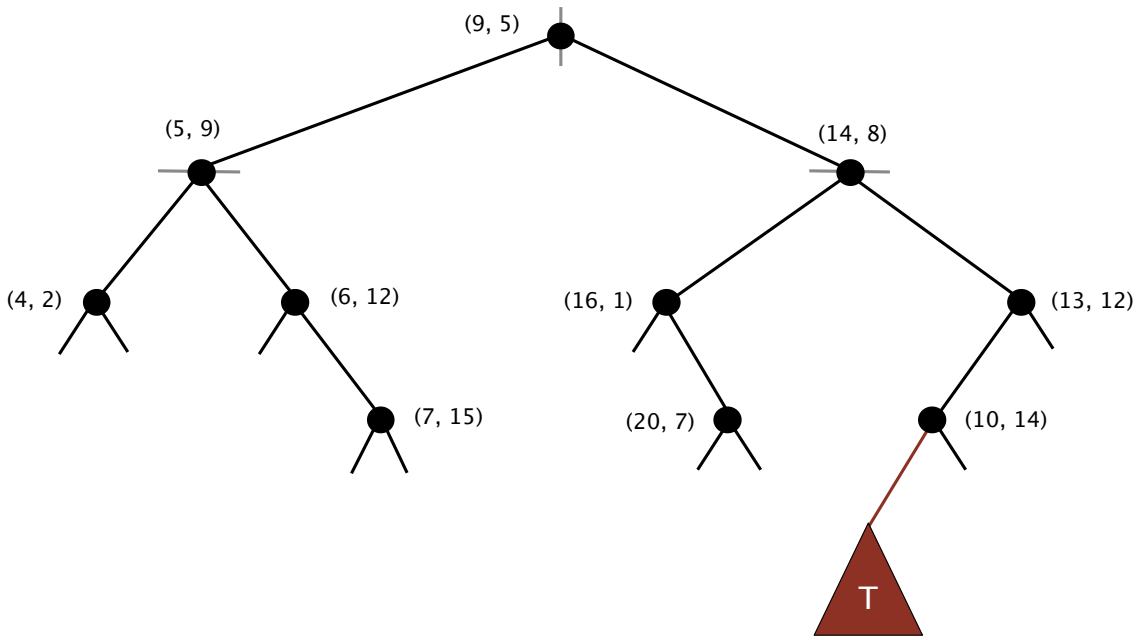
☐

0	1	2	3	4	5
–	A	B	D	C	E

☐

0	1	2	3	4	5
–	B	A	E	D	C

(b) Consider the following *2d-tree*:



Which of the following points could be in the subtree T?

Fill in all checkboxes that apply.

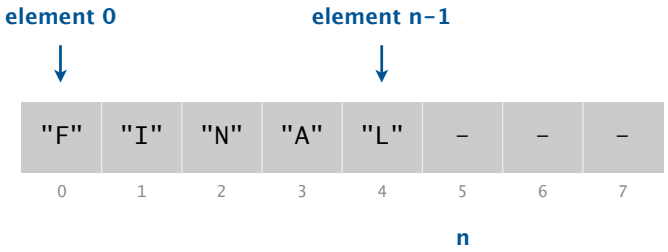
☐ (5, 10) ☐ (7, 16) ☐ (10, 10) ☐ (11, 16) ☐ (12, 9) ☐ (16, 13)

(c) Consider the following code fragment for creating a uniformly shuffled version of an `ArrayList` containing n strings.

```
ArrayList<String> from = ...;
ArrayList<String> to = new ArrayList<String>();
while (from.size() > 0) {
    int r = StdRandom.uniformInt(from.size());
    String x = from.remove(r);  // remove and return item at index r in list,
                                // shifting subsequent elements to the left
    to.append(x);               // appends x to the end of the list
}
```

Assume that the `ArrayList` data type is implemented using a *resizing array* (with doubling when full and halving when one-quarter full) and that element i in the list is stored at index i in the resizing array.

All operations perform as efficiently as could be expected for this representation.



What is the order-of-growth of the *worst-case* running time as a function of n?

◯ $\Theta(1)$ ◯ $\Theta(n)$ ◯ $\Theta(n \log n)$ ◯ $\Theta(n^2)$ ◯ $\Theta(n^3)$

What is the order-of-growth of the *best-case* running time as a function of n?

◯ $\Theta(1)$ ◯ $\Theta(n)$ ◯ $\Theta(n \log n)$ ◯ $\Theta(n^2)$ ◯ $\Theta(n^3)$

8. **Dynamic programming. (6 points)**

You are taking an idealized exam with n questions and have m minutes to complete it. Question j is worth p_j points and takes t_j minutes to earn the points. Your goal is to maximize the number of points earned in the allotted time. Assume that all p_j and t_j are positive integers (and that there is no partial credit).

You will solve this problem using *dynamic programming*. Define the following subproblems, one for each i and j with $0 \le i \le m$ and $0 \le j \le n$:

$OPT(i, j)$ = *max points earned in* i *minutes by working only on questions* 1 *through* j

Consider the following partial bottom-up implementation:

```
int[][] opt = new int[m+1][n+1];

for ( 1 ) {
    for ( 2 ) {
        if ( 3 ) {
            opt[i][j] = 4 ;
        }
        else {
            opt[i][j] = Math.max( 5 ,
                                  points[j] + 6 );
        }
    }
}
```

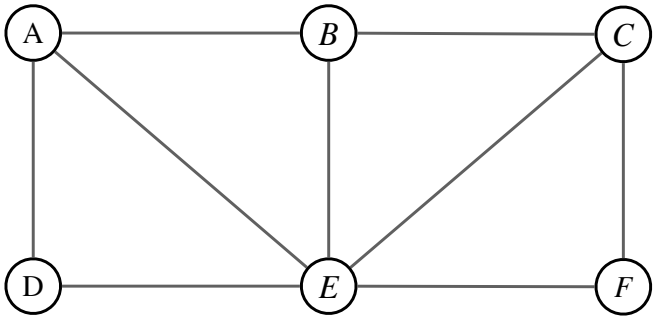
A. `(int i = 1; i <= m; i++)`
B. `(int i = m; i >= 1; i--)`
C. `(int j = 1; j <= n; j++)`
D. `(int j = n; j >= 1; j--)`
E. `times[j] > i`
F. `points[j] > i`
G. `opt[i-1][j]`
H. `opt[i][j-1]`
I. `opt[i-1][j-1]`
J. `points[j]`
K. `times[j]`
L. `opt[i - times[j]][j-1]`
M. `opt[i - points[j]][j-1]`

For each numbered oval above, write the letter of the corresponding code fragment on the right in the space provided. You may use each letter once, more than once, or not at all.

1	2	3	4	5	6

9. **Karger's algorithm. (5 points)**

Run one execution of Karger's algorithm for finding a *global mincut* in the following graph. The table at right gives the uniformly random weights that this execution of Karger's algorithm assigns to the edges.



edge	*random weight*
A–B	0.5
A–D	0.4
A–E	0.6
B–C	0.9
B–E	0.8
C–E	0.1
C–F	0.3
D–E	0.7
E–F	0.2

(a) Which cut does this execution of Karger's algorithm find?
Mark all vertices that are on the same side of the cut as vertex A.

A ■ *B* □ *C* □ *D* □ *E* □ *F* □

(b) How many edges cross the cut found by this execution of Karger's algorithm?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4

10. **Multiplicative weights (9 points).**

Consider the *experts problem* with $n \geq 2$ experts over a period of T days.

Identify each property as either always true or sometimes/always false.

true	*false*	
◯	◯	Suppose that one of the n experts always predicts correctly. Then, the total number of mistakes made by the *elimination algorithm* is $\leq \log_2 n$.
◯	◯	Suppose that one of the n experts always predicts correctly. Then, after $\lceil \log_2 n \rceil$ days, there will be exactly one expert remaining in the *elimination algorithm.*
◯	◯	Suppose that exactly two of the n experts always predict correctly. Then, the total number of mistakes made by the *elimination algorithm* is $\leq \frac{1}{2} \log_2 n$.
◯	◯	Suppose that more than $n/2$ of the n experts predict 1 on a given day. Then, the *multiplicative weights* algorithm also predicts 1 for that day.
◯	◯	In the *multiplicative weights* algorithm, an expert who has made 5 mistakes will have exactly one-half of the weight of an expert who has made 10 mistakes.
◯	◯	Suppose that the best expert makes 7 mistakes. Then, the total number of mistakes made by the *multiplicative weights* algorithm is ≥ 7.

11. **Intractability (8 points).**

Suppose that Problem X is **NP**-complete; Problem Y is in **NP**; and Problem X poly-time reduces to Problem Y. Which of the following can you infer? *Fill in all checkboxes that apply.*

☐ Problem X is SAT.

☐ Problem X is in **NP**.

☐ The INTEGER-FACTORIZATION problem poly-time reduces to Problem X.

☐ Problem Y poly-time reduces to Problem X.

☐ Problem Y is **NP**-complete.

☐ If Problem X can be solved in poly-time, then $\mathbf{P} = \mathbf{NP}$.

☐ If Problem Y cannot be solved in poly-time, then $\mathbf{P} \neq \mathbf{NP}$.

☐ $\mathbf{P} \neq \mathbf{NP}$.

12. **Princeton path game. (10 points)**

Two players compete on a digraph G with two distinguished vertices, s and t.

- The *orange player* tries to build a directed path from vertex s to vertex t. The *black player* tries to prevent this.
- The two players alternate moves. The orange player moves by coloring an uncolored edge orange. The black player moves by coloring an uncolored edge black.
- The orange player wins if there is a directed path of orange edges from s to t. The black player wins if every directed path from s to t contains a black edge.

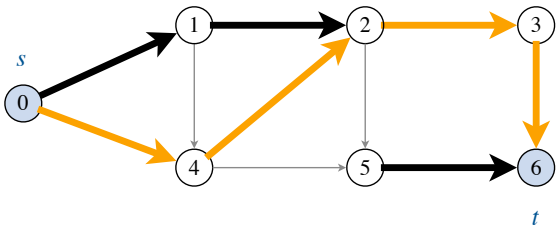
To make the game interesting, assume that $s \neq t$ and that G contains at least one directed path from s to t.

Goal. Your goal is to design an algorithm that, given the current state of the game (i.e., a graph G with each edge either uncolored, orange, or black), determines whether either player has already won and, if so, who. Note that the game may end before all of the edges are colored.

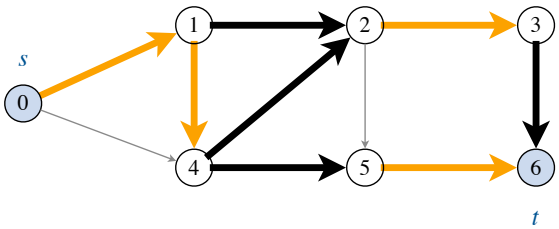
- The orange player wins as soon as there is a directed path of orange edges from s to t.
- The black players wins as soon as every directed path from s to t contains one (or more) black edges.

Examples. Consider two examples of the game being played on the same digraph.

- In the example at left, the orange player has won: the directed path $0 \to 4 \to 2 \to 3 \to 6$ contains only orange edges.
- In the example at right, the black player has won: every directed path from s to t contains one of the black edges $1 \to 2$, $4 \to 2$, or $4 \to 5$.



orange wins
(s-t path of orange edges)



black wins
(every s-t path contains a black edge)

Performance requirements. For full credit, your algorithm must take $\Theta(E + V)$ time, where V and E are the number of vertices and edges in G, respectively. Assume that, given access to an edge, you can determine its color in $\Theta(1)$ time.

Your answer will be graded for correctness, efficiency, and clarity.

(a) Given a digraph G with each edge either uncolored, orange, or black, design an algorithm to determine whether there is a directed path from s to t containing only orange edges.

(b) Given a digraph G with each edge either uncolored, orange, or black, design an algorithm to determine whether every path from s to t contains one (or more) black edges.

(c) Can the game can end in a *tie*, with all edges colored and neither player winning?

Yes ◯ *No* ◯

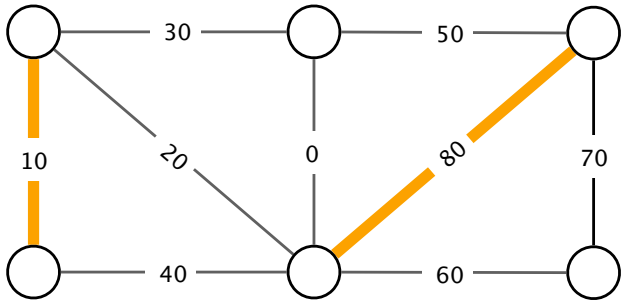
13. **Princeton minimum spanning trees. (10 points)**

Consider the classic minimum spanning tree problem and a variant.

- CLASSIC-MST: Given a connected, edge-weighted graph G', find a spanning tree of G' that has minimum total weight.
- PRINCETON-MST: Given a connected, edge-weighted graph G with each edge colored orange or black, find a spanning tree of G that has minimum total weight among all spanning trees that *contain all of the orange edges* (or report that no such spanning tree exists).

Example. Consider the edge-weighted graph below.

- The CLASSIC-MST includes the edges of weight 0, 10, 20, 50, 60.
- The PRINCETON-MST includes the edges of weight 0, 10, 20, 60, and 80.



Goal. Design an efficient algorithm to solve the PRINCETON-MST problem on an edge-weighted and edge-colored graph G. *To do so, model it as a* CLASSIC-MST *problem on a closely related edge-weighted graph* G'.

Performance requirements. For full credit, your algorithm must run in $O(E \log E)$ time, where V and E are the number of vertices and edges in G, respectively.

Your answer will be graded for correctness, efficiency, and clarity.

(a) Describe your algorithm for solving the PRINCETON-MST problem. Your description should work for any instance of PRINCETON-MST, not just the one on the facing page.

(b) *Draw* the CLASSIC-MST instance G' that your algorithm would construct in order to solve the PRINCETON-MST instance G on the facing page. Be sure to draw the vertices, edges, and edge weights.

This page is intentionally blank. You may use this page for scratch work.