COS 226 **Algorithms and Data Structures** **Fall 2010**

Final

This test has 14 questions worth a total of 100 points. You have 180 minutes. The exam is closed book, except that you are allowed to use a one page cheatsheet (8.5-by-11, both sides, in your own handwriting). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write out and sign the Honor Code pledge before turning in the test.**

"I pledge my honor that I have not violated the Honor Code during this examination."

Problem	Score
0	
1	
2	
3	
4	
5	
6	
Sub 1	

Problem	Score
7	
8	
9	
10	
11	
12	
13	
Sub 2	

Total	

Name:

Login ID:

Precept:

P01	11	Bob Tarjan
P02	12:30	Yuri Pritykin
P02A	12:30	Bob Tarjan
P03	1:30	Aman Dhesi
P03A	1:30	Siyu Yang

0. **Miscellaneous. (1 point)**

Write your name and Princeton NetID in the space provided on the front of the exam, and circle your precept number.

1. **Analysis of algorithms. (12 points)**

Which of the following can be performed in *linear time* in the *worst case*? For the purposes of this question, assume $P \neq NP$. Write P (possible), I (impossible), or U (unknown).

___ Find a *maximum* spanning tree in a connected edge-weighted graph.

___ Find all vertices reachable from a given *set* of source vertices in a digraph.

___ Find a Hamilton path in a digraph (if one exists).
A *Hamilton path* is a simple path that visits each vertex in the digraph exactly once.

___ Find a Hamilton path in a DAG (if one exists).

___ Find the strong components of a digraph.

___ Insert N `Comparable` keys into a binary heap.

___ Sort an array of N `Comparable` keys.

___ Insert N `Comparable` keys into a binary search tree.

___ Compute the inverse Burrows-Wheeler transform.

___ Insert N strings into an R-way trie.
(Here, linear means linear in the sum of the lengths of the N strings.)

___ Print the N strings in a ternary search trie in ascending order.
(Here, linear means linear in the sum of the lengths of the N strings.)

___ Perform a nearest-neighbor query in a 2-d tree.

2. **Equivalence relations. (5 points)**

   Which of the following are equivalence relations?
   Put a ✓ next to those that are equivalence relations.

   ___ `v.equals(w)` for objects in a Java class that correctly implements the `equals()` method

   ___ `v.compareTo(w) < 0` for objects in a Java class that correctly implements the `Comparable` interface

   ___ `connected(v, w)` in `CC` for connectivity in an undirected graph

   ___ `reachable(v, w)` in `TransitiveClosure` for reachability in a digraph

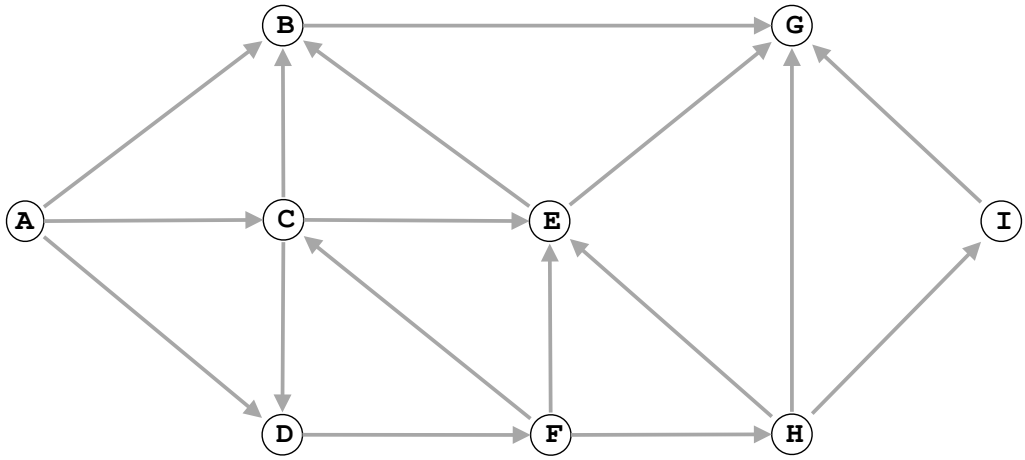
   ___ `stronglyConnected(v, w)` in `KosarajuSCC` for strong connectivity in a digraph

   Recall, an *equivalence relation* $\equiv$ is a binary relation that is

   - *reflexive*: $v \equiv v$
   - *symmetric*: if $v \equiv w$, then $w \equiv v$
   - *transitive*: if $v \equiv w$ and $w \equiv x$, then $v \equiv x$

3. **Depth-first search. (8 points)**

(a) Run *depth-first search* on the digraph below, starting at vertex A. Assume the adjacency lists are in sorted order: for example, when exploring vertex F, consider the edge $F \to C$ before $F \to E$ or $F \to H$.



List the vertices in preorder and postorder.

```
preorder:    A    B    ___   ___   ___   ___   ___   ___   ___

postorder:   G    B    ___   ___   ___   ___   ___   ___   ___
```

(b) Consider two vertices x and y that are simultaneously on the function-call stack at some point during the execution of depth-first search from vertex s in a *digraph*. Which of the following must be true?

I. There is *both* a directed path from s to x *and* a directed path from s to y.

II. If there is *no* directed path from x to y, then there is a directed path from y to x.

III. There is *both* a directed path from x to y *and* a directed path from y to x.

(a) I only.

(b) I and II only.

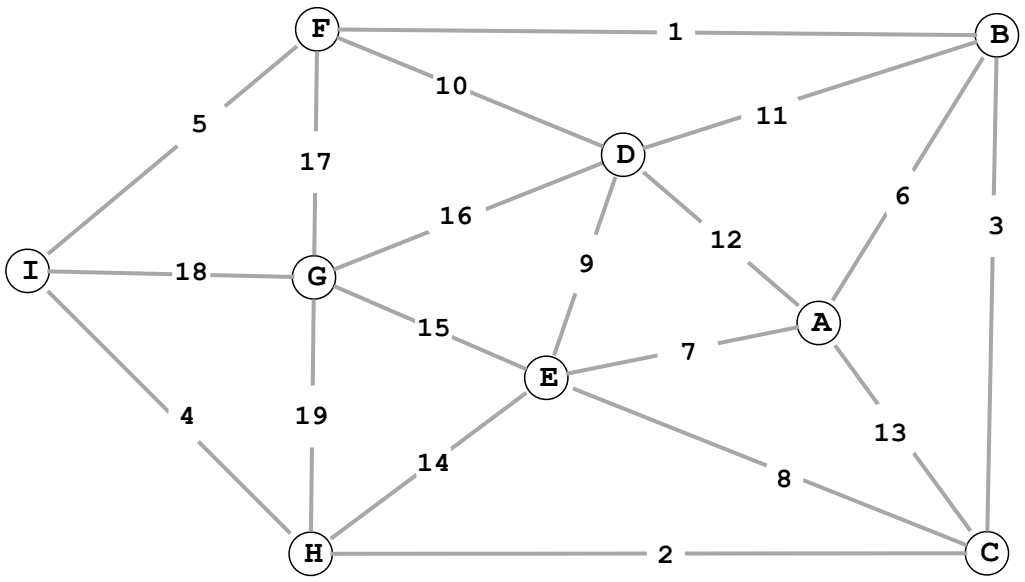
(c) I and III only.

(d) I, II and III.

(e) None.

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

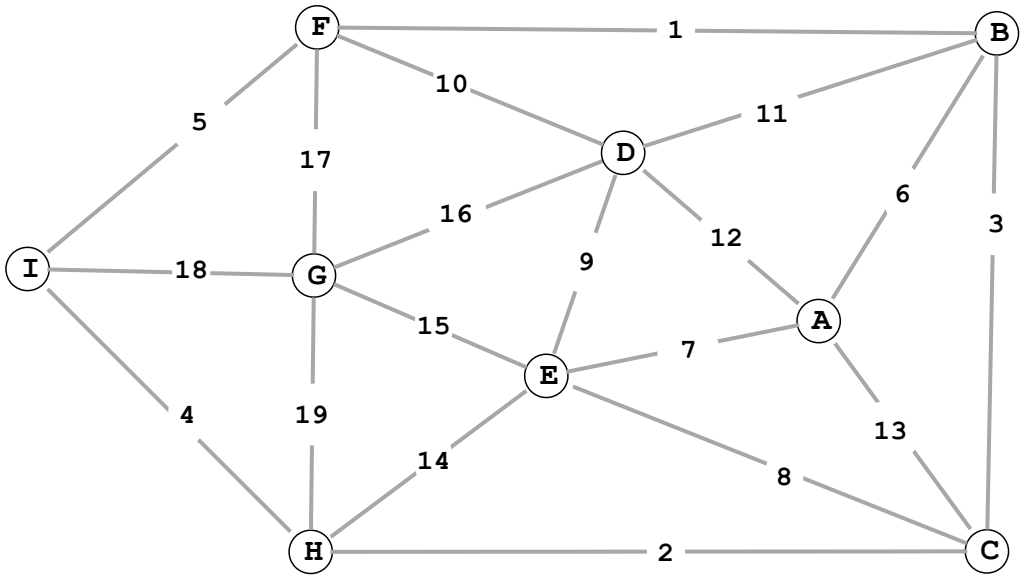
Consider the following edge-weighted graph with 9 vertices and 19 edges. Note that the edge weights are distinct integers between 1 and 19.



(a) Complete the sequence of edges in the MST in the order that *Kruskal's algorithm* includes them (by specifying their edge weights).

1 ____ ____ ____ ____ ____ ____ ____

The edge-weighted graph from the previous page is repeated here for reference.

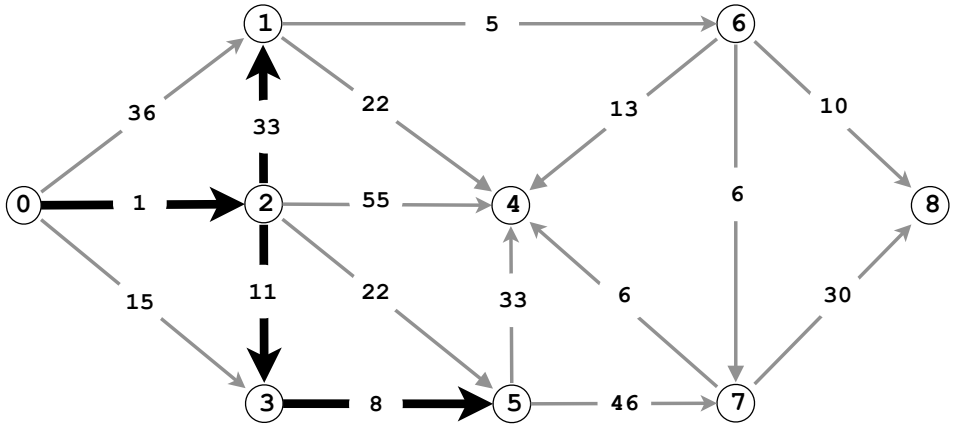


(b) Complete the sequence of edges in the MST in the order that *Prim's algorithm* includes them (by specifying their edge weights). Start Prim's algorithm from vertex A.

6 ____ ____ ____ ____ ____ ____ ____

5. **Shortest paths. (10 points)**

Run the eager version of Dijkstra's algorithm on the following edge-weighted digraph, starting from vertex 0.



(a) Complete the table of `edgeTo[]` and `distTo[]` values immediately after the first 5 vertices (0, 2, 3, 5, and 1) have been deleted from the priority queue and relaxed.

v	edgeTo[]	distTo[]
0	-	0.0
1	2->1 33.0	34.0
2	0->2 1.0	1.0
3	2->3 11.0	12.0
4		
5	3->5 8.0	20.0
6		
7		
8		

(b) Complete the table of `edgeTo[]` and `distTo[]` values immediately after the 6^{th} vertex has been deleted from the priority queue and relaxed. Circle those values that changed from (a).

v	edgeTo[]	distTo[]
0	-	0.0
1	2->1 33.0	34.0
2	0->2 1.0	1.0
3	2->3 11.0	12.0
4		
5	3->5 8.0	20.0
6		
7		
8		

(c) Draw the edges in the (final) shortest-paths tree with thick lines in the figure above.

6. **Polar sort. (6 points)**

Consider the following proposed `Comparator` for sorting points in the plane by polar angle with respect to a base point. It is based on the ccw method from lecture.

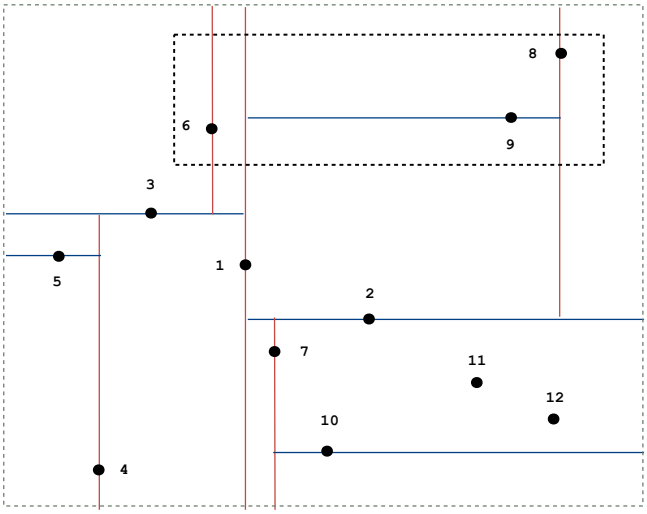
```
import java.util.Comparator;

public class Point {
   private final int x, y;

   public final Comparator<Point> POLAR_ORDER = new PolarOrder();

   public Point(int x, int y) { this.x = x; this.y = y; }

   // is a->b->c a counterclockwise turn?
   // -1 if clockwise; +1 if counterclockwise; 0 if collinear
   public static int ccw(Point a, Point b, Point c) {
      double area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
      if      (area2 < 0) return -1;
      else if (area2 > 0) return +1;
      else                return  0;
   }

   // compare q1 and q2 by polar angle they make with this point
   private class PolarOrder implements Comparator<Point> {
      public int compare(Point q1, Point q2) {
         return ccw(Point.this, q2, q1);
      }
   }
}
```

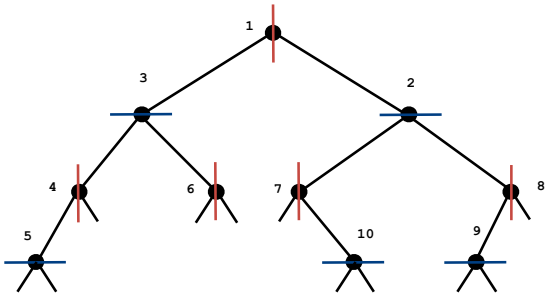
(a) What is the *fatal flaw* with the `Comparator` implementation? Note: there is nothing wrong with the Java code (and `Point.this` is Java's obscure way of referring to the enclosing object from within a nested class). Do not worry about integer overflow.

(b) Suggest an easy approach to fixing the flaw.

7. **Kd-trees. (8 points)**

The figures below illustrate the results of inserting points 1 through 10 into a 2d-tree.





(a) Circle all of the the points below in the 2d-tree that are examined (not necessarily just those inside the query rectangle) during the range search for the query rectangle specified above.

1 2 3 4 5 6 7 8 9 10

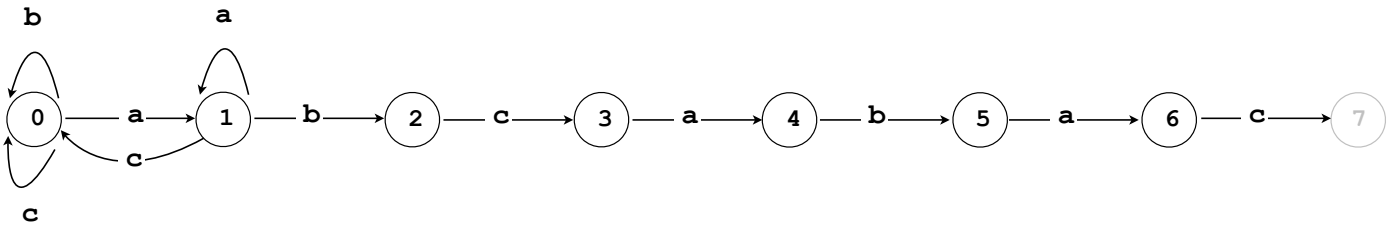
(b) Draw the result of inserting point 11, then point 12 in the *two* figures above.

8. **Substring search. (6 points)**

Create the Knuth-Morris-Pratt DFA for the string `abcabac` over the alphabet { `a, b, c` } by completing the following table. As usual, state 0 is the start state and state 7 is the accept state.

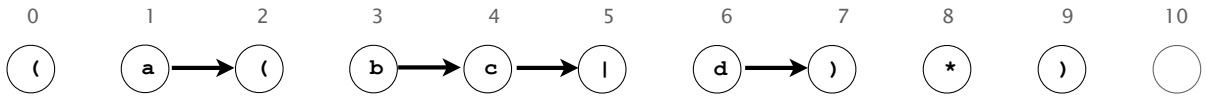
	0	1	2	3	4	5	6
a	1	1		4		6	
b	0	2			5		
c	0	0	3				7

You may use the following partially-completed graphical representation of the DFA for scratch work (but we will consider your solution to be the completed table above).



9. **Regular expressions. (6 points)**

   Convert the regular expression `( a ( b c | d ) * )` into an equivalent NFA (nondeterministic finite state automaton) using the algorithm described in lecture and the textbook by adding ϵ-transition edges to the diagram below.



10. **Substring search and pattern matching. (7 points)**

For each algorithm on the left (the version discussed in lecture and the textbook), give the best-matching *worst-case order of growth* on the right.

_____ brute-force substring search for a query string of size M in a text string of size N

_____ Knuth-Morris Pratt substring search for a query string of size M in a text string of size N

_____ Boyer-Moore (with only mismatch heuristic) substring search for a query string of size M in a text string of size N

_____ Monte Carlo version of Rabin-Karp substring search (that checks only for a hash match) for a query string of size M in a text string of size N

_____ regular-expression pattern matching for a pattern of size M on a text string of size N

_____ simulating a DFA with M vertices and $2M$ edges on a text string of size N

_____ simulating an NFA with M vertices and $3M$ edges on a text string of size N

A. M

B. N/M

C. N

D. $M + N$

E. MN

F. 2^M

G. 2^N

11. **Huffman codes. (5 points)**

Consider the following variable-length codes for the 36-character text string:

```
F C F C E C A C B D E D F E A B F B A F F C D C B E D F F F C C D E E F
```

symbol	freq	code 1	code 2	code 3	code 4	code 5
A	3	110	011	011	1110	100
B	4	111	010	010	1111	101
C	8	10	00	00	00	01
D	5	010	110	101	110	110
E	6	011	001	100	10	111
F	10	00	10	11	01	00

Identify each code (on the left) with the best-matching descriptions (on the right). Write as many letters next to each code as apply.

_____ code 1

_____ code 2

_____ code 3

_____ code 4

_____ code 5

A. prefix-free code

B. Huffman code (assume that when merging the two minimal subtries, either subtrie can become the left or right child)

C. optimal prefix-free code

12. **Cyclic rotation of a strings. (9 points)**

A string s is a *cyclic rotation* of a string t if s and t have the same length and s consists of a suffix of t followed by a prefix of t. Design a linear-time algorithm to determine whether one string is a cyclic rotation of another. You may assume they have the same length N.

For example, `"winterbreak"` is a cyclic rotation of `"breakwinter"` (and vice versa).

Your answer will be graded on correctness, efficiency, clarity, and succinctness. Let N denote the length of s and t. For full credit, the running time of your algorithm should be proportional to N in the worst case.

(a) Describe your algorithm in the space below.

(b) What is the order of growth of the worst-case running time of your algorithm as a function of N? Circle the best answer.

1 $\log N$ N $N \log N$ N^2 N^3

13. **Reductions. (9 points)**

Consider the following two problems:

- MULTIPLICATION. Given two N-bit integers x and y, compute $x \times y$.
- SQUARING. Given an N-bit integer x, compute x^2.

We measure the running time as a function of the number of bits in the input(s). For example, adding or subtracting two N-bit integers takes time proportional to N using the standard grade-school algorithm.

Potentially useful facts about integers: $(a + b)^2 = a^2 + b^2 + 2ab$; $(a - b)^2 = a^2 + b^2 - 2ab$, $(a + b)(a - b) = a^2 - b^2$.

(a) Show that SQUARING linear-time reduces to MULTIPLICATION. To demonstrate your reduction, give the MULTIPLICATION instance(s) that you would construct to solve the following SQUARING instance: given x, compute x^2.

(b) Show that MULTIPLICATION linear-time reduces to SQUARING. To demonstrate your reduction, give the SQUARING instance(s) that you would construct to solve the following MULTIPLICATION instance: given x and y, compute $x \times y$.

(c) Suppose that Alice discovers an $N \log\log\log N$ algorithm for SQUARING and Bob discovers an $N\alpha(N)$ lower bound for MULTIPLICATION, where $\alpha(N)$ is a really really slowly growing (but super-constant) function. Which of the following can you infer from the fact that SQUARING and MULTIPLICATION linear-time reduce to one another?

I. There does not exist a linear-time algorithm for MULTIPLICATION.

II. SQUARING and MULTIPLICATION have the same asymptotic complexity.

III. There exists an $N \log\log\log N$ algorithm for MULTIPLICATION.

(a) I only.

(b) I and II only.

(c) I and III only.

(d) I, II and III.

(e) None.