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.”

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Sub 1</td>
<td></td>
<td>Sub 2</td>
<td></td>
</tr>
</tbody>
</table>

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 ≡ is a binary relation that is

• reflexive: v ≡ v

• symmetric: if v ≡ w, then w ≡ v

• transitive: if v ≡ w and w ≡ x, then v ≡ 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 → C before F → E or F → 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.

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>2-&gt;1</td>
<td>33.0</td>
</tr>
<tr>
<td>2</td>
<td>0-&gt;2</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>2-&gt;3</td>
<td>11.0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3-&gt;5</td>
<td>8.0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>v</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>2-&gt;1</td>
<td>33.0</td>
</tr>
<tr>
<td>2</td>
<td>0-&gt;2</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>2-&gt;3</td>
<td>11.0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3-&gt;5</td>
<td>8.0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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.

```java
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 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.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

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 \(\epsilon\)-transition edges to the diagram below.

```
  (     (     b   c   j     d   j   *   )   )
  \( a \) \(\rightarrow\) \( a \) \(\rightarrow\) \( b \) \(\rightarrow\) \( c \) \(\rightarrow\) \( j \)
```

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 $N$
  
  A. $M$

- ______ Knuth-Morris Pratt substring search for a query string of size $M$ in a text string of size $N$
  
  B. $N/M$

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

- ______ 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 $N$
  
  D. $M + N$

- ______ regular-expression pattern matching for a pattern of size $M$ on a text string of size $N$
  
  E. $MN$

- ______ simulating a DFA with $M$ vertices and $2M$ edges on a text string of size $N$
  
  F. $2^M$

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

<table>
<thead>
<tr>
<th>symbol</th>
<th>freq</th>
<th>code 1</th>
<th>code 2</th>
<th>code 3</th>
<th>code 4</th>
<th>code 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>110</td>
<td>011</td>
<td>011</td>
<td>1110</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>111</td>
<td>010</td>
<td>010</td>
<td>1111</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>10</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>010</td>
<td>110</td>
<td>101</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>011</td>
<td>001</td>
<td>100</td>
<td>10</td>
<td>111</td>
</tr>
<tr>
<td>F</td>
<td>10</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>

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.

\[ 1 \quad \log N \quad N \quad N \log N \quad N^2 \quad N^3 \]
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.