This exam has 14 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
- Other

**Precept:**

- P01
- P02
- P02A
- P03
- P03A
- P03B
- P04
- P04A
- P05

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

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.25</td>
<td>0.5</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>( V ) 200</td>
<td>2.0</td>
<td>4.0</td>
<td>8.0</td>
<td>16.0</td>
</tr>
<tr>
<td>400</td>
<td>16.0</td>
<td>32.0</td>
<td>64.0</td>
<td>128.0</td>
</tr>
<tr>
<td>800</td>
<td>128.0</td>
<td>256.0</td>
<td>512.0</td>
<td>1024.0</td>
</tr>
</tbody>
</table>

(a) Estimate the running time of the program (in seconds) for a graph with \( V = 1,600 \) vertices and \( E = 1,600 \) edges.

\[
\begin{array}{cccccc}
2,000 & 4,000 & 8,000 & 16,000 & 32,000 \\
\end{array}
\]

(b) What is the order of growth of the running time as a function of both \( V \) and \( E \)?

\[
\begin{array}{cccccc}
\Theta(V^3 + E) & \Theta(V + E^3) & \Theta(V^2E) & \Theta(V^3E) & \Theta(V^2E^2) \\
\end{array}
\]
3. Analysis of algorithms. (6 points)

Determine the order of growth of the running time of each of the following code fragments as a function of $V$ and $E$, where $V$ and $E$ are the number of vertices and edges in graph $G$, respectively. Assume the standard adjacency-lists representation.

(a) 
```java
int count = 0;
int V = G.V();
for (int v = 0; v < V; v++)
    for (int w = 0; w < v; w++)
        count++;
```

\[
\begin{align*}
\Theta(V) & \quad \Theta(E) & \quad \Theta(V\log V) & \quad \Theta(V^2) & \quad \Theta(V^2\log V) \\
\hline
\end{align*}
\]

(b) 
```java
int count = 0;
int V = G.V();
for (int v = 0; v < V; v++)
    for (int w : G.adj(v))
        count++;
```

\[
\begin{align*}
\Theta(V) & \quad \Theta(E) & \quad \Theta(E + V) & \quad \Theta(V^2) & \quad \Theta(VE) \\
\hline
\end{align*}
\]

(c) 
```java
int count = 0;
int V = G.V();
for (int v = V; v >= 1; v = v / 2)
    for (int w = 1; w <= v; w++)
        count++;
```

\[
\begin{align*}
\Theta(V) & \quad \Theta(E) & \quad \Theta(V\log V) & \quad \Theta(V^2) & \quad \Theta(V^2\log V) \\
\hline
\end{align*}
\]
4. **String sorts. (5 points)**

The column on the left contains the original input of 24 strings to be sorted; the column on the right contains the strings in sorted order; the other 5 columns contain the contents at some intermediate step during one of the 3 radix-sorting algorithms listed below. Match each algorithm by writing its letter in the box under the corresponding column.

*You may use each letter once, more than once, or not at all.*

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3543</td>
<td>1100</td>
<td>2346</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>2346</td>
<td>6501</td>
<td>1664</td>
<td>1491</td>
</tr>
<tr>
<td>2</td>
<td>9397</td>
<td>3006</td>
<td>1100</td>
<td>1532</td>
</tr>
<tr>
<td>3</td>
<td>8686</td>
<td>5609</td>
<td>1563</td>
<td>1563</td>
</tr>
<tr>
<td>4</td>
<td>1100</td>
<td>5316</td>
<td>1719</td>
<td>1664</td>
</tr>
<tr>
<td>5</td>
<td>3239</td>
<td>3117</td>
<td>1532</td>
<td>1719</td>
</tr>
<tr>
<td>6</td>
<td>9458</td>
<td>3419</td>
<td>1864</td>
<td>1864</td>
</tr>
<tr>
<td>7</td>
<td>7868</td>
<td>1719</td>
<td>1491</td>
<td>2346</td>
</tr>
<tr>
<td>8</td>
<td>5609</td>
<td>5629</td>
<td>3239</td>
<td>3543</td>
</tr>
<tr>
<td>9</td>
<td>5316</td>
<td>1532</td>
<td>3419</td>
<td>3239</td>
</tr>
<tr>
<td>10</td>
<td>3006</td>
<td>3239</td>
<td>3006</td>
<td>3006</td>
</tr>
<tr>
<td>11</td>
<td>1864</td>
<td>3543</td>
<td>3117</td>
<td>3419</td>
</tr>
<tr>
<td>12</td>
<td>1491</td>
<td>2346</td>
<td>3543</td>
<td>3117</td>
</tr>
<tr>
<td>13</td>
<td>3419</td>
<td>4149</td>
<td>4149</td>
<td>4149</td>
</tr>
<tr>
<td>14</td>
<td>4149</td>
<td>9458</td>
<td>7584</td>
<td>5609</td>
</tr>
<tr>
<td>15</td>
<td>7584</td>
<td>1563</td>
<td>5316</td>
<td>5316</td>
</tr>
<tr>
<td>16</td>
<td>1532</td>
<td>1864</td>
<td>6501</td>
<td>5629</td>
</tr>
<tr>
<td>17</td>
<td>6501</td>
<td>1664</td>
<td>5609</td>
<td>6501</td>
</tr>
<tr>
<td>18</td>
<td>1719</td>
<td>7868</td>
<td>7092</td>
<td>7868</td>
</tr>
<tr>
<td>19</td>
<td>7092</td>
<td>7584</td>
<td>7868</td>
<td>7584</td>
</tr>
<tr>
<td>20</td>
<td>1563</td>
<td>8686</td>
<td>5629</td>
<td>7092</td>
</tr>
<tr>
<td>21</td>
<td>5629</td>
<td>1491</td>
<td>9458</td>
<td>8686</td>
</tr>
<tr>
<td>22</td>
<td>3117</td>
<td>7092</td>
<td>8686</td>
<td>9397</td>
</tr>
<tr>
<td>23</td>
<td>1664</td>
<td>9397</td>
<td>9397</td>
<td>9458</td>
</tr>
</tbody>
</table>

A. Original input

B. LSD radix sort

C. MSD radix sort

D. 3-way radix quicksort (*no shuffle*)

E. Sorted
5. Depth-first search. (8 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?

\[ \text{yes} \quad \text{no} \]
6. 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 $s$. 
7. Shortest paths. (8 points)

Suppose that you are running Dijkstra’s algorithm in the following edge-weighted digraph, with source vertex \( s = 0 \). Just prior to relaxing vertex 6, the \texttt{distTo[]} array is as follows:

\[
\begin{array}{c|c}
\text{v} & \text{distTo[]} \\
\hline
0 & 0.0 \\
1 & 50.0 \\
2 & \\
3 & 43.0 \\
4 & 41.0 \\
5 & 35.0 \\
6 & 38.0 \\
7 & 36.0 \\
\end{array}
\]

(a) Which vertices (including vertex 6) are currently in the priority queue? 
Mark all that apply.

- [ ] 0
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7

(b) Which vertex will Dijkstra’s algorithm relax immediately after vertex 6?

- [ ] 0
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
cannot be determined
- [ ] 7

(c) Which is the weight of edge 7\( \rightarrow \)3?

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
cannot be determined
- [ ] 8
8. Maxflows and mincuts. (8 points)

Consider the following flow network and maximum flow $f$.

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

- 20
- 31
- 32
- 36
- 37

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

- 31
- 42
- 50
- 63
- 76

(c) Which vertices are on the source side of a minimum cut? Mark all that apply.

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J

(d) Suppose that the capacity of edge $B \to C$ is increased from 9 to 10. Which of the following paths would become augmenting paths with respect to flow $f$? Mark all that apply.

- $A \to G \to B \to C \to I \to D \to E \to J$
- $A \to B \to C \to D \to E \to J$
- $A \to G \to B \to C \to I \to J$
- $A \to G \to H \to C \to I \to J$
- none of the above
9. **Data structures. (12 points)**

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

<table>
<thead>
<tr>
<th>key</th>
<th>hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
</tbody>
</table>

Which of the following hash tables could arise? Assume that the initial size of the hash table is 5 and that it neither grows nor shrinks.

*Fill in all checkboxes that apply.*

- [ ] 0 1 2 3 4
  
  A B C D E

- [ ] 0 1 2 3 4
  
  D B C E A

- [ ] 0 1 2 3 4
  
  C D E A B
(b) Consider the following 2d-tree:

Which of the following points could correspond to \((x, y)\)?

*Fill in all checkboxes that apply.*

- [ ] (5, 10)
- [ ] (7, 10)
- [ ] (7, 16)
- [ ] (8, 8)
- [ ] (8, 14)
- [ ] (10, 10)
Consider the following ternary search trie, where the question mark represents an unknown digit:

Which of the following string keys are (or could possibly be) in the TST?

Fill-in all checkboxes that apply.

1 226 236 36 56 646 76

81 822 8225 826 8269 869 96
10. **Data compression. (8 points)**

For each of the following data compression algorithms, identify the *worst-case compression ratio*. Recall that the compression ratio is the number of bits in the encoded message divided by the number of bits in the original message.

*For each algorithm on the left, write the letter of the best-matching term on the right. You may use each letter once, more than once, or not at all.*

- Run-length coding with 8-bit counts.  
  - A. $\sim 1$

- Huffman coding over the extended ASCII alphabet ($R = 256$).  
  - B. $\sim 3/2$

- LZW compression over the extended ASCII alphabet ($R = 256$), with 12-bit codeword.  
  - C. $\sim 2$

- Burrows–Wheeler compression over the extended ASCII alphabet ($R = 256$). This includes the Burrows–Wheeler transform, move-to-front encoding, and Huffman coding.  
  - D. $\sim 4$

  - E. $\sim 8$

  - F. $\sim 12$

  - G. $\sim 16$

  - H. $\sim 256$

(a) What is the Burrows–Wheeler transform of the following string?

A N A B E L L A

Feel free to use this grid for scratch work.

(b) Consider all strings whose core Burrows–Wheeler transform (i.e., the Burrows–Wheeler transform excluding the integer index) is the same as the core Burrows–Wheeler transform of

A N A B E L L A

In the space below, write the lexicographically smallest such string (i.e., the first one that would appear alphabetically).
12. **DFS postorder. (5 points)**

Consider the following partial implementation for computing the DFS postorder in a digraph:

```java
private void dfs(Digraph G, int v) {
    marked[v] = true;
    postorder.enqueue(v);
    for (int w : G.adj(v))
        if (!marked[w])
            dfs(G, w);
}
```

```java
public PostorderDFS(Digraph G) {
    marked = new boolean[G.V()];
    postorder = new Queue<Integer>();
    for (int v = 0; v < G.V(); v++)
        if (!marked[v])
            dfs(G, v);
}
```

For each numbered oval above, write the letter of the corresponding code fragment on the right in the space provided. Use each letter at most once.

For each numbered oval above, write the letter of the corresponding code fragment on the right in the space provided. Use each letter at most once.

```
1 2 3 4 5
```
13. **Shortest tiger path. (10 points)**

Consider a graph in which each vertex is colored black or orange. A *tiger path* is a path that contains exactly one edge whose endpoints have opposite colors.

**Shortest tiger path problem.** Given an undirected graph $G$ and two vertices $s$ and $t$, find a tiger path between $s$ and $t$ that uses the fewest edges (or report that no such path exists).

**An example.** Consider the graph $G$ below with $s = 0$ and $t = 6$.

- The shortest path between $s$ and $t$ is 0–4–5–6, but it is not a tiger path.
- The shortest *tiger path* between $s$ and $t$ is 0–1–2–3–6.

![Graph G](image)

**Goal.** Formulate the shortest tiger path problem as a traditional (unweighted) shortest path problem in a *directed* graph. Specifically, define a digraph $G'$, source $s'$, and destination $t'$ such that the length of the shortest path from $s'$ to $t'$ in $G'$ is always equal to the length of the shortest tiger path between $s$ and $t$ in $G$. For simplicity, you may assume that $s$ is black and $t$ is orange.

**Performance requirements.** For full credit, the number of vertices in $G'$ must be $\Theta(V)$ and the number of edges must be $\Theta(E)$, where $V$ and $E$ are the number of vertices and edges in $G$, respectively.

*Your answer will be graded for correctness, efficiency, and clarity.*
Briefly describe how to construct the digraph $G'$, $s'$, and $t'$ from $G$, $s$, and $t$. Your description should work for any graph $G$, not just the one on the facing page.

Draw the digraph $G'$ corresponding to graph $G$ on the facing page. Label $s'$ and $t'$.
Necklaces. (10 points)

A necklace consists of a sequence $n$ beads, each of which is either orange (0) or black (1). In this question, we have a set of $m$ necklaces and are interested in identifying a common sequence of beads that appears at the end of many necklaces.

The problem. Given a set of $m$ necklaces, each containing $n$ beads, and a positive integer $k \leq n$, design an algorithm to find the most popular ending sequence of length $k$.

An example. Consider the following $m = 4$ necklaces, each containing $n = 5$ beads.

<table>
<thead>
<tr>
<th>bead 1</th>
<th>bead 2</th>
<th>bead 3</th>
<th>bead 4</th>
<th>bead 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>necklace 1</td>
<td>orange</td>
<td>black</td>
<td>orange</td>
<td>black</td>
</tr>
<tr>
<td>necklace 2</td>
<td>orange</td>
<td>orange</td>
<td>orange</td>
<td>black</td>
</tr>
<tr>
<td>necklace 3</td>
<td>black</td>
<td>black</td>
<td>orange</td>
<td>black</td>
</tr>
<tr>
<td>necklace 4</td>
<td>orange</td>
<td>black</td>
<td>black</td>
<td>black</td>
</tr>
</tbody>
</table>

The most popular ending sequences for various values of $k$ are as follows:

- $k = 1$: black (3 necklaces end with a black bead).
- $k = 2$: black–black (3 necklaces end with two black beads).
- $k = 3$: orange–black–black (2 necklaces end with this sequence).
- $k = 4$: black–orange–black–black (2 necklaces end with this sequence).
- $k = 5$: orange–orange–orange–black–orange (1 necklace ends with this sequence).

There are three alternative answers.

Performance requirements. For full credit, the running time of your algorithm must be $\Theta(mn)$ in the worst case.

Your answer will be graded for correctness, efficiency, and clarity (but not Java syntax). If your solution relies upon an algorithm or data structure from the course, do not reinvent it; simply describe how you are applying it.
(a) Describe your algorithm for identifying a most popular ending sequence of length $k$.

(b) *Draw* a diagram of the underlying data structures (such as arrays, linked lists, or binary trees) that your algorithm uses for the example input on the facing page. Show all relevant information, including any links and auxiliary data.
This page is intentionally blank. You may use this page for scratch work.