Write your name and Cornell netid. There are 6 questions on 8 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>/15</td>
<td>/20</td>
<td>/30</td>
<td>/10</td>
<td>/5</td>
<td>/20</td>
<td></td>
</tr>
</tbody>
</table>

Grader
1. (15 points) You probably know the game of “Towers of Hanoi”. Here’s a picture of a typical initial configuration: there is a stack of N disks on the first of three poles (call them A, B and C) and your job is to move the disks from pole A to pole B without ever putting a larger disk on top of a smaller disk. The game can be solved recursively using the code shown below.

```java
class Hanoi {
    public static void main (String args[]) {
        playHanoi(Integer.parseInt(args[0]), "A", "B", "C");
    }

    // Moves n disks from pole "from" to pole "to" using "other" as a temporary place to hold n-1 disks
    static void playHanoi(int n, String from, String to, String other) {
        if (n > 0) {
            playHanoi(n - 1, from, other, to);
            System.out.printf("Move one disk from pole %s to pole %d\n", from, to);
            playHanoi(n - 1, other, to, from);
        }
    }
}
```

(a) 5 points. If moving a single disk from one pole to another is the operation we’re counting, give a formula for the cost of moving N disks in terms of the cost of moving N-1 disks.

\[ C(N) = C(N-1) + 1 + C(N-1) = 2C(N-1)+1 \]

(b) 5 points. Solve the formula in (a), obtaining a polynomial that gives the cost.

\[ C(N) = 2(2\ldots2(2^0+1)+1\ldots)+1 = 2^N+2^{N-1}+2^{N-2}+\ldots+2^0 \]

(c) 5 points. Simplify formula (b). *Hint: Think about the binary representation of an integer.*

\[ C(N) = 2^{N+1}-1 \]
2. (20 points) True or false?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>b</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>c</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>d</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>e</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>f</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>g</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>h</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>i</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>j</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>k</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>l</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>m</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>n</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>o</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>p</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>q</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>r</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>s</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>t</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
3. (30 points) Suppose we create a heap (of maximum size eight) by inserting five elements 9, 17, 5, 31, 2 in the order shown (leaving room for three more). Draw the resulting balanced binary heap (exactly as seen in class) and then fill in the vector with the elements in “heap representation” (again, using the vector representation of a heap seen in class).

(a) 10 points. The heap:

![Binary Heap Diagram]

(b) 10 points. The corresponding vector (put a slash through “empty” elements):

| 2 | 5 | 9 | 31 | 17 | / | / | / |

(c) 10 points. Now show the vector after we call poll() once, to extract the smallest element.

| 5 | 17 | 9 | 31 | / | / | / | / |
4. (10 points) Run Kruskal’s and Prim’s minimum spanning tree algorithms, as presented in class, by hand on the graphs shown below and draw the resulting graphs for us. These algorithms select edges to include as they run. Darken the selected edges and label them 1, 2, 3 so that we can see the order in which they were selected. Circle your edge numbers so that we can’t confuse them with the gray-colored edge weights we provided.

(a) 5 points. Kruskal’s Algorithm

(b) 5 points. Prim’s Algorithm. Start at vertex G.
5) 5 points. Run Dijkstra’s algorithm on the graph below. Start Dijkstra’s algorithm at node A and mark each node with the distance computed by Dijkstra’s method. Be careful when writing the node distances so that we can read what you write and won’t get confused by the gray edge weights we provided.
6. (20 points)

Suppose that we represent a dungeon as an undirected graph. Each node is a room which may contain treasures. Adjacent rooms are stored as an adjacency list. Note that this graph could have cycles.

Our goal is to print directions from a start room to the treasure. If we started in the ATRIUM and we were looking for a RING in the SECRET PASSAGE BEHIND THE WALL, our directions would look like this:

- From ATRIUM go to DUNGEON.
- From DUNGEON go to BROOM CLOSET.
- From BROOM CLOSET go to SECRET PASSAGE BEHIND THE WALL
- In SECRET PASSAGE BEHIND THE WALL is the RING you seek.

Or, if the dungeon does not contain the RING:

Sorry, master, I can’t find you any RING.

a-i) 3 points. Read part (b). Explain how you will solve it here. Include details about which data structures you will use and how you will use them. Hint: Your master (e.g.: your grader) will know graph algorithms well. Focus your efforts not on explaining these algorithms, but explaining how you will implement them. If an algorithm has a name, use that name in your answer.  Limit: 3 sentences.

I would use a recursive depth-first search to explore the maze, putting the Rooms already searched into a Java LinkedList<Room> so that I could rapidly check to see if I’ve already visited a given room. If the treasure is found my routine would remember the sequence of rooms visited.

a-ii) 2 points. Would your answer to (a-i) change if we had asked you to find the shortest path to the treasure? If not, explain why; if so, tell us what you would do differently.

Same solution but now I would use a breadth-first search (slightly more complex to code).
b) 10 points.

class Room {
    // Don’t add more class variables... You may assume that these three are never null
    String name; // Name of this room
    ArrayList<String> treasures; // Treasures contained in this room. Empty list if none
    ArrayList<Room> adjacent; // Rooms reachable from this room. Empty list if none

    public LinkedList<Room> findTreasurePath(String goal) {
        LinkedList<Room> visited = new LinkedList<Room>();
        LinkedList<Room> path = new LinkedList<Room>();
        if(explore(goal, visited, path) == true)
            return path;
        return null;
    }

    private boolean explore(String goal, LinkedList<Room> visited, LinkedList<Room> path) {
        // Been here before?
        if(visited.contains(this))
            return false;

        // Remember that I was here once
        visited.addLast(this);

        // Optimistically, add this room to the end of the path
        path.addLast(this);

        // Did I find my goal?
        if(this.treasures.contains(goal))
            return true;

        // Is there a path to my goal through some adjacent room?
        for(Room r : adjacent)
            if(r.explore(goal, visited, path) == true)
                return true;

        // Can’t get to the goal from this room, remove it from the path
        path.remove(this);
        return false;
    }
}
c) 5 points.

    public void printTreasurePath(String goal, LinkedList<Room> path) {
        // If the object wasn’t found the argument will be null
        if(path == null)
            {
                System.out.printf("Sorry master. I cannot find you any %s\n", goal);
                return;
            }

        Room lastRoom = null;

        // Room by room, list the path to follow. Takes advantage of the fact that a
        // LinkedList can be traversed using “for” in the order the nodes were added
        for(Room r: path) {

            // Print the next step on the path (except on the first room)
            if(from != null)
                System.out.printf("From %s go to %s\n", lastRoom.name, r.name)

            // Remember which room I was in last
            lastRoom = r;
        }

        // After printing the path, print the “final line” of the output
        System.out.printf("In %s is the %s you seek\n", lastRoom, goal);
    }