Problem Set 10

due by 10:00 p.m. on Wednesday, August 6
No late submissions will be accepted.

Notes:
- There are no extra grad-credit problems for this problem set.
- There are no programming problems for this problem set.
- This problem set is pair-optional. See the syllabus for more detail.

Short-Answer Problems (100 points total)
Put your answers in a plain-text file named ps10.txt or a PDF named ps10.pdf.

1. Heaps and heapsort (10 points total)
   a. (3 points) Illustrate the process of turning the tree at right into a max-at-top heap. Show the tree after each sift operation.
   b. (2 points) What is the array representation of the max-at-top heap that you obtain in part a?
   c. (5 points) Heapsort begins by turning the array to be sorted into a heap. Assume that your answer to part b is the result of this process of turning the original array into a heap. Illustrate the remaining steps involved in applying heapsort to this array. Show the contents of the array after each element is put into its final position – i.e., at the end of each iteration of the while loop in the heapsort method covered in lecture.

2. Hash tables (12 points total; 4 points each part)
The following sequence of keys is to be inserted in a hash table of size 8:
   cat, goat, dog, bird, bison, ant, flea, bat, duck
The hash function assigns to each key the number of characters in the key.
   a. Assume that open addressing with linear probing is used to insert the keys. Determine which key causes the table to overflow, and show the table at the point at which it does so.
   b. Assume that open addressing with quadratic probing is used to insert the keys. Determine which key causes the table to overflow, and show the table at the point at which it does so.
   c. Now assume instead that open addressing with double hashing is used to insert the keys, with the value of the second hash function based on the first character in the key: a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, g = 7, etc. Determine which key causes the table to overflow, and show the table at the point at which it does so.
3. **List-based priority queue** (8 points total)
   a. (5 points) A priority queue can be implemented using a list instead of a heap. Describe how you could use a list to implement a priority queue in which the remove operation would have a time complexity of $O(1)$.
   b. (3 points) What would be the efficiency of the insert operation in this list-based implementation?

4. **Testing for a path between vertices** (10 points total; 5 points each part)
   A graph may or may not include a path between a given pair of vertices.
   a. Given the graph representation used in the `Graph` class from lecture, write an algorithm that takes references to two `Vertex` objects and determines if there is a path from the first vertex to the second vertex. The method should return `true` if there is a path and `false` otherwise.
   b. For a graph with $n$ vertices, what would be the best-case time efficiency of the algorithm from part a? What would be the worst-case time efficiency? Use big-O notation, and explain your answers briefly.

Questions 5-7 refer to the following graph:

![Graph Image]

5. **Graph traversals** (8 points; 2 points each part)
   Suppose this graph represents an airline's routes between cities, and that you have purchased a pass that allows you to fly anywhere along these routes.
   a. List the order in which you will visit the cities if you start from Denver and do a breadth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the examples in lecture.
   b. What is the path from Denver to Boston in the breadth-first spanning tree? Give the path in the form A -> B -> C -> etc., where A, B, and C are vertices.
   c. List the order in which you will visit the cities if you start from Denver and do a depth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the examples in lecture.
   d. What is the path from Denver to Boston in the depth-first spanning tree? Give the path in the form A -> B -> C -> etc., where A, B, and C are vertices.
6. **Minimal spanning tree** (8 points)
   A cheaper version of the same pass requires that you confine yourself to flights that are part of a minimal spanning tree for the graph. List the order in which flights/edges will be added to this tree if you build it using Prim's algorithm, starting from Denver. Use the form (city1, city2) when specifying an edge.

7. **Dijkstra's shortest-path algorithm** (10 points total)
   Suppose you set out to use Dijkstra's algorithm to determine the shortest distance from Denver to every other city in the graph.
   a. (5 points) Make a table showing the order in which the cities are finalized and the minimum distance to each.
   b. (3 points) What path(s) does the algorithm discover from Denver to L.A.? Include both the final shortest path and any temporary estimates for the shortest path that are later replaced. Give the paths in the form A -> B -> C -> etc., where A, B, and C are vertices.

8. **Directed graphs and topological sort** (10 points total; 5 points each part)
   a. Is graph 8-1 a directed acyclic graph (a DAG)? If it isn’t a DAG, specify all cycles that are present in the graph. If it is a DAG, use topological sort to find one of the possible topological orderings of the vertices in the graph.
   b. Repeat the process outlined above on graph 8-2.

9. **Maximum-cost spanning tree** (8 points)
   Consider the problem of constructing a maximum-cost spanning tree. Invent a variant of Prim's algorithm to solve this problem, and specify your algorithm using pseudocode.
10. **Alternative MST algorithm** (8 points)

Prim's algorithm is just one possible algorithm for finding a minimum spanning tree. Another such algorithm was developed by Kruskal:

\[
\text{kruskal\_MST(Graph g) \{} \\
\quad \text{put each of g's vertices in its own set} \\
\quad \text{while (there is more than one set) \{} \\
\quad \quad \text{let e = the minimum-cost edge that hasn't been considered} \\
\quad \quad \text{if (e connects vertices that are in different sets) \{} \\
\quad \quad \quad \text{add e to the MST} \\
\quad \quad \quad \text{merge the sets containing the vertices connected by e} \\
\quad \quad \}\} \\
\}\}
\]

Note that this algorithm considers the edges in order of increasing cost. In addition, at an intermediate stage of the algorithm, there may be multiple trees that are not connected to each other, although they will ultimately be joined together to form a single MST.

For example, if we applied Kruskal's algorithm to the graph above, we would start out with the following sets:

\[
\{ A \}, \{ B \}, \{ C \}, \{ D \}, \{ E \}, \{ F \}
\]

We would consider the edge (C, E) first, because it has the lowest cost (400). Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

\[
\{ A \}, \{ B \}, \{ C, E \}, \{ D \}, \{ F \}
\]

We would next consider the edge (D, F), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the
tree and merge the sets involved to get:

{ A }, { B }, { C, E }, { D, F }  

We would next consider the edge (D, E), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

{ A }, { B }, { C, D, E, F }  

We would next consider the edge (C, D), because it has the smallest remaining cost. Because it connects vertices that are already in the same set, we would not add this edge to the tree.

We would next consider the edge (A, B), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

{ A, B }, { C, D, E, F }  

We would next consider the edge (B, C), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

{ A, B, C, D, E, F }  

Finally, we would consider the edge (B, D). Because it connects vertices in the same set, we would not add this edge to the tree.

Apply Kruskal's algorithm to the airline graph from earlier in the problem set, which is reproduced below. List the edges of the MST in the order in which the algorithm would add them, using the format (city1, city2) for an edge.
11. **Routing packets** (8 points)

You have been asked to hard-code a portion of the network routing table for one of the servers that your company owns. The lengths of cable connecting pairs of your company's servers are given in the matrix below; a value of -1 in a given location indicates that there is no cable connecting that pair of servers.

Assume that you are focusing on server #5. For each of the other servers, determine where server #5 should route packets destined for that server – i.e., to which other server should packets destined for that server be sent next? Your goal is to minimize the total length of cable over which the packet would need to travel, assuming that all of the other servers are also routing packets in this way. *Your answer should apply one of the graph algorithms that we covered in lecture. Show the steps* that you take in employing the algorithm to solve this problem, and *explain briefly* why it is an appropriate algorithm for this problem.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>125</td>
<td>-1</td>
<td>-1</td>
<td>325</td>
<td>500</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>125</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>150</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>140</td>
<td>120</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>140</td>
<td>-1</td>
<td>275</td>
<td>-1</td>
<td>-1</td>
<td>285</td>
</tr>
<tr>
<td>5</td>
<td>325</td>
<td>150</td>
<td>120</td>
<td>275</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>625</td>
</tr>
<tr>
<td>6</td>
<td>500</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>115</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>115</td>
<td>-1</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>285</td>
<td>625</td>
<td>-1</td>
<td>400</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Submitting Your Work**

- Go to the [homework submissions dropbox](#) (logging in as needed using the Login link in the upper-right corner, and entering your Harvard ID and PIN).

- Open the folder for ps10, and upload your `ps10_partI.txt` or `ps10_partI.pdf` into this folder.

- If you worked on one or more problem with a partner, you should click on the Comment link for the relevant files and include a comment that specifies that the name of your partner and the problems that you worked on together.

- In addition, you should click on the link for each file to view it so that you can ensure that you submitted the correct file. We will not accept any files after the fact, so please check your submission carefully.

**Note:** If you encounter problems submitting your files, close your browser and start again, or try again later if you still have time. If you are unable to submit and it is close to the deadline, email your homework before the deadline to libs111@fas.harvard.edu (with the spaces removed).