Week 8 Section Solutions

1. Sorting (MergeSort1 on CSBS)
   1st split  {29, 17, 3, 94} {46, 8, -4, 12}
   2nd split {29, 17} {3, 94} {46, 8} {-4, 12}
   3rd split {29} {17} {3} {94} {46} {8} {-4} {12}
   1st merge {17, 29} {3, 94} {8, 46} {-4, 12}
   2nd merge {3, 17, 29, 94} {-4, 8, 12, 46}
   3rd merge {-4, 3, 8, 12, 17, 29, 46, 94}

2. Graph Properties
   Graphs 1, 3, and 6 are directed. 4 and 6 are weighted. 2 and 5 are connected, and
   6 is weakly connected. Only Graph 2 is acyclic.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A - in: 0, out: 2; B - in: 2, out: 1; C - in: 1, out: 1; D - in: 2, out: 1; E - in: 2, out: 2; F - in: 2, out: 1; G - in: 2, out: 1; H - in: 2, out: 2; I - in: 0, out: 2</td>
</tr>
<tr>
<td>2</td>
<td>A - 1; B - 3; C - 1; D - 2; E - 2; F - 1</td>
</tr>
<tr>
<td>3</td>
<td>A - in: 1, out: 2; B - in: 3, out: 1; C - in: 0, out: 1; D - in: 2, out: 1; E - in: 1, out: 2</td>
</tr>
<tr>
<td>4</td>
<td>A - 2; B - 2; C - 2; D - 1; E - 1</td>
</tr>
<tr>
<td>5</td>
<td>A - 3; B - 3; C - 3; D - 3</td>
</tr>
<tr>
<td>6</td>
<td>A - in: 2, out: 2; B - in: 2, out: 3; C - in: 2, out: 3; D - in: 2, out: 0; E - in: 2, out: 2; F - in: 3, out: 2; G - in: 1, out: 2</td>
</tr>
</tbody>
</table>

3. kth Level Friends
   ```
   void kthLevelHelper(BasicGraph& graph, string v, Set<string>& result, int k) {
     if (k == 0) {
       result.add(v);
     } else {
       for (Vertex* buddy : graph.getNeighbors(v)) {
         kthLevelHelper(graph, buddy->name, result, k - 1);
       }
     }
   }
   
   Set<string> kthLevelFriends(BasicGraph& graph, string v, int k) {
     Set<string> result;
     kthLevelHelper(graph, v, result, k);
     return result;
   }
   ```

Thanks to Marty Stepp, Chris Gregg, and other CS106B and X instructors and TAs
for contributing problems on this handout.
4. isReachable

```cpp
bool isReachableHelper(BasicGraph& graph, string v1, string v2) {
    if (v1 == v2) {
        return true;
    } else {
        graph.getVertex(v1)->visited = true;
        for (Vertex* neighbor : graph.getNeighbors(v1)) {
            if (!neighbor->visited && isReachableHelper(graph, neighbor->name, v2)) {
                return true;
            }
        }
        return false;
    }
}

bool isReachable(BasicGraph& graph, string v1, string v2) {
    graph.resetData();
    return isReachableHelper(graph, v1, v2);
}
```

5. isConnected

```cpp
bool isConnected(BasicGraph& graph) {
    graph.resetData();
    for (Vertex* v1 : graph.getVertexSet()) {
        for (Vertex* v2 : graph.getVertexSet()) {
            if (!isReachable(graph, v1, v2)) {
                return false;
            }
        }
    }
    return true;
}
```
6. Popular
Set<Vertex*> popular(BasicGraph& graph) {
    Set<Vertex*> results;
    for (Vertex* v : graph.getVertexSet()) {
        // total degree and cost going out from v
        int outDegree = 0;
        int outEdgeWeight = 0;
        for (Edge* e : v->edges) {
            outDegree++;
            outEdgeWeight += e->weight;
        }
        // total degree and cost of all edges going in to v
        int inDegree = 0;
        int inEdgeWeight = 0;
        for (Edge* e : graph.getEdgeSet()) {
            if (e->finish == v) {
                inDegree++;
                inEdgeWeight += e->weight;
            }
        }
        if (inDegree >= 2 && inDegree > outDegree && inEdgeWeight > outEdgeWeight)
        {
            results.add(v);
        }
    }
    return results;
}