Week 8 Section

This week's section handout has practice with sorting and graphs. Follow along at CodeStepByStep: https://codestepbystep.com/problemset/view?id=47

---

1. Sorting (MergeSort1 on CSBS)

Trace the complete execution of the merge sort algorithm. Show the sub-vectors that are created by the algorithm and show the merging of sub-vectors into larger sorted vectors.

```
// index    0   1  2   3   4   5   6   7
           {29, 17, 3, 94, 46,  8, -4, 12};
```

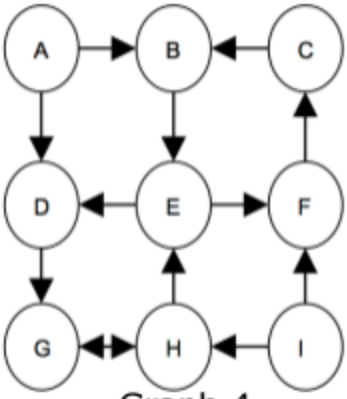
Bonus: Trace through this example with the other sorting algorithms we learned in class (QuickSort, Selection Sort, Insertion Sort, etc.).

---

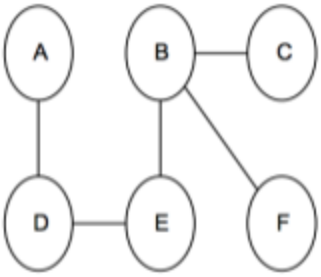
2. Graph Properties

For each of the graphs below, answer the following questions:
Is the graph directed or undirected? Is the graph weighted or unweighted? Which graphs are connected, and which are not? Is any graph strongly connected? Which graphs are cyclic, and which are acyclic? What is the degree of each vertex? (If directed, what is the in-degree and the out-degree?)

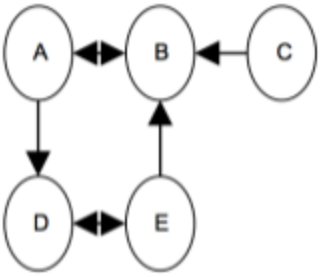
Graph 1



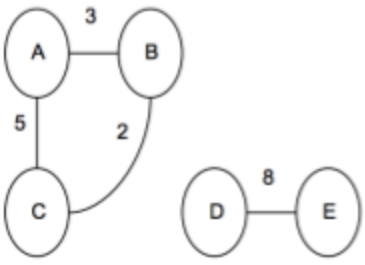
Graph 2



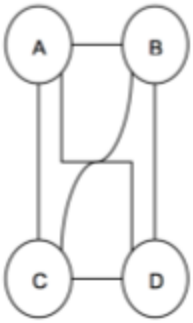
Graph 3



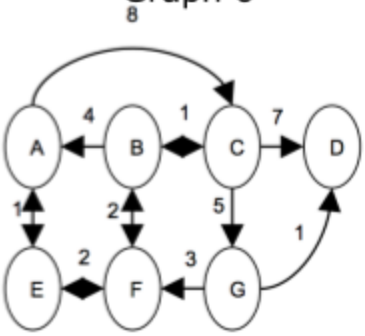
Graph 4



Graph 5



Graph 6



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

---

3. kth Level Friends

Imagine a graph of Facebook friends, where users are vertexes and friendships are edges. Write a function named **`kthLevelFriends`** that accepts three parameters: a reference to a `BasicGraph`, the string name of a vertex to start from, and an integer *K*. Your function should return a set of strings representing the set of people who are exactly *K* hops away from the given vertex (and not fewer). For example, if *K* = 1, those are the person's direct friends; if *K* = 2, they are the person's friends-of-friends.

---

4. isReachable

Write a function named **`isReachable`** that accepts three parameters: a reference to a `BasicGraph`, and two strings representing names of vertexes *v1* and *v2*. Your function should return `true` if a path can be made from the vertex *v1* to the vertex *v2*, or `false` if not. If the two vertexes are the same, return `true`. You may assume that the parameter values passed are valid.

---

5. isConnected

Write a function named **`isConnected`** that accepts one parameter: a reference to a `BasicGraph`. Your function should return `true` if a path can be made from *every* vertex to any other vertex, or `false` if there is any vertex cannot be reached by a path from some other vertex. An empty graph is defined as being connected. You may assume that the parameter values passed are valid.

---

6. Popular

Write a function named **`popular`** that accepts a reference to a `BasicGraph` as a parameter and returns a `Set` of pointers to all vertexes in that graph that are "popular" by the following definition. Suppose that the graph represents users on a social network such as Facebook. A vertex represents a user, and a directed edge from A to B represents the fact that user A "likes" user B, or is "friends" with B. The weight of the edge represents how much A "likes" B.
A user *v* is "popular" if all of the following conditions are met:

- At least 2 other users "like" *v*.
- More users "like" *v* than *v* "likes" other users. (More arrows are coming in than going out.)
- The combined weight of all "likes" toward *v* is more than the combined outbound weight of all the edges to other users that *v* "likes". (More total edge weight is coming in than going out.)
