COSC 302: Analysis of Algorithms — Spring 2018
Prof. Darren Strash

This is an *individual* assignment: collaboration (such as discussing problems and brainstorming ideas for solving them) on this assignment is highly encouraged, but the work you submit must be your own. Give information only as a tutor would: ask questions so that your classmate is able to figure out the answer for themselves. It is unacceptable to share any artifacts, such as code and/or write-ups for this assignment. If you work with someone in close collaboration, you must mention your collaborator on your assignment.

Suggested practice problems, from CLRS: 22.1-1 through 22.1-5; 22.1-6 (challenge); 22.2-3; 22.2-6; 22.-3-6; 22.3-8; 22.3-11; 22.4-2; 22.4-5; 22.5-3; 22.5-4

1. **(From problem set 5; previous exam question)** Let $A[1..n]$ be an array of non-integers taken from some set K of size $k > 1$. *(Note: For this problem, you are not given the set K or k; this is only to illustrate that there are k distinct non-integer numbers. We only have access to elements through A. Further, note that k may be small or large: from constant to even larger than n.)*

   (a) An algorithm that sorts A is described as a counting sort that iterates through the A, when a new key, k is found a node with a varible count which contains k, when then hash the key to hash table H. It is created and inserted in the binary search tree B. If In the case of key we have already found we increment $H[k]$ by one. If there is collision in the table, the key is added to a linked list. After the array is exausted, We will traverse the array in order, then access the hash table to find the amount each node's value repeats and then add them to the output array. Going through the array will take n time. Then for k disinct keys we will create a mode which will take at most lgk time. So, the entire algorithm will take $O(n + klg(k))$.

   (b) The worst-case running time for my algorithm would be $O(nlgn)$. Since there would be at most n distinct keys where there are no repeated values.

2. Let $G = (V, E)$ be an n-vertex undirected graph consisting of *cops* and *robbers* as vertices. You are given two facts about the graph:

   (a) G is connected, and

   (b) each edge is incident to exactly one cop and one robber. (That is, no edge is incident to two cops, and no edge is incident to two robbers.)

   Suppose we know that Dave $\in V$ is a cop. Give an efficient algorithm to distinguish all cops from all robbers.

   We will find all cops and robbers by using a modified breadth-first search. Since Dave $/in$ V and is a cop, there will be a set of neighbors. Since each edge is incident to only one cop and one robber. All of of Dave's neighbors would be robbers. Then each of the robber's neighbors that haven't been visited are cops ad well. And the search will reach every node since the graph is connected.

3. Problem 22.2-5 from CLRS.

4. Problem 22.5-1 from CLRS. In addition to stating your answer, also (formally) prove its correctness.

   When a edge is added the amount of strongly connected components can either stay the same if the edge is incident to two nodes in the same component or decrease by 1 if the edge is added between 2 components.

5. One DFS will not show if it is semiconnected or not. Only that it is connected. We need to do a DFS-Visit on not visited nodes to see if they're connected or not.

```
1:  function SEMICONNECTED-CHECK(G)
2:      G_R ← STRONGLY-CONNECTED-COMPONENTS(G)
3:      G_J ← TOPOLOGICAL-SORT(G_R)
4:      G_JT ← TRANSPOSE-GRAPH(G_R)
5:      DFS-VISIT(G_R,s)
6:      DFS-VISIT(G_R,s_T)
7:      for all u ∈ G_R do
8:          if u.color == WHITE then
9:              return false
10:     return true
```