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 Dave's neighbors are robbers. Suppose each vertex's status is updated in a boolean array named $Cops$ that is length n. Vertex u is either a cop which in this cays $Cops[u] == true$ or a robber which would be expressed as $Cops[u] == false$. The source vertex is a cop so we make we update the array and go through it's neighbors. With each neighbor v, we check the ancestor and then change the value to be the negation of it's ancestor. We will do this until the main queue is empty.

3. Problem 22.2-5 from CLRS. The tree in the book had two valid light edges vertex w.But t is before x in $Adj[w]$ so we get the BFS in the figure. But if x was before t in $Adj[w]$ then we could get edge (x, u) in the BFS tree.

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

   When a edge is added to a graph that contains i multiple components $V_1, V_2, V_3...V_i$ there are two cases. If the edge is added within one component, the number of components will stay the same, since adding an edge between to vertices in a component doesn't change it since it's already strongly connected. Also, if the edge is added between two components but does not create a cycle then i stays the same.

   However, in the case of the edge being added between two components does create a cycle, the amount of components can decrease to no lower than 1. So in this case, the number of edges after the edge is added i_n is $1 \leq i_n <$. Suppose the edge (u, v) is drawn between component V_1 and V_2. Note that $u \in V_1$ and $v \in V_2$. If (u, v) creates a cycle that means that there exists a path of vertices $p = u, ..., v$ so that $u \rightsquigarrow v$ and $v \rightsquigarrow u$ since cycles are strongly connected by definition. So, each component that contains a vertex that belongs in p is now part of the same component since every vertex in each component has a path to every other node, including vetices that are in p. If there are a number of components that are connected by a given cycle, then i will decrease by $a - 1$.

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:      for i := 1 → G_J.length − 1 do
5:          N ← G_J[i][1]
6:          N_2 ← G_J[i + 1][1]
7:          for all v ∈ adj[N] do
8:              if v == N_2 then
9:                  childfound = true
10:         if childfound == false then
11:             return false
12:         childfound = false
13:     return true
```