ASSIGNMENT 1: Solutions CMSC 451 (Spring 2016)

1. *Asymptotic growth.*

   For each of the following pairs of functions $f(n)$ and $g(n)$, indicate whether $f(n)$ is in O, o, Ω, ω, and/or Θ of $g(n)$. (You should either write true or false for each of these five possibilities, or give an exhaustive list of the true relationships that hold.)

   (a) $f(n) = 2^n$, $g(n) = n^2$

   Solution: Ω, ω (since $\lim_{n\to\infty} n^2/2^n = 0$, and $f(n) \in \omega(g(n)) \implies f(n) \in \Omega(g(n))$)

   (b) $f(n) = (\log n)^{\log n}$, $g(n) = n^{\log\log n^2}$

   Solution: O, o (since $g(n) = n^{\log(2\log n)} = n^{(\log\log n)+(\log 2)} = f(n)n^{\log 2}$, so $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} n^{-\log 2} = 0$)

   (c) $f(n) = 2^n$, $g(n) = 2^{2n-4}$

   Solution: O, o (since $\lim_{n\to\infty} 2^n/2^{2n-4} = \lim_{n\to\infty} 16/2^n = 0$, and $f(n) \in o(g(n)) \implies f(n) \in O(g(n))$)

   (d) $f(n) = n^{\cos n}$, $g(n) = \sqrt{n}$

   Solution: none (since $n^{\cos n}$ fluctuates between values that grow linearly in n and values that do not grow with n)

   (e) $f(n) = \log(n!)$, $g(n) = n^{1.01}$

   Solution: O, o (since $\log(n!) = \Theta(n \log n)$ by Stirling's approximation, and $\lim_{n\to\infty} \frac{n\log n}{n^{1.01}} = 0$)

2. *Finding a most common element.*

   Suppose you are given a list of n integers as input. Design an algorithm to find an integer in the list that appears the maximum number of times. (If more than one integer appears the same maximum number of times, you may output any such integer.) Your algorithm should run in time $O(n \log n)$. Prove that your algorithm is correct and that is has the desired running time.

   Solution: Sort the list using any off-the-shelf fast sorting algorithm to obtain a sorted list L in time $O(n \log n)$. Given this sorted list, we can compute the number of occurrences of each integer in $O(n)$ time and $O(1)$ space as follows. We iterate once through the sorted list and at each step store the number of occurrences of the current element seen so far and also the element seen the most number of times so far. Algorithm 1 shows the pseudocode. Correctness follows from the fact that all occurrences of any integer appear consecutively in the sorted list.

```
**Initialize:** count = 1, maxcount = 0, sort(L)
for $i$ = 1 to $n-1$ do
    if $L[i] == L[i-1]$ then
        count++
    else
        if count > maxcount then
            maxcount = count
            maxint = $L[i-1]$
        count = 1
if count > maxcount then
    maxcount = count
    maxint = $L[i-1]$
```

Algorithm 1: Finding Most Common Element

3. *Stable matching for a class project.*

   Suppose we would like to assign partners in a class of $2n$ students. Each student provides a ranking of the other $2n-1$ students, expressing his or her preferences for working with those students on a joint project. A stable matching is an assignment of partners so that no two students who are not partners would prefer to work with each other than with their assigned partners. Does a stable matching always exist? Either prove that it does or give a counterexample (with a proof that no stable matching exists for that counterexample).

   Solution: A stable matching might not exist. Since there is only one way to pair up two students, any counterexample must involve at least four students. Indeed, there is a counterexample in that case. Suppose the four students Alice, Bob, Cathy, and David have the following preferences:

Alice:	Bob, Cathy, David
Bob:	Cathy, David, Alice
Cathy:	David, Bob, Alice
David:	Bob, Cathy, Alice

   Suppose $X \in \{\text{Bob}, \text{Cathy}, \text{David}\}$ is paired with Alice. Since Alice is ranked lowest by all her potential partners, X would prefer to be with either of the alternatives. Since each of $\{\text{Bob}, \text{Cathy}, \text{David}\}$ is ranked first by someone in that set, there is some $Y \in \{\text{Bob}, \text{Cathy}, \text{David}\}$ that ranks X first. Students X, Y are not paired (since X is paired with Alice) but they would prefer to be with each other than with their assigned partners. Hence no matching can be stable.

4. *Stable matching with multiple internships.*

   In the simple version of the stable matching problem analyzed in class, we assumed that each company had exactly one internship available, and that there were as many internships as students. More realistically, suppose each company wants to hire some specified number of interns, and there are more students looking for internships than the total number of available positions. As before, each company has a ranking of all students and each student has a ranking of all companies. We call an assignment of students to companies *unstable* if either

   (i) student S is assigned to company C and student S' is assigned to company C', but S prefers C' to C and C' prefers S to S'; or

(ii) student S is assigned to company C and student S' is not assigned to any company, but C prefers S' over S.

Design an algorithm to find a stable matching, and prove its correctness.

Solution: A stable assignment always exists and can be found by a variant of the Gale-Shapley algorithm. Algorithm 2 generalizes the standard stable-marriage algorithm for this case. Let $\mathcal{S}$ and $\mathcal{C}$ denote the set of all students and all companies respectively. For a company $C \in \mathcal{C}$, let k_C denote the number of students that C is looking to hire.

```
Initially all students $S \in \mathcal{S}$ are free
Initially every company $C \in \mathcal{C}$ has an empty waiting list with $k_C$ slots
while there exists a student $S$ who is free and hasn't proposed to every company do
    Choose such a student $S$
    Let $C$ be the highest-ranked company in $S$'s preference to whom $S$ hasn't proposed yet
    if $C$'s waiting list has an empty slot then
        $S$ is added to $C$'s waiting list
    else
        Say $S'$ is the least preferred student on $C$'s waiting list
        if $C$ prefers $S'$ to $S$ then
            $S$ remains free
        else
            $S$ is added to $C$'s waiting list
            $S'$ becomes free
Return the set of all $(S, C)$ where $S$ is a student in $C$'s waiting list
```

Algorithm 2: Stable Matching with Multiple Internships

Claim 1. *In the assignments returned by Algorithm 2, every company C is assigned exacty k_C students. The set of pairs returned by Algorithm 2 is a matching such that all companies $C \in \mathcal{C}$ are matched.*

Proof. Once a company's waiting list gets full, it always stays full. Suppose for the sake of contradiction that a company C is assigned fewer than k_C students. Since there are more students than the total number of available positions, there exists a free student S in the final assignment. S must have proposed to C during the execution of the algorithm. However this implies that either S should be assigned to C in the final assignment or C should have no free slots in the waiting list and hence we have a contradiction. □

Claim 2. *The set of pairs returned by Algorithm 2 is a stable assignment.*

Proof. Suppose for the sake of contradiction that the assignment returned by Algorithm 2 is unstable due to an instability of type (i), i.e., we have assignments (S, C) and (S', C'), but S prefers C' to C and C' prefers S to S'. During the execution of the algorithm, we know that S's last proposal was to company C. But, since S prefers C' to C, S must have proposed to C' earlier. This implies that C' rejected S and let S'' be the least preferred student on C''s waiting list at this iteration. Now, since S' is finally assigned to C', we know that either $S' = S''$ or C' prefers S' to S''. In either case, this implies that C' prefers S' to S that contradicts our assumption.

On the other hand, suppose for the sake of contradiction that the assignment is unstable due to an instability of type (ii), i.e., we have an assignment (S, C) and S' is free, but C prefers S' to S. During the execution of the algorithm, S' must have proposed to company C at some iteration. Now, suppose that S has proposed to C at an earlier iteration, then since C prefers S' to S, C should remove S from its waiting list before removing S'. This is a contradiction since S and C are matched. On the other hand, say S proposes to C at a later iteration and S gets engaged to C. This implies that C prefers S to S' and we again have a contradiction. □

5. *A characterization of trees.*

   Prove that a graph is a tree if and only if it has a unique simple path between any pair of vertices. (In your proof, you should be sure to explicitly prove any claim that does not follow immediately from the definitions.)

   Solution: We separately prove each direction of the implication.

   Only if: A tree is connected, so there is at least one simple path between each pair of vertices. Suppose for the sake of contradiction that there are vertices s and t having two (or more) simple paths between them. Let $(s = u_1, u_2, \ldots, u_l = t)$ and $(s = v_1, v_2, \ldots, v_m = t)$ denote the two paths. Let j be the largest index such that $u_i = v_i$ for all $i \leq j$, i.e., let $u_j = v_j$ be the vertex where the two paths first *split*. By definition, we know that $u_{j+1} \neq v_{j+1}$. Now consider the (possibly non-simple) path $(u_j, u_{j+1}, \ldots, u_l = t = v_m, v_{m-1}, \ldots, v_{j+1})$. This path must contain a simple path from u_j to v_{j+1} that can be completed to form a cycle by adding the edge (v_j, v_{j+1}). But this is a contradiction and hence there can only be a single simple path between any pair of vertices.

   If: Since there is a path between any pair of vertices, the graph is connected. Suppose that the graph is not a tree and contains a cycle $C = (u_1, u_2, \ldots, u_l, u_1)$. Now the vertices u_1 and u_l have at least two distinct paths: one consisting of the single edge (u_1, u_l) and the other consisting of the walk around the cycle $(u_1, u_2, \ldots, u_l)$.