Reminder: Turn in each problem as a separate document as indicated in the instructions.

The following paragraphs are taken in large part from the course website, but are worth repeating here for your first graded problem set.

Prelude: Problem sets are where much of the real learning happens in a course like this one (but come to class anyway). This is how you find out what you know, what you don’t yet know, and what you need to study further. To be effective, problem sets should be challenging, interesting, occasionally frustrating, and fun. If you believe the problem sets are failing in any of those respects, let me know. If you are enjoying them and/or finding them helpful learning experiences, I’d like to know that, too!

Instructions: Answer all questions. Even if you do not believe you can completely answer the question, give it your best effort: Describe clearly what you were attempting to do and how/why you got stuck. Doing this will work to your advantage, grade-wise, and, more importantly, will often help you to make further progress. Problems such as the first one require only a short justifications; in general, problems will usually require either a proof or a counter-example. Visualizations (graphs, diagrams, etc) are welcome components of solutions. They can be very effective methods of clarifying a concept, but they generally do not themselves constitute a proof—and can often be misleading. Similarly, pseudo-code can be a helpful addition in describing an algorithm you’ve developed, but it does not replace a clear, concise textual description of the algorithm. The proofs given in your text should provide good models of proof presentation style.

Fine Print (but read it anyway): Problems will be graded on the correctness, clarity, thoroughness, and appropriateness of responses. And they should be easy to read! Use complete sentences and organize your work reasonably neatly on the page. Solutions should be written using LATEX as described at [here](#). Instructions on submitting solutions can be found [here](#). And remember: No sources beyond the course materials (textbook, class notes, other assigned readings) may be consulted when working on assignments. For example, internet searches, use of solutions from other offerings of this or other courses, and similar aids, are prohibited. You may collaborate with other students in the course on problems but any collaboration must be cited and must write your solutions on your own.

**Question 1.** Take the following list of functions and arrange them in ascending order of growth rate. That is, if function \( g(n) \) immediately follows function \( f(n) \) in your list, then it should be the case that \( f(n) = O(g(n)) \). Please prove your claims.

1. \( f_1(n) = 2\sqrt{(\log n)} \)
2. \( f_2(n) = 2^n \)
3. \( f_3(n) = n^{4/3} = 2^{4/3\log n} \)
4. \( f_4(n) = n(\log n)^3 \)
5. \( f_5(n) = n^{\log n} = 2^{(\log n)^2} \)
6. \( f_6(n) = 2^{2^n} \)
7. \( f_7(n) = 2^{n^2} \)

**Question 2.** Decide whether you think the following statements are true or false. If a statement is true, give a short explanation why it’s the case. If it’s false, give a counterexample.

(a) In every instance of the Stable Matching Problem there is a stable matching containing a pair \((m, w)\) such that \(m\) was ranked first on the preference list of \(w\) and \(w\) was ranked first on the preference list of \(m\).

(b) Consider an instance of the Stable Matching Problem in which there exists a man \(m\) and a woman \(w\) such that \(m\) is ranked first on the preference list of \(w\) and \(w\) is ranked first on the preference list of \(m\). Then the pair \((m, w)\) belongs to every possible stable matching for this instance.
**Question 3.** Consider the following variant of the stable marriage problem. There are $2n$ people, each of whom completely ranks the other $2n - 1$ people in order of preference (with no ties). For example, Alice, Bob, Carlos, and Don might have the following rankings:

<table>
<thead>
<tr>
<th>Name</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob, Carlos, Don</td>
</tr>
<tr>
<td>Bob</td>
<td>Alice, Carlos, Don</td>
</tr>
<tr>
<td>Carlos</td>
<td>Don, Alice, Bob</td>
</tr>
<tr>
<td>Don</td>
<td>Carlos, Alice, Bob</td>
</tr>
</tbody>
</table>

The goal is to find a stable perfect matching: i.e., $n$ pairs of people such that no two people prefer each other to their current matches. We saw in class that a stable perfect matching always exists for instances of the stable marriage problem. Do stable matchings always exist for this problem? If so, provide a proof. If not, give a counterexample.

**Question 4 (Algorithms in the wild).** The National Resident Matching Program (NRMP) matches medical students with residency programs. Here is what the NRMP says on their website:

The NRMP conducts its Matches using a mathematical algorithm that pairs the rank ordered preferences of applicants and program directors to produce a best fit for filling available training positions. Research on the NRMP algorithm was a basis for Dr. Alvin Roth’s receipt of the 2012 Nobel Prize in Economics.

In this matching problem, there are $n$ students and $m$ hospitals. Each hospital $h_i$ has $p_i$ available positions. Each student ranks the $m$ hospitals and each hospital ranks the $n$ students. Since there are more students than total positions available, we assume that $n > \sum_{i=1}^{m} p_i$. Thus, some students are never matched. As a result, we need a slightly expanded version of stability. As before, the matching is unstable if

- $s$ is assigned to $h$ and $s'$ is is assigned to $h'$ but $s$ prefers $h'$ to $h$ and $h'$ prefers $s$ to $s'$.

But it is also unstable if

- $s$ is assigned to $h$ and $s'$ is not assigned to a hospital but $h$ prefers $s'$ to $s$.

Give an algorithm to find a stable matching of students to hospitals where every hospital position is filled with a student. Show that your algorithm is correct and that it runs in time polynomial in $n$ and $m$. Your algorithm description and analysis should be clear and concise.

**Question 5.** Free-style: Suppose you are given a set $S$ of $n$ intervals—that is, $n$ pairs $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ of numbers with each $s_i < t_i$.

(a) Design an algorithm to find the maximum sized subset of $S$ such that every pair of intervals in the subset overlap and prove that your algorithm works correctly in all cases.

Suggestion: Draw some pictures. See if you can identify any helpful properties that hold for any subset of intervals such that all of the intervals in the subset intersect pairwise (if you use such a property, though, you must prove that it holds!). Did I say “Draw some pictures”? Yes. Yes, I did. Imagine processing the intervals in, say, left-to-right order.

(b) Describe an implementation of your algorithm, including any appropriate data structures, and determine its time and space requirements. Pseudo-code can be helpful, but is insufficient on its own (and is also not required). A clear, concise prose description of an algorithm should be considered a necessity for your response to be complete.