Peer Assessment in TCSS 343

What is peer assessment?

Peer assessment is the analysis, critique, and evaluation of the work of peers. In this course, this means that you will evaluate some of your classmates' work and you will be evaluated by your classmates on some of your work.

Why do peer assessment?

In the world of industry, peer assessment is a standard way to achieve high quality. In programming and engineering jobs, managers or other engineers will review each other's code to look for obvious bugs, design flaws, missing cases, bad comments, poor variable names, etc. The reason that this works is because multiple pairs of eyes and brains (that is, different perspectives) help to find errors better than just one person can. In design meetings, groups of engineers will discuss different proposals for getting a job done, and so it is important that the engineers are able to make judgments about different ideas. It does not matter whether you are Bill Gates or Joe Millionaire – ideas are judged on their own merit, not by who says them.

In the world of academia, peer assessment happens all the time. Whenever someone proposes some idea and defends it in a conference or some other public forum, it undergoes great scrutiny, criticism, and evaluation. Sometimes the discussions and debates can get heated. Is the idea important? Were the conclusions the result of a rigorous line of reasoning? Was the data obtained in a sound way? Is there any bias? What facts or considerations were overlooked?

Professionals do peer assessment not because they are mean and they want to put other people down. There is a serious purpose behind providing high-quality evaluation of other people's work. Critical review and public debate is the way knowledge and science advances.

In this course, you will experience this process by directly participating in it. You will provide work that will be evaluated by others, and you will evaluate other people's work. It will not be enough to just memorize facts. You will have to understand concepts and apply them. You will have to judge whether someone else's understanding is adequate, fair, or poor.

How will peer assessment be done in TCSS 343?

Four times this quarter, there will be a problem that is designated to be the peer assessment problem. Half of the groups in the class will solve that week's problem. By the end of the course, every group will have solved two peer problems. The solution should be as complete as possible, so that someone who reads it can understand it (not just the instructor). In about a week, they will submit their solutions to the problem...
electronically. The instructor will then post the solutions, with the names of the students erased, on the course web site. In about one more week, each group (including the ones who solved the problem) will read the solutions and provide an evaluation and critique of the solutions. The critique will also include a numerical score (0 to 10). **Both the solutions and the critiques need to be submitted electronically and by hardcopy.**

After the critiques have been turned in, we will have a discussion in class about the solutions.

Both the solutions and the critiques will be graded by the instructor. The numerical scores provided by the critiques of the class will contribute a small percentage (about 10%) to the score for that problem of the students who provided the original solutions. Note that the peer assessment schedule is slightly different than the normal homework schedule, so you need to always look at the course schedule to be aware of the deadlines. It is clearly important that both the solutions and the critiques be turned in on time, so **no late submissions regarding peer assessment assignments will be allowed.**

You are free to discuss the assignments with your groupmates, but you may NOT discuss the problems with other classmates. You MAY discuss the problem with others not taking the class (for example, mentors, students who have taken the class before). If you consult someone other than your groupmates, you must acknowledge the help of anyone you discuss the problem with and you must follow the spirit of the *Gilligan's Island* rule (throw away notes you generate, wait an hour before writing up your final solution or critique).

**How to Evaluate Someone Else's Work**

In some ways, it is much more difficult to evaluate someone else's work than to create your own solution. Why? Because you not only have to understand how to solve the problem in your own way, but also be open to the possibility that there are other correct ways to solve the problem. As the saying goes, "There's more than one way to skin a cat." If you discover that the solution someone gives is not quite right, it isn't enough to say that it is wrong. You also need to be able to explain why it is wrong.

There is no recipe for how to evaluate someone else's work, but here are some tips to help you understand what it means to do so.

1. **Understand the question being asked or problem being solved.** If you don't understand the problem, how can you know if someone's solution is correct? You probably will need to solve the problem yourself before you are able to evaluate someone else's answer.

2. **Understand what a correct answer could look like.** For example, if the problem is to devise an algorithm to sort an \( n \)-element array in \( O(n \log n) \) time, then a solution would be a clear description of an algorithm along with some sort of analysis that explains or proves that the algorithm is correct and that it runs in \( O(n \log n) \) time.
3. **Be on the lookout for faulty arguments.** Here are some examples.

a. The solution does not answer the question being asked.
   
   **Example:** What is the worst case running time of quicksort?
   
   **Bad answer:** If the pivot is always chosen so that the two partitions are of equal size, then the worst case running time of quicksort is $T(n) = 2T(n/2) + n$ and $T(1) = 1$. So, $T(n)$ is $O(n \log n)$.
   
   (This answers how quicksort behaves in the best case, not the worst case.)

b. The solution makes false claims.
   
   **Example:** Let $f(n) = 4n^2 \log n$. Is $f(n) \in O(n \log^3 n)$?
   
   **Bad answer:** Since $4n^2 \log n \in O(n \log^2 n)$ and $n \log^2 n \in O(n \log^3 n)$, then by transitivity of big-Oh, $4n^2 \log n \in O(n \log^3 n)$.
   
   (In the bad answer above, it is not true that $4n^2 \log n \in O(n \log^2 n)$.)

   **Another bad answer:** No, because $\log n \notin O(\log^3 n)$.
   
   (In this bad answer, the conclusion is right, but the reason is false. Furthermore, it is not clear why the claim $\log n \notin O(\log^3 n)$, even if it were true, would show that $f(n) \notin O(n \log^3 n)$.)

c. The solution is missing a link in the chain of reasoning.
   
   **Yet another bad answer to the above problem:** No, because $n^2 \log n / n \log^3 n = n / \log^2 n$.
   
   (The reason given is actually true, but there is no explanation for why the reason is important.)

d. The solution is missing a case in the analysis.
   
   **Example:** What is the running time of the following algorithm in the worst case?
   
   ```java
   public static boolean TwoNPlus1(int n) {
   if (n == 1) {
   return true;
   } else if (n % 2 == 0) {
   return TwoNPlus1(n/2);
   } else {
   return TwoNPlus1(2*n + 1);
   }
   }
   ```
   
   **Bad answer:** If $n$ is a power of two, then the algorithm will call itself with $n/2$ as an argument and recurse until the method is called with $n=1$, when it will halt. After the $k$th level of recursion, the argument to the method will be $n/2^k$. Therefore, it will take $O(\log n)$ time to return true.
   
   (What if $n$ is not a power of two?)

e. The solution has insufficient detail or justification.
   
   **Example:** Show that $n^{\log n} \in O(2^n)$.
   
   **Bad answer:** $n^{\log n}$ grows more slowly than $2^n$.
   
   (This answer basically restates the problem.)
4. **Pay attention to detail.** Sloppy arguments are like sloppy code; they are "more or less" correct, but one or two minor mistakes can result in false claims or code that crashes in critical situations.

*Example:* Prove by induction that $2^n \geq n + 1$, for all $n \geq 0$.

*Bad answer: Claim:* $2^n \geq n + 1$, for all $n \geq 0$.

The proof of the claim is by induction on $n$.

**Basis step:** $n=0$. Then $2^0 = 1 \geq 0 + 1$. So, $2^n \geq n + 1$ for $n = 0$.

**Induction step:** Suppose the claim is true for all $0 \leq k \leq n$, for some fixed $n \geq 0$.

Then

\[
2^{n+1} = 2 \cdot 2^n \\
\geq 2 \cdot (n+1) \quad \text{(by the induction hypothesis)} \\
= 2n + 2 \\
\geq n + 1
\]

Therefore, $2^{n+1} > n + 1$, and so $2^{n+1} \geq n + 1$. So, the claim is true for $n+1$. This concludes the proof by induction.

(This proof is almost correct. See if you can figure out what exactly is missing from the proof. Another thing to think about: Why is that the solution above concluded that $2^{n+1} > n + 1$ from the chain of equalities and inequalities and not conclude $2^{n+1} \geq n + 1$ directly? The answers to these questions are on the next page.)

5. **Acknowledge parts of the work that could lead to a correct solution.** Evaluating someone else's work is not all about pointing out where errors are. It's also about recognizing those parts of the work that are constructive. For example, in the above proof by induction problem, the solution is almost completely correct, except for one detail regarding the justification of one of the steps in the chain of inequalities.
What is the answer under "Pay attention to detail" missing? Generally speaking, algebraic manipulations do not need to be justified. For example, $2^{n+1} = 2 \cdot 2^n$ is a simple algebraic manipulation and need no further comment. However, the claim a couple lines down that $2n + 2 > n + 1$ needs some sort of justification, because in general, it's not true! For example, if $n$ is $-1$, then the statement is not true. The inequality $2n + 2 > n + 1$ is true when $n \geq 0$, so there should be a justification to the right of that line that says that $n \geq 0$. This may seem like a trivial point, and it is true that it is subtle, but in more complicated arguments, such details are sometimes critical to the validity of the argument.

Now, the really astute reader should be asking, "If we had to justify $2n + 2 > n + 1$ with the fact that $n \geq 0$, why don't we have to use the fact that $n \geq 0$ for the step in the induction step that uses the induction hypothesis?" Technically speaking, we do, since we need to verify that the value of $n$ in the application of the induction hypothesis is in a valid range of the hypothesis. However, the solution has "set up" the application of the induction hypothesis so that $n$ is in a valid range. So, the justification that the induction hypothesis is being used is sufficient.

What about the question of why the solution could only conclude that $2^{n+1} > n + 1$ from the chain of equalities and inequalities and not conclude $2^{n+1} \geq n + 1$ directly? The chain of inequalities of the solution are of the form $a = b \geq c = d > e$. When you create such a chain, generally you try to conclude the strongest statement you can make. In this case, the strongest statement we can make is that $a > e$. Then, we can observe that if $a > e$, then $a \geq e$ (why?).

Writing an Assessment/Critique

Writing a critique involves understanding what the problem is asking for, what the author is trying to say, determining whether the approach he or she takes is one that is on the right track, and then pointing out the errors (if any) in the facts presented or in the line of reasoning. The examples above give you an idea of the kinds of things to be watchful of. When writing a critique of someone's work, be clear about what you think is right or wrong and why you think it is right or wrong.

A useful way to approach the evaluation of the work is to classify errors in the work as either major errors or minor errors. Major errors are errors where the concept that was missed was critical to the explanation or correctness of the problem. Examples (a) through (d) in item 3 above ("Be on the lookout for faulty arguments") are examples of major errors. Minor errors are technically incorrect, but are not central to the main argument or solution. They typically can be "fixed" with a small modification to the original solution. Example (e) is an example of a minor error.

Note that an error that would be considered a major error in one context could be considered a minor error in another. For example, in a problem that requires the use of quicksort to solve a larger problem, one might make the claim that quicksort runs in worst case time $O(n \log n)$, using the line of reasoning in example (a) above. However, the use of quicksort might be the correct solution to the larger problem, and so the error
in running time analysis is a small part of the bigger problem. In contrast, in a problem where the main point of the problem is to analyze quicksort (as in example (a)), this would be a major error – or even more than a major error, since that is the point of the problem.

Some minor errors are more minor than others. For example, in a coding example, the condition in an if statement might say "\(<=\)" when it really should be "\(<\)". If it said "\(>\)", then it could be a major or minor error, depending on how important that part of the code was to the correctness of the algorithm.

In the "real world" errors are errors: a major error or a minor error will still cause your program to run incorrectly. They still must be identified and corrected. However, it is useful to classify errors as either major or minor in order to judge the relative importance of the concepts involved.

When you write your critique, divide it into three sections. The first section should be an overview of the solution being critiqued (how they approached it and a general assessment of whether the approach is right or wrong). The second section should be a list of errors, along with a classification as to whether the error is major or minor. The third section should be a score between 0 and 10, along with a brief justification of your score.

Assigning a Number

When you critique work in this course, you will assign a number from 0 to 10 that represents the quality of the work. The general scheme that we will use is as follows:

<table>
<thead>
<tr>
<th>Score</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-10</td>
<td>correct solution. 8 and 9 are reserved for solutions that have a few minor errors.</td>
</tr>
<tr>
<td>7</td>
<td>mostly correct. The solution has most of the correct elements of a correct answer, but has several minor errors or one major error.</td>
</tr>
<tr>
<td>6</td>
<td>not quite right, but very close. The solution has some of the correct elements of a correct answer or displays some key fact or idea needed for a correct solution, but there are too many missing links in the chain of reasoning or the chain of reasoning itself is faulty.</td>
</tr>
<tr>
<td>3-5</td>
<td>not close to correct. The solution displays some relevant facts, but there is no coherent line of reasoning.</td>
</tr>
<tr>
<td>0-2</td>
<td>definitely incorrect. Little or no demonstration of understanding of the problem or what is needed for a correct solution.</td>
</tr>
</tbody>
</table>

Different problems will have different ideas that are important. The point of this guideline is to provide a common basis to evaluate other people's work.