Rubrics for CS 374 Fall 2022

September 28, 2022 Version: **1.0**

The following are modified rubrics based on Jeff's rubrics. Thanks Jeff!

Jeff: These are the standard rubrics we used in CS 374 in Fall 2021, with a few proposed changes that I plan to use the next time I teach the course. Individual problems sometimes vary from these rubrics, of course.

Rubric: Guidelines

- These rubrics are intended both as a description for students of what we expect in their solutions, and instructions to graders for rewarding partial credit. I strongly encourage other instructors to modify these rubrics to match their own priorities and goals, but I do recommend keeping the same level of detail.

- Each bullet point should be appear as a separate rubric item in Gradescope, so that we can track exactly which mistakes students are making. I recommend using ***positive*** grading on Gradescope (adding points for each component) rather than negative grading (subtracting points for each mistake).

- Rubrics had been modified to be out of 100 points per question, as specified in the constitution. Note that this value can be changed by modified by changing the value of `\TotalPoints` in the LaTeX document.

- Each of these standard rubrics describe partial credit for problems worth 100 points. Partial credit for (sub)problems worth less than 100 points should be scaled appropriately. I recommend rounding scaled partial credit to the nearest half-integer.

- Most rubrics can be summarized roughly as follows:

  +20 points for an answer with the correct *syntax*.

  +40 points for an answer with the correct *meaning*.

  +40 points for an English explanation or proof.

- Official solutions for algorithm design questions will provide target time bounds. Faster algorithms are worth more points, and slower algorithms are worth fewer points, typically by 20 or 30 points (out of a 100) for each factor of n in either direction. Partial credit is ***scaled*** up or down to the new maximum score, and all points above 100 (for algorithms that are faster than our target time bound) are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we do include them, significantly more students submit incorrect algorithms with the target running time (earning 0/100) instead of correct algorithms that are slower than the target (earning 70/100).

Rubric: Standard induction rubric

100 points =

+10 for explicitly considering an *arbitrary* object.

- + 20 for a valid ***strong*** induction hypothesis
  - ⋆ **Deadly Sin!** No credit here for stating a weak induction hypothesis, unless the rest of the proof is *absolutely perfect*.
- + 20 for explicit exhaustive case analysis
  - ⋆ No credit here if the case analysis omits an infinite number of objects. (For example: all odd-length palindromes.)
  - ⋆ −10 if the case analysis omits an finite number of objects. (For example: the empty string.)
  - ⋆ −10 for making the reader infer the case conditions. Spell them out!
  - ⋆ No penalty if the cases overlap (for example: even length at least 2, odd length at least 3, and length at most 5.)
- + 10 for cases that do not invoke the inductive hypothesis ("base cases")
  - ⋆ No credit here if one or more "base cases" are missing.
- + 20 for correctly applying the ***stated*** inductive hypothesis
  - ⋆ No credit here for applying a ***different*** inductive hypothesis, even if that different inductive hypothesis would be valid.

+20 for other details in cases that invoke the inductive hypothesis ("inductive cases")

- ⋆ No credit here if one or more "inductive cases" are missing.

Rubric: Standard regular expression rubric

100 points =

- 20 points for a syntactically correct regular expression.
- **Homework only:** 40 points for a *brief* English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - ⋆ For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - ⋆ We do not want a *transcription*; don't just translate the regular-expression *notation* into English.
- 40 points for correctness. (80 points on exams, with all penalties doubled)

  −40 for incorrectly answering $\varnothing$ or Σ^*.

  −10 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language.

  −20 for incorrectly including/excluding more than one but a finite number of strings.

  −40 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are more complex than necessary may be penalized. Regular expressions that are *significantly* too complex may get no credit at all. On the other hand, minimal regular expressions are *not* required for full credit.

Rubric: Standard DFA/NFA design rubric

100 points =

+20 points for an unambiguous description of a DFA or NFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.

- **Drawings:**
  - Use an arrow from nowhere to indicate the start state s.
  - Use doubled circles to indicate accepting states A.
  - If $A = \varnothing$, say so explicitly.
  - If your drawing omits a junk/trash/reject state, say so explicitly.
  - **Draw neatly!** If we can't read your solution, we can't give you credit for it.
- **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - You must explicitly specify $\delta(q, a)$ for *every* state q and *every* symbol a.
  - If you are describing an NFA with ϵ-transitions, you must explicitly specify $\delta(q, \epsilon)$ for *every* state q.
  - If you are describing a DFA, then every value $\delta(q, a)$ must be a single state.
  - If you are describing an NFA, then every value $\delta(q, a)$ must be a set of states.
  - In addition, if you are describing an NFA with ϵ-transitions, then every value $\delta(q, \epsilon)$ must be a set of states.
- **Product constructions:** You must give a complete description of each of the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA. In particular, we will not assume that product constructions compute intersections by default.

+40 **Homework only:** points for *briefly* explaining the purpose of each state *in English*. This is how you argue that your DFA or NFA is correct.

- In particular, each state must have a mnemonic name.
- For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
- Yes, we mean it: A perfectly correct drawing of a perfectly correct DFA with no state explanation is worth at most 6 points.

+40 points for correctness. (80 points on exams, with all penalties doubled)

−10 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
−40 for incorrectly accepting every string, or incorrectly rejecting every string.
−20 for incorrectly accepting/rejecting more than one but a finite number of strings.
−40 for incorrectly accepting/rejecting an infinite number of strings.

- DFAs or NFAs that are more complex than necessary may be penalized. DFAs or NFAs that are *significantly* more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are *not* required for full credit, unless the problem explicitly asks for them.

- Half credit for describing an NFA when the problem asks for a DFA.

Rubric: Standard fooling set rubric

100 points =

+40 points for the fooling set:

- + 20 for explicitly describing the proposed fooling set F.
- + 20 if the proposed set F is actually a fooling set for the target language.
- − No credit for the proof if the proposed set is not a fooling set.
- − No credit for the *problem* if the proposed set is finite.

+60 points for the proof:

- ∘ The proof must correctly consider *arbitrary* pairs of distinct strings $x, y \in F$.
  - − No credit for the proof unless both x and y are *always* in F.
  - − No credit for the proof unless x and y can be *any* pair of distinct strings in F.
- +20 for explicitly describing a suffix z that distinguishes x and y.
- +20 for proving either $xz \in L$ or $yz \in L$.
- +20 for proving either $yz \notin L$ or $xz \notin L$, respectively.

Rubric: Standard language transformation rubric

For problems worth 100 points:

+20 for a formal, complete, and unambiguous description of the output automaton, including the states, the start state, the accepting states, and the transition function, as functions of an *arbitrary* input DFA. The description must state whether the output automaton is a DFA, an NFA without ϵ-transitions, or an NFA with ϵ-transitions.

- No points for the rest of the problem if this is missing.

+20 for a *brief* English explanation of the output automaton. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?

- **Deadly Sin:** No points for the rest of the problem if this is missing.

+60 for correctness

  +30 for accepting *all* strings in the target language

  - But no credit for incorrectly accepting every string in Σ^*

  +30 for accepting *only* strings in the target language

  - But no credit for incorrectly rejecting every string in Σ^*

  −10 for a single mistake in the formal description (for example a typo)

- Double-check correctness when the input language is $\varnothing$, or $\{\epsilon\}$, or $\texttt{0}^*$, or Σ^*.

Rubric: General instructions for presenting algorithms

Full credit for ***every*** algorithm in the class requires a clear, complete, unambiguous description, at a sufficient level of detail that a strong student in CS 225 could implement it in their favorite programming language (which you don't know), using a software library containing every algorithm and data structure we've seen in CS 124, 173, 225, and previously in 374. In particular:

- Watch for hand-waving, pronouns (especially "this") without clear antecedents, meaningless filler (like "go through the array and"), and the Deadly Sin "repeat this process".
- Do not regurgitate algorithms we've already seen. We've read the book.
- Every algorithm must be clearly specified in English, unless the specification is already given *precisely* in the problem statement. In particular, if the algorithm solves a more general problem than requested, the more general problem must be specified explicitly. Similarly, if the algorithm assumes any conditions that are not explicit in the given problem statement, those assumptions must be stated explicitly.
- The meaning of every variable must be either clear from context (like n for input size, or i and j for loop indices) or specified explicitly.

Rubric: Standard dynamic programming rubric

100 points =

+30 points for a clear and correct English description of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)

−10 for naming the function "OPT" or "DP" or any single letter.

- No credit if the description is inconsistent with the recurrence.
- No credit if the description does not explicitly describe how the function value depends on the named input parameters.
- No credit if the description refers to internal states of the eventual dynamic programming algorithm, like "the current index" or "the best score so far". The function must have a well-defined value that depends *only* on its input parameters (and constant global variables).
- An English explanation of the *recurrence* or *algorithm* does not qualify. We want a description of *what* your function returns, not (here) an explanation of *how* that value is computed.

+40 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.

+10 for base case(s). $-\frac{1}{2}$ for one *minor* bug, like a typo or an off-by-one error.

+30 for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error.

−20 for greedy optimizations without proof, even if they are correct.

- **No credit for the rest of the problem if the recursive case(s) are incorrect.**

+30 points for iterative details

+ 10 for describing an appropriate memoization data structure

+ 10 for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested for loops, be sure to specify the nesting order.

+ 10 for correct time analysis. (It is not necessary to state a space bound.)

- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem specifically says otherwise.

- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but iterative pseudocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. But you ***do*** still need and English description of the underlying recursive function (or equivalently, the contents of the memoization structure). ***Perfectly correct iterative pseudocode, with no explanation or time analysis, is worth at most 6 points out of 10.***

- Partial credit for incomplete solutions depends on the running time of the ***best possible*** completion (up to the target running time). For example, consider a solution that contains *only* a clear English description of a function, with no recurrence or iterative details. If the described function *can* be developed into an algorithm with the target running time, the solution is worth 3 points; however, if the function leads to an algorithm that is slower than the target time by a factor of n, the solution could be worth only 2 points (= 70% of 3, rounded).

Rubric: Standard graph-reduction rubric (proposed)

100 points =

+ 30 for constructing the correct graph.

  +10 for correct vertices

  +10 for correct edges

  −5 for forgetting "directed" if the graph is directed

  +10 for correct weights, costs, labels, or other annotations, if any

  - The vertices, edges, and so on must be described as explicit functions of the input data.
  - For most problems, the graph can be constructed in linear time by brute force; in this common case, no explicit description of the construction algorithm is required. If achieving the target running time requires a more complex algorithm, that algorithm will graded out of 50 points using the appropriate standard rubric, and all other points are cut in half.

+ 30 for explicitly relating the given problem to a specific **problem** involving the constructed graph. For example: "The minimum number of moves is equal to the length of the shortest path in G from the start vertex $(0,0,0)$ any target vertex of the form $(k,\cdot,\cdot)$ or $(\cdot,k,\cdot)$ or $(\cdot,\cdot,k)$." or "There is a French-flag walk from s to t in G if and only if $(s,0)$ can reach $(t,0)$ in H."

  - No points for just writing (for example) "shortest path" or "reachability". Shortest path in which graph, from which vertex to which other vertex? How does that shortest path related to the given problem?
  - No points for only naming the algorithm, not the problem. "Breadth-first search" is not a problem!

+ 20 for correctly applying the correct black-box **algorithm** to solve the stated problem. (For example, "Perform a single breadth-first search in H from $(0,0,0)$ and then examine every target vertex." or "Whatever-first search in H.")

  −10 for using a slower or more specific algorithm than necessary, for example, breadth- or depth-first search instead of whatever-first search, or Dijkstra's algorithm instead of breadth-first search.

  −10 for explaining an algorithm from lecture or the textbook instead of just invoking it as a black box.

+ 20 for time analysis in terms of the *input* parameters (not just the number of vertices and edges of the constructed graph).

★ An extremely common mistake for this type of problem is to attempt to modify a standard algorithm and apply that modification to the input data, instead of modifying the input data and invoking a standard algorithm as a black box. This strategy can work in principle, but it is much harder to do it correctly, and it is terrible software engineering practice. ***Clearly correct*** solutions using this strategy will be given full credit, but partial credit will be given only sparingly.

Rubric: Standard NP-hardness rubric (proposed)
100 points =

+10 point for choosing a reasonable NP-hard problem X to reduce from.

- The Cook-Levin theorem implies that *in principle* one can prove NP-hardness by reduction from *any* NP-complete problem. What we're looking for here is a problem where a simple and direct NP-hardness proof seems likely.
- You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).

+20 points for a *structurally sound* polynomial-time reduction. Specifically, the reduction must:

- take an *arbitrary* instance of the declared problem X ***and nothing else*** as input,
- transform that input into a corresponding instance of Y (the problem we're trying to prove NP-hard),
- transform the output of the oracle for Y into a reasonable output for X, and
- run in polynomial time.

(The output transformation is usually trivial.) This is strictly about the structure of the reduction algorithm, not about its correctness. No credit for the rest of the problem if this is wrong.

+20 points for a *correct* polynomial-time reduction. That is, assuming a black-box algorithm that solves Y in polynomial time, the proposed reduction actually solves problem X in polynomial time.

+20 points for the "if" proof of correctness. (Every good instance of X is transformed into a good instance of Y.)

+20 points for the "only if" proof of correctness. (Every bad instance of X is transformed into a bad instance of Y.)

+10 point for writing "polynomial time"

- An incorrect but structurally sound polynomial-time reduction that still satisfies half of the correctness proof is worth at most 60/100.
- A reduction in the wrong direction is worth 0/100.

Rubric: Standard rubrics for undecidability proofs

- **Diagonalization:**
  1. +40 for correct wrapper Turing machine
  2. +60 for self-contradiction proof (= 30 for $\Leftarrow$ + 30 for $\Rightarrow$)
- **Reduction:**
  1. +40 for correct reduction
  2. +30 for "if" proof
  3. +30 for "only if" proof
- **Rice's Theorem:**
  1. +40 for positive Turing machine
  2. +40 for negative Turing machine
  3. +20 for other details (including using the correct variant of Rice's Theorem)