Rules. Without them we live with the animals.

— Winston [Ian McShane], *John Wick: Chapter 2* (2017)

Good men don't need rules.
Today is not the day to find out why I have so many.

— The Doctor [Matt Smith], "A Good Man Goes To War", *Doctor Who*, season 6, episode 7 (June 4, 2011)

Meta-rubric.

- These rubrics are intended both as a description for students of what we expect in their solutions, and instructions to graders for awarding partial credit. Individual problems sometimes vary from these standard rubrics. We reserve the right to modify these rubrics during the semester, but changes will never be applied retroactively.
- 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).
- Each standard rubric describes partial credit for a problem worth 10 points. Partial credit for subproblems worth less than 10 points should be scaled appropriately. I recommend rounding scaled partial credit to the nearest half-integer.
- Most rubrics can be summarized roughly as follows:
  - 2 points for an answer with the correct *syntax*.
  - 4 points for an answer with the correct *meaning*.
  - 4 points for an English explanation or proof.
- ***The graders cannot read your mind, and they have been instructed not to try.*** Submissions that are unclear for any reason — for example, sloppy handwriting, out-of-focus scanning, poor spelling or grammar, ambiguous variable names, not enough detail, too much detail — will be penalized. The graders have *complete* discretion here; if the grader thinks something is unclear, then it is unclear. Similarly, if something in your solution is ambiguous, the graders have been explicitly instructed to interpret it the wrong way.
- Long and painful experience suggests that the words "clearly", "obviously", "simply", and "just" indicate places where the author trusted their intuition instead of actually working through the details, and thus is very likely **wrong**. Good intuition is the result of hard work, not a replacement for it. The graders will treat any submissions that use these words with *very* close scrutiny.

Standard induction rubric. 10 points =

+ 1 for explicitly considering an *arbitrary* object.
+ 2 for an explicit valid ***strong*** induction hypothesis
  - **Deadly Sin!** No credit here for stating a weak induction hypothesis, unless the rest of the proof is *absolutely perfect*. Weak induction should die in a dumpster fire.
  - Yes, we want you to write it down. Yes, even if it's "obvious". Remember that the goal of the homework is to communicate with people who aren't as clever as you.
+ 2 for explicit exhaustive case analysis
  - No credit here if the case analysis omits an infinite number of objects. (For example: all odd-length palindromes.)
  - −1 if the case analysis omits a finite number of objects. (For example: the empty string.)
  - −1 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.)
+ 1 for cases that do not invoke the inductive hypothesis ("base cases")
  - No credit here if one or more "base cases" are missing.
+ 2 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.
+ 2 for other details in cases that invoke the inductive hypothesis ("inductive cases")
  - No credit here if one or more "inductive cases" are missing.
− 1 for arguing $\forall k \leq n : P(k) \Rightarrow P(n+1)$ (or worse, $P(k) \Rightarrow P(k+1)$) instead of arguing $\forall k < n : P(k) \Rightarrow P(n)$. Yes, we know it's mathematically correct, but it's bad style. Yes, we know that's how your discrete math book does it, but it's bad style. No, I don't care who your discrete math instructor was; it's still bad style. Write your inductive proofs *exactly* the way you write your recursive algorithms. Yes, I *will* die on this hill.

Standard regular expression rubric. 10 points =

- 2 points for a syntactically correct regular expression.
- **Homework only:** 4 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.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - −4 for incorrectly answering $\emptyset$ or Σ^*.
  - −1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language. (The incorrectly handled string is almost always the empty string ε.)
  - −2 for incorrectly including/excluding more than one but a finite number of strings. (Does this rubric item ever actually apply?)
  - −4 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.

Standard DFA/NFA design rubric. 10 points =

- 2 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 $A = \emptyset$, say so explicitly.
    - If your drawing omits a junk/trash/reject/hell 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 $\delta(q, \varepsilon)$ 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 $\delta(q, \varepsilon)$ 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.
- **Homework only:** 4 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.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - -1 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. (The incorrectly accepted/rejected string is almost always the empty string ε.)
  - -4 for incorrectly accepting every string, or incorrectly rejecting every string.
  - -2 for incorrectly accepting/rejecting more than one but a finite number of strings. (Does this rubric item ever actually apply?)
  - -4 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.

Standard fooling set rubric. 10 points =

- 4 points for the fooling set:
  - + 2 for explicitly describing the proposed fooling set F.
  - + 2 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.
- 6 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.
  - + 2 for explicitly describing a suffix z that distinguishes x and y.
  - + 2 for proving either $xz \in L$ or $yz \in L$.
  - + 2 for proving either $yz \notin L$ or $xz \notin L$, respectively.

Alternate fooling set rubric. 10 points =

- 4 points for the fooling set:
  - + 2 for proposing an infinite fooling set $X = \{x_1, x_2, x_3, \ldots\}$, by explicitly defining a string x_i for each positive integer i.
  - + 2 if the proposed set X is actually a fooling set for the target language.
  - — No credit for the proof if X is not a fooling set.
  - — No credit for the problem if the strings in X depend on more than parameter (for example: 0^*1^*).
- 6 points for the proof:
  - The proof must correctly consider *arbitrary* indices $i < j$.
    - — No credit for the proof unless i and j can be *any* pair of distinct positive integers.
  - + 2 for explicitly describing a suffix z_{ij} that distinguishes x_i and x_j.
  - + 2 for proving either $x_i z_{ij} \in L$ or $x_j z_{ij} \in L$.
  - + 2 for proving either $x_j z_{ij} \notin L$ or $x_i z_{ij} \notin L$, respectively.

Standard language transformation rubric. For problems worth 10 points:

+ 2 for a formal, complete, and unambiguous description of the output automaton M', including the states, the start state(s), the accepting states, and the transition function, as functions of an *arbitrary* given DFA M. The description must state whether the output automaton is a DFA or an NFA, and if it is an NFA, whether it uses multiple start states and/or ε-transitions.
  - No points for the rest of the problem if this is missing.
+ 2 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.
+ 6 for correctness
  + 3 for accepting *all* strings in the target language
    * But no credit for incorrectly accepting every string in Σ^*
  + 3 for accepting *only* strings in the target language
    * But no credit for incorrectly rejecting every string in Σ^*
  – 1 for a single mistake in the formal description (for example a typo)
  - Double-check correctness when the input language is $\emptyset$, or $\{\varepsilon\}$, or 1^*, or Σ^*.

Proposed language transformation rubric. For problems worth 10 points:

+ 2 for a formal description, as above
+ 2 for a *brief* English explanation, as above
+ 6 for correctness — The existing rubric is extremely harsh toward submissions that make only minor mistakes. This revision tries to ameliorate that.
  + 1 for correct states — Almost always a product of the states Q of the given DFA with other side information; does the side information make sense? Could you build a transformation using *only* this side information?
  + 1 for correct start state(s)
  + 1 for correct accepting states
  + 3 for correct transition function
    — 1 for a single minor mistake
  - Double-check correctness when the input language is $\emptyset$, or $\{\varepsilon\}$, or 1^*, or Σ^*.
  - Partial credit should be awarded relative to the *most similar correct solution*. For example, if a given incorrect solution can be fixed either by changing the accepting states or by changing the transition function, it should get partial credit for a good transition function.

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. We assume that you've also read the book, and that you also know we've read the book. Moreover, we assume that you know that we know that you've read the book.
- Every algorithm must be clearly ***specified*** in English, unless the specification is already given *precisely* in the problem statement. A description of the algorithm is not enough; we want a description of *what* your algorithm computes, not just an explanation of *how* your algorithm works.

  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.
- **Target time bounds.** Every algorithm design question has a target time bound. Faster algorithms are worth more points, and slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n in either direction. Partial credit is ***SCALED*** up or down to the new maximum score.

  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 0/10) instead of correct algorithms that are slower than the target (earning 7/10).

  Yes, it is possible to score more than full credit. A few past students have earned homework *averages* and even *exam* scores over 100% by submitting faster-than-expected algorithms. (As far as I know, nobody has ever earned a *course* average over 100%, but nothing prevents that. In principle, you can earn a *ridiculously* high score by submitting a completely broken algorithm that runs in $O(n)$ time when the target time is $O(n^3)$, receiving full+extra credit from the grader, and then submitting a regrade request to correct overly lenient grading.)

  On the other hand, every completely specified and correct algorithm is worth at least 3/10, regardless of its running time. Yes, even if the running time is triply exponential. But watch for hand-waving like "for every subset" or "try every path"; if you want to exhaustively enumerate subsets or paths or some other structure, you have to tell us how. (In particular, depth-first search does *not* explore every path in the input graph.)

Standard dynamic programming rubric. 10 points =

- 3 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.)
  - — 1 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.
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 for base case(s). —½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 for recursive case(s). —1 for each *minor* bug, like a typo or an off-by-one error.
  - — 2 for greedy optimizations without proof, even if they are correct.
  - – **No credit for iterative details if the recursive case(s) are incorrect.**
- 3 points for iterative details
  - + 1 for describing an appropriate memoization data structure. **Hash tables are NOT an appropriate memoization data structure!**
  - + 1 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. (In particular, if you draw a rectangle for a 2d array, be sure to label and direct the row and column indices.)
  - + 1 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.
- Iterative pseudocode is ***not*** required for full credit, provided the other details of your solution are clear and correct. We usually give two official solutions, one with pseudocode and one without.

  If your solution does 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***, or at most 5 points out of 10 if you name the memoization array "DP".
- 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).

Standard graph-reduction rubric. 10 points =

- + 3 for constructing the correct graph.
  - + 1 for correct vertices
  - + 1 for correct edges
  - — ½ for forgetting "directed" if the graph is directed
  - + 1 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 5 points using the appropriate standard rubric, and all other points are cut in half.
- + 3 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 relate to the original problem?
  - – No points for only naming the algorithm, not the problem. "Breadth-first search" is not a problem!
- + 2 for correctly applying the correct **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.")
  - — 1 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.
  - — 1 for explaining an algorithm from lecture or the textbook instead of just invoking it as a black box.
- + 2 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. Really, just do it the other way.

Standard NP-hardness rubric. 10 points =

+ 1 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).
+ 2 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.
+ 2 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.
+ 2 points for the "if" proof of correctness. (Every good instance of X is transformed into a good instance of Y.)
+ 2 points for the "only if" proof of correctness. (Every bad instance of X is transformed into a bad instance of Y.)
+ 1 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 5/10 (= 1 for reasonable reduction source + 2 for structural soundness + 2 for half of the proof)
- A reduction in the wrong direction is worth at most 1/10 (for choosing a reasonable problem)

Standard rubrics for undecidability proofs.

- **Diagonalization:**
  - \+ 4 for correct wrapper Turing machine
  - \+ 6 for self-contradiction proof (= 3 for $\Leftarrow$ + 3 for $\Rightarrow$)
- **Reduction:**
  - \+ 4 for correct reduction
  - \+ 3 for "if" proof
  - \+ 3 for "only if" proof
- **Rice's Theorem:**
  - \+ 4 for positive Turing machine
  - \+ 4 for negative Turing machine
  - \+ 2 for other details (including using the correct variant of Rice's Theorem)