Computability and Computational Complexity, A.Y. 2025–2026

Guide to the answers (revised version based on student answers to the test)

Tuesday, July 23, 2026, revised on July 24, 2026

Exercise 1

For each of the following properties of Turing machines, show whether they are recursive or not. Invoke Rice's Theorem when possible; if it doesn't apply, explain why.

$$\begin{aligned}
\mathcal{P}_1 &= \{\mathcal{M} | \mathcal{M} \text{ halts in at most 100 steps on the empty tape}\};\\
\mathcal{P}_2 &= \{\mathcal{M} | \mathcal{M} \text{ halts in at most 100 steps on every input}\};\\
\mathcal{P}_3 &= \{\mathcal{M} | \mathcal{M} \text{ halts on every input}\};\\
\mathcal{P}_4 &= \{\mathcal{M} | L(\mathcal{M}) \text{ only contains strings of even length}\}.
\end{aligned}$$

Solution 1

- $\mathcal{P}_1$ isn't semantic: two TMs may recognize the same language, yet have totally different runtimes on the empty string, just by adding dummy states at the beginning or at the end of the computation. Therefore Rice's Theorem doesn't apply.
  The property is computable: to test $\mathcal{P}_1(\mathcal{M})$ we just need to simulate the first 100 steps of $\mathcal{M}(\varepsilon)$. If it halts within the simulation then the property is true, else it isn't.

- $\mathcal{P}_2$ isn't semantic for the same reason as $\mathcal{P}_1$.
  Again, the property is computable. Although the property requires testing "every input" of the machine, since the simulation only looks for 100 steps we only need to test $\mathcal{M}(s)$ it for all possible combinations s of symbols in the 201 tape cells that are reachable within the time limit (100 to the right and 100 to the left): the remaining cells, albeit infinite, have no influence on the simulation.

- $\mathcal{P}_3$ is semantic: as soon as we define "acceptance" as halting (i.e., if $\mathcal{M}(s)$ halts then $\mathcal{M}$ accepts s), then the property means that $\mathcal{M}$ accepts every input, i.e., $L(\mathcal{M}) = \Sigma^*$ (where Σ is $\mathcal{M}$'s alphabet). The property is not trivial (there are plenty of machines that accept all strings, and plenty that don't). Hence, Rice's Theorem is applicable and $\mathcal{P}_3$ is not computable.
  As analternative, if we don't want to invoke the "semanticity" of $\mathcal{M}$, we can reduce $\text{HALT}_\varepsilon(\mathcal{M})$ to $\mathcal{P}_3$: given $\mathcal{M}$, transform it into $\mathcal{M}'$ that erases it input and then executes $\mathcal{M}$. Clearly, $\mathcal{M}'$ halts on every string if and only if $\mathcal{M}$ halts on the empty string:

  $$\text{HALT}_\varepsilon(\mathcal{M}) \Leftrightarrow \mathcal{P}_3(\mathcal{M}').$$

  We have therefore shown that $\text{HALT}_\varepsilon \leq_\text{T} \mathcal{P}_3$, meaning that $\mathcal{P}_3$ is "less" computable that the halting problem.

- $\mathcal{P}_4$ is clearly semantic, because it refers to the language accepted by $\mathcal{M}$. More specifically, if $L(\mathcal{M}_1) = L(\mathcal{M}_2) = L$ then either L only has even-sized strings (thus $\mathcal{P}_4(\mathcal{M}_1) = \mathcal{P}_4(\mathcal{M}_2) =$ true) or it doesn't (thus $\mathcal{P}_4(\mathcal{M}_1) = \mathcal{P}_4(\mathcal{M}_2) =$ false). In both cases, $\mathcal{P}_4(\mathcal{M}_1) = \mathcal{P}_4(\mathcal{M}_2)$. $\mathcal{P}_4$ is clearly not trivial, hence Rice's Theorem applies and $\mathcal{P}_4$ is not computable.

Observations

For $\mathcal{P}_3$ it is not enough to say "to see if $\mathcal{M}$ halts on every string we should test the halting problem on infinite strings, which is not feasible". This reasoning only considers *one* possible way of solving the problem (namely, using HALT$(\mathcal{M}, s)$), but others could exist. In other words, the reduction is performed in the wrong way, $\mathcal{P}_3 \leq_{\mathrm{T}}$ HALT, while the correct sense to show $\mathcal{P}_3$'s uncomputability would be HALT $\leq_{\mathrm{T}} \mathcal{P}_3$, as in the proposed solution.

Exercise 2

An astronomer has a list of 50 stars that must be photographed by an automated telescope during one night. The telescope starts from its initial resting position and must subsequently point to each of the 50 stars. The time for moving the telescope from one star to the next is proportional to their angular distance. After pointing (and shooting) to all the stars in the list, the telescope must move back to its resting position.
In order to minimize the expensive telescope time, the astronomer writes a program that, given the list of star coordinates, computes the optimal photographing sequence, i.e., the sequence of telescope movements starting and ending at the resting position, covering all 50 stars, and taking the shortest possible time. The program does not directly drive the telescope, and can be run at any moment during the day: on the 50-star list it takes 10 minutes and outputs the optimal sequence. In the night, the sequence is fed to the actual telescope driver and is completed in 2 hours of telescope time.
A few days later, the astronomer has a new list of twice as many (100) stars to be photographed on the next night. In the morning they fire up their program, but it keeps crunching numbers for the whole day until the astronomer, out of patience, kills it.

2.1) Why did this happen? Identify a known hard problem and show by reduction that this behavior is, indeed, expected.

2.2) Using the existing program with a reasonably short runtime, is it still possible for the astronomer to plan a (possibly suboptimal) sequence that still takes all 100 required photographs in one night?

Hint — *For 2.2, you might want to consider a* divide et impera *strategy.*
For simplicity, assume that all required stars are visible throughout the night.

Solution 2

2.1) First of all, observe that the Astronomer's Problem (AP from now on) is just a reformulation of TSP (the Traveling Salesperson Problem) where the "cities" are the star positions and the resting position, and the distance between all pairs of "cities" is known (the angle that the telescope must "travel" to move from one to another).
More formally, we can prove that the astronomer's problem is NP-hard.
First of all, we need to define a decision version of AP, let's call it DAP, by adding an angular distance budget and asking whether there is a solution whose total angular distance is below the budget.
We can show that DAP (and hence AP) is **NP**-hard by defining a reduction from TSP to DAP. Suppose that we have a set of cities with their coordinates; we can transform it into a set of stars to be visited by mapping every city to a position in the night sky so that distances are approximately preserved. One city will be mapped to the telescope's resting position.
At this point, it is clear that if we could solve AP in polynomial time then TSP $\in$ **P** too, and this is considered extremely unlikely.
Therefore, the Astronomer's algorithm probably has exponential time complexity with respect to the input size: every time we add a new star to the list, we can expect the runtime to go up by a multiplicative factor, say $f > 1$. Hence, by doubling the number of stars (i.e., by adding 50 more stars to the list) the runtime goes up by a factor of f^{50}, which is likely an extremely high multiplier to the initial 10-minute runtime. We cannot expect the program to halt any time soon.

2.2) A possible solution for the astronomer would be to split the list of 100 stars into two 50-star lists, planning the two optimizations separately and running one plan after the other. The two separate algorithm runs would likely require $10 + 10 = 20$ minutes, and the actual telescope

sessions would take $2 + 2 = 4$ telescope hours.
The astronomer could probabily optimize the task a little more by dividing the list by proximity (e.g., all stars left of some median line go to the first list, all others go to the second), so that the telescope is likely to move through smaller distances — however, this is outside the scope of this exercise.

Observations

- This was an open-ended exercise where a plurality of answers is possible, the above is just an example.
- Indeed, the proposed answer isn't completely satisfactory: we aren't reducing a "general" TSP instance to the Astronomer's Problem, rather one in which distances are Euclidean. This would require a little more clarity on whether the "2D Euclidean" TSP is **NP**-complete (it is); however, for the scope of our course it is OK to skip that passage.
- Moreover, "not polynomial" doesn't mean "at least exponential": there are other complexities in the middle, which we are ignoring for the sake of simplicity.
- As always, the direction of the reduction is important! Showing that AP $\leq_{\mathbf{P}}$ TSP is useless, since every problem in **NP** (including polynomial ones) can be reduced to an **NP**-complete one by definition.

Exercise 3

3.1) Define the time classes **P**, **NP**, and **EXP**.

3.2) Prove that **P** $\subseteq$ **NP** $\subseteq$ **EXP**.

Solution 3

See the notes.

Observations

- Speaking of polynomial/exponential time is fine, but with respect to *what*? In other words, if you write $O(n^k)$ you should also say what is n.