ECE 374 B: Algorithms and Models of Computation, Fall 2025

Midterm 3 – December 04, 2025

- **You will have 75 minutes (1.25 hours) to solve all the problems. Most have multiple parts.** Don't spend too much time on questions you don't understand and focus on answering as much as you can!
- ***BUDGET YOUR TIME WISELY***. I highly recommend working on the questions you know first and the questions you need to think about second.
- *No* resources are allowed for use during the exam except a multi-page cheatsheet and scratch paper on the back of the exam. ***Do not tear out the cheatsheet or the scratch paper!*** It messes with the auto-scanner.
- You should write your answers *completely* in the space given for the question. We will not grade parts of any answer written outside of the designated space.
- Please *use a dark-colored pen* unless you are *absolutely* sure your pencil writing is forceful enough to be legible when scanned. We reserve the right to take off points if we have difficulty reading the uploaded document.
- Unless otherwise stated, assume $\mathsf{P} \neq \mathsf{NP}$.
- Assume that whenever the word "reduction" is used, we mean a (not necessarily polynomial-time) *mapping/many-one* reduction.
- You can only refer to the cheat sheet content as a black box.
- ***Don't cheat.*** If we catch you, you will get an F in the course.
- ***Good luck!***

Name: _____

NetID: _____

Date: _____

1 Short Answer I (2 questions) - 10 points

For each of the reductions containing a known problem and an unknown problem (X) please circle the complexity classes that X **MAY** belong to. Only circle the classes, no explanation is needed. There will be no partial credit. If no complexity classes should be selected, please circle "none of the above."

(a) **SAT**: Given a conjunctive normal formula, determine if there is a truth assignment that makes the formula evaluate to true.

Reduction: SAT $\leq_P$ X

Choose the classes that X **MAY** belong to:

Solution: Choose the classes that X **MAY** belong to:

P NP NP-complete Decidable Undecidable

None of the above

Since SAT can be reduced to X in polynomial time, X must be atleast as hard as SAT. We know SAT is NP-Complete, hence X must be NP-Hard and may be NP. As P $\neq$ NP and SAT is not P, X can not be P.

Since SAT is decidable, we cannot be rule out Decidable. Hence, without more information, it may be Decidable or Undecidable.

In summary, we can only rule out P. ■

(b) **LIS**: Given a sequence A and an integer k, return TRUE if the longest increasing subsequence is more than k in length. FALSE otherwise.

Reduction: LIS $\leq_P$ X

Choose the classes that X **MAY** belong to:

Solution: Choose the classes that X **MAY** belong to:

P NP NP-complete Decidable Undecidable

None of the above

As LIS can be reduced to X in polynomial time, X must be atleast as hard as LIS. As LIS is P and decidable, all options except *None of the above* are possible. ■

2 Short Answer II (2 questions) - 10 points

For each of the reductions containing a known problem and an unknown problem (X) please circle the complexity classes that X **MUST** belong to. Only circle the classes, no explanation is needed. There will be no partial credit. If no complexity classes should be selected, please circle "none of the above."

(a) **SAT**: Given a conjunctive normal formula, determine if there is a truth assignment that makes the formula evaluate to true.

Reduction: X $\leq_P$ SAT

Choose the classes that X **MUST** belong to:

P NP NP-complete Decidable Undecidable

None of the above

Solution: Choose the classes that X **MUST** belong to:

P (NP) NP-complete (Decidable) Undecidable

None of the above

- X **is NP:** Since there is a polynomial time reduction from X to SAT, X is at most as complex as SAT. SAT is an NP-complete problem, and therefore X must also be NP.
- X **is Decidable:** Any NP problems are decidable, so X must be decidable.

Assuming that P $\neq$ NP, X may be a problem that is NP but not P or NP-complete, therefore X is not necessarily P or NP-complete. Since X must be Decidable, it can't be Undecidable. ■

(b) **HALT**: Given a TM M and input string w, if $M(w)$ halts, return TRUE, otherwise return false.

Reduction: X $\leq_P$ HALT

Solution: Choose the classes that X **MUST** belong to:

P NP NP-complete Decidable Undecidable

(None of the above)

- X **may be decidable:** A decidable problem SAT can be reduced to HALT, by constructing a TM M' that tests every assignments and halts if there exists a satisfying assignment.
- X **may be undecidable:** An undecidable problem HALTONSELF may be reduced to HALT, by passing $(M, \langle M \rangle)$ as the input.

Since X can either be decidable or undecidable, there is no class that X **must** belong to. ■

3 Short Answer III (4 questions) - 20 points

For each of the problems provide a brief and concise solution. These are short answer questions and partial credit will be limited.

(a) Write an example of a unsatisfiable 3SAT formula where the literals within a clause are distinct.

Solution: In 3SAT, each clause is in the form of **OR** of 3 literals. Assume that we have 3 variables x, y, z. Furthermore, there can be at most $2^3 = 8$ total assignments to (x, y, z). Notice that a 3SAT formula is **unsatisfiable** if no assignment for x, y, z exists that makes all the clauses true at the same time. Consequently, we can write a 3CNF formula as follows:

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z)$$
$$\wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

As you can see, in each clause, the 3 literals are distinct and because every assignment to (x, y, z) makes atleast one clause false, the conjunction of all 8 clauses is unsatisfiable. ■

(b) For the SAT formula snippet below, transform it to a **3**SAT formula snippet. You may not make any assumptions for how the literals are used in the other clauses. If you add any extra variables, clearly state/mark that you did so.

$$\ldots \wedge (w \vee x \vee y \vee z) \wedge (u \vee v) \wedge \ldots$$

Solution: Introduce new variables a and b. By introducing these 2 variables, we can re-write the 4-literal clause and the 2-literal clause as follows:

$$(w \vee x \vee y \vee z) = (w \vee x \vee a) \wedge (\neg a \vee y \vee z)$$
$$(u \vee v) = (u \vee v \vee b) \wedge (u \vee v \vee \neg b)$$

Hence, the new formula becomes:

$$\ldots \wedge (w \vee x \vee a) \wedge (\neg a \vee y \vee z) \wedge (u \vee v \vee b) \wedge (u \vee v \vee \neg b) \wedge \ldots$$

■

(c) Given a directed, **fully-connected** graph ($G = (V, E)$), what is the minimum value of k (minimum number of colors), needed to color this graph.

Solution: In a directed **fully-connected** graph every pair of distinct vertices is adjacent, so no two vertices can share a color. Hence, the minimum value is $k = |V|$ ■

(d) You know that a problem (X) is context-free. Is it decidable or undecidable?

Solution: All context-free languages are decidable, as provided on the cheat sheet. A language L is decidable if there exists an algorithm which takes any arbitrary string w and determines if w is in L. There is a standard algorithm(CYK) that achieves this for context-free languages, although the algorithm itself is out of scope. ■

4 Classification I (P/NP) - 12 points

Is the following problem in P, NP, or some combinations of complexity classes? For each of the following problems, circle all the complexity classes that problem belongs to. Whatever class it is in, prove it!

A **TasteTheRainbow(TTR) coloring** asks if there is a coloring for an undirected graph $G = (V, E)$ where each vertex can be colored such that no two adjacent vertices are colored the same, *and* there are an equal number of vertices for each color.

- INPUT: A undirected graph G and integer k.
- OUTPUT: TRUE if there exists a coloring with at most k colors and all colors are used equally. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle ***all*** that apply:

Solution: P (NP) (NP-hard) (NP-complete)

NP Proof

To prove that the problem is NP, we have the certificate that is a coloring, and a verifier that checks the following

- check that k colors are used
- check that for an edge, the two vertices it connects are not the same color
- count how many times each color is used for the verticies and confirm they are equivalent.

The verifier is in polynomial time, so the problem is in NP.

NP Hardness Proof 1

We reduce from NP-hard to 3-coloring. Consider the TTR case for $k = 3$. Build G' as three copies of G and for every $v_i \in G$ add three edges between $v_{i1}, v_{i2}, v_{i3} \in G'$, where 1,2,3 is the version of the copy of G.

$\Rightarrow$: if G is 3-colorable, G' is TTR for $k = 3$
For the coloring of the vertices in G', let there be an cycled ordering of three unique colors. Give v_{i1} the same color as $v_i \in G$, give v_{i2} the subsequent color in the ordering after v_{i1}'s color, and give v_{i3} the subsequent color in the ordering after v_{i2}'s color. Every copy will be a proper 3 coloring because we relabeled the colors. Also, there will be no neighboring colors in each connected v_{i1}, v_{i2}, v_{i3}. With the way we distributed the colors on G', each of the colors are used the same amount of times. Therefore G' is TTR for $k = 3$.

$\Leftarrow$: if G' is TTR with $k = 3$, G is 3-colorable.
If G' has a proper TTR $k = 3$ coloring, take only the first copy of the three copies for G, and we have a proper 3-coloring for G.

NP Hardness Proof 2

There is also another way to reduce the solution from the NP-hard problem 3-coloring. To summarize the reduction we add a three cycle to every vertex in a graph $G = (V, E)$. Start by taking an input graph G, and create $G' = (V', E')$ by adding two vertices to every vertex, and add three edges such that we create a triangle of three connected vertices in G' for every vertex in G.

$\Rightarrow$: if G is 3-colorable, G' is TTR for $k = 3$
For the coloring of the vertices in G', again let there be an cycled ordering of three unique colors. color the two vertices you attached to G to create G' the other two colors. This gives a proper TTR-coloring for $k = 3$, since each color is used $|V|$ times, and no adjacent edges are colored the same color, because the two nodes we added to each vertex are isolated to just the existing vertex in G. Therefore G' is TTR for $k = 3$.

$\Leftarrow$: if G' is TTR with $k = 3$, G is 3-colorable.
Start with G' and analyze the graph without the two extra nodes we added for every vertex. If G' has a proper TTR $k = 3$ coloring, G has a proper 3-coloring by construction.

NP Complete Proof

Since the problem is in both NP and NP-hard it is also NP-complete.

■

5 Classification II (P/NP) - 12 points

Is the following problem in P, NP, or some combinations of complexity classes? For each of the following problems, circle all the complexity classes that problem belongs to. Whatever class it is in, prove it!

A **HamilDAG** problem asks if there is a path that visits every vertex in a directed acyclic graph exactly once.

- INPUT: A directed acyclic graph G
- OUTPUT: TRUE if there exists a path that visits every vertex exactly once. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle ***all*** that apply:

Solution: (P) (NP) NP-hard NP-complete

P proof:

For each vertex in the graph, we count the in-degree and out-degree. We then count the number of vertices with in-degree equals 0 (source), and out-degree equals 0 (sink). If there are exactly one sink and one source, we return true; and false otherwise. Traverse each vertex takes $O(V+E)$ time so the algorithm is clearly in P.
One could also conduct topological sort to check the number of sinks and sources; or check if there is an edge from v_i to v_{i+1} for all $i \in [1, n-1]$. And the algorithm still runs in $O(V+E)$ time, which means the problem is in P.

NP Proof:

Since the problem is in P, it's also in NP.
One could also prove NP using the certificate and certifier. The certificate will be a given Hamiltonian Path H in a DAG G. And the certifier will be iterate through H to check if H is a valid HamilDAG.

common mistake and notes:

1. If X is a known NPC problem, $X \leq_P Y$ only implies $Y \in NPH$, it does not show $Y \in NP$ even if the reduction is completely correct.
2. The problem is essentially checking if the longest path in the graph has length n, which is almost the same thing as Lab 16 question $1b$. But you can't simply using it as a black box, as it's not in the cheat sheet. This applies to some other approach that uses algorithm that is not in the cheat sheet as black box algorithm. Only partial credit will be given.
3. Consider the graph with node $v_1, v_2, ..., .v_k$ where $k \geq 3$, where there is an edge from v_i to v_j is $i < j$. If we do topological sort and start a traverse on v_1, the result might

not be a path. As we might get a tree with 1 root and $n-1$ leaves. This basically shows that if one attempt to use BFS/DFS to find the path, it won't work, as the search result might be a tree even if there exists a path.

4. Consider three node that forms a binary tree: v_1, v_2, v_3, where there are two edges one from v_1 to v_2, the other is from v_1 to v_3. Do DFS/BFS on v_1 can visit all three nodes, but the graph has no HamilDAG.

5. We did not want to put hard questions as exam questions, but that does not imply algorithm could get really vague or skipping a lot of details. Stating something like 'run topological sort and then check if the graph has a hamilDAG' is not acceptable, as this is basically answering the question with the question itself, where the question itself of whether this Hamiltonian Path existed or not remain unsolved.

6. Solution that states check all the path and verify if there is a path with length n also need to go into detail of describing a recursion on finding all the path and explain why it could run polynomial time if the graph is a DAG. As for arbitrary graph, the size of all paths is exponential.

■

6 Classification (Decidability) - 12 points

Same type of problem as before, but now you have to determine if it is decidable or not and provide a concise explanation/proof why.

A **AsManyAsPossibleSAT (AMAPSAT)** problem asks if a formula ϕ has a truth assignment that can satisfy atleast k clauses.

- INPUT: A formula ϕ and an integer k
- OUTPUT: TRUE if there exists a truth assignment that satisfies at least k clauses. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle ***all*** that apply:

Solution: (Decidable) Undecidable

The input is a boolean formula ϕ and an integer k which returns TRUE if there exists a truth assignment that satisfies at least k clauses and FALSE otherwise. Let n be the number of variables to consider in ϕ. We can construct a deterministic algorithm for this problem: enumerate all 2^n possible truth assignments, for each assignment count the number of clauses that are satisfied, and return TRUE iff there is some assignment that satisfies at least k clauses. If no such assignment exists, return FALSE. There are a finite number of assignments to check, and each evaluation takes finite time, so the algorithm always halts, and therefore AMAPSAT is decidable. ■

Solution: (Decidable) Undecidable

Another proof that would work is to prove that AMAPSAT is in NP and since all problems in NP are Decidable, then AMAPSAT is Decidable. Below is the proof for NP. The input is (ϕ, k), and let ϕ have n variables $v_1, \ldots v_n$ and m clauses $C_1, \ldots, C_m$.

Certificate: An assignment ξ for the n variables in ϕ.

Polynomial-time Certifier: Check that the assignment ξ results in at least k clauses of ϕ to be satisfied. (Just loop over all clauses and count number that are satisfied under ξ.) ■

Solution: (Decidable) Undecidable

Alternatively, you can construct the following reduction: AMAPSAT $\leq_p$ SAT (reduction from the AMAPSAT problem). Our input would be (ϕ, k), and we would want to produce a formula ψ such that ψ is satisfiable iff there exists a truth assignment for ϕ that satisfies at least k clauses.

Note: I don't know why one would want to do it this way unless you are a masochist, or just got confused, both cases being okay. But now I have to sit here and figure it out, but that's what I signed up for as a TA, so that I could help y'all.

Our construction would be as follows. Let ϕ have n variables $v_1, \ldots v_n$ and m clauses $C_1, \ldots, C_m$. I am going to ignore the edge cases of $k < 0$ or $k > m$ since they are trivial. For each clause C_i, we introduce a new boolean variable u_i, such that $u_i = 1$ iff C_i is satisfied for an assignment of v. For each clause and its corresponding literals $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ldots \ell_{i,p})$, construct a new clause $H_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ldots \ell_{i,p} \vee \neg u_i)$ to add to ψ, for $1 \leq i \leq m$. For each

literal $\ell_{i,j}$ in a clause C_i, construct a new clause $G_{i,j} = (\neg\ell_{i,j} \vee u_i)$ to add to ψ, for $1 \le i \le m$, $1 \le j \le |C_i|$. Construction of the clauses H_i and $G_{i,j}$ guarantees us that $u_i \iff C_i$. Now we need to encode the requirement of at least k clauses being satisfied, which we can do by encoding the following summation: $\sum_{i=1}^{m} u_i \ge k$ through a polynomial time sequential-counter encoding as follows. Introduce auxiliary variables $s_{i,j}$ for $1 \le i \le m$, $1 \le j \le k$, where $s_{i,j}$ means that among the first i variables $u_1, \ldots, u_i$, at least j are true. To enforce $\sum_{i=1}^{m} u_i \ge k$, we just need to enforce $s_{m,k}$ to be TRUE by adding $(s_{m,k})$ to ψ. We first add the clauses $(\neg s_{1,j})$ to ψ since for 1 variable, we can't have ≥ 2 TRUEs so they must be FALSE. Additionally, we add the clauses $(\neg u_1 \vee s_{1,1})$ and $(\neg s_{1,1} \vee u_1)$ to ψ to enforce $u_1 \iff s_{1,1}$. We then add transition constraints: for $j = 1$ and for $i = 2, \ldots, m$ add the clauses $(\neg s_{i-1,1} \vee s_{i,1})$, $(\neg u_i \vee s_{i,1})$, and $(\neg s_{i,1} \vee s_{i-1,1} \vee u_i)$ to ψ to enforce $s_{i,1} \iff (s_{i-1,1} \vee u_i)$. Then the transition constraints for the general case ($2 \le j \le k$) to force the correct propagation of counts (to enforce $s_{i,j} \iff (s_{i-1,j} \vee (u_i \wedge s_{i-1,j-1}))$), we add the following clauses to ψ: $(\neg s_{i-1,j} \vee s_{i,j})$ (to enforce $s_{i-1,j} \Rightarrow s_{i,j}$), $(\neg u_i \vee \neg s_{i-1,j-1} \vee s_{i,j})$ (to enforce $u_i \wedge s_{i-1,j-1} \Rightarrow s_{i,j}$), and then both $(\neg s_{i,j} \vee s_{i-1,j} \vee u_i)$ and $(\neg s_{i,j} \vee s_{i-1,j} \vee s_{i-1,j-1})$ (to enforce $s_{i,j} \Rightarrow (s_{i-1,j} \vee (u_i \wedge s_{i-1,j-1}))$). This concludes the construction of the formula ψ, which in the worst case, (assuming m clauses, n variables, and q total literals in ϕ) has $O(m + mq + 1 + k + 2 + 2m + 4mk) = O(mq + mk)$ clauses, making the construction polynomial in time.

($\Rightarrow$: If ψ is satisfiable then ϕ has an assignment that satisfies at least k clauses.) Let α be any satisfying assignment for ψ. Take the corresponding assignment ν of the original variables under α (since $\nu_1, \ldots, \nu_n$ are in ψ by construction, they already have assignments, so take those corresponding assignments). We know that by construction of H_i and $G_{i,j}$, that $u_i \iff C_i$. If $u_i = 1$ under α, that means that there must be a literal in H_i that is not u_i that is assigned to 1, and since all other literals, other than u_i, in H_i are in C_i, then C_i must also have at least one literal assigned to 1 and therefore C_i is satisfied. If some literal in C_i is true under an assignment, then the corresponding $G_{i,j}$ forces $u_i = 1$ under α, meaning that $u_i = 1$ when C_i is TRUE. Additionally, the sequential-counter encoding along with the clause $(s_{m,k})$ means that $s_{m,k} = 1$ under α. By construction of the encoder, $s_{m,k} = 1$ iff at least k of u_i are 1. Therefore, if ψ is satisfiable, then ϕ has an assignment that satisfies at least k clauses.

($\Leftarrow$: If there exists an assignment to ν that satisfies at least k clauses of ϕ, then ψ is satisfiable.) Let β be an assignment for the ν variables in ϕ that makes at least k clauses TRUE. We want to construct a corresponding assignment ζ for ψ. Construct the assignment in the following manner: $\zeta(\nu_i) = \beta(\nu_i)$ (for all original variables ν_i, $1 \le i \le n$), $\zeta(u_i) = 1$ if C_i is TRUE under β, else $\zeta(u_i) = 0$ (for all original clauses C_i, $1 \le i \le m$). By this construction, then each H_i and $G_{i,j}$ is satisfiable in ψ. If $u_i = 1$, then by construction C_i is TRUE under β, so H_i is TRUE. If some literal in C_i is 1, then set $u_i = 1$ so all $G_{i,j}$ are satisfied. For the sequential-counting encoding, set $\zeta(s_{i,j}) = 1$ iff there are at least j ones in $u_1, \ldots, u_i$. For $i = 1$ and $j \ge 2$ $\zeta(s_{1,j}) = 0$, and $\zeta(s_{1,1}) = \zeta(u_1)$, satisfying the base case and initialization clauses. By the definition of $\zeta(s_{i,j})$ and the construction of the general case counting clauses, the equivalence of $s_{i,j} \iff (s_{i-1,j} \vee (u_i \wedge s_{i-1,j-1}))$ holds for each i, j. Finally, because β satisfies at least k clauses, then at least k us are one, and by construction $\zeta(s_{m,k}) = 1$, satisfying the final $(s_{m,k})$ clause. Therefore, given an assignment β to ν that satisfies at least k clauses of ϕ, there is an assignment ζ that makes ψ satisfiable.

Therefore, we have a polynomial time AMAPSAT $\le_p$ SAT reduction, and since SAT is NP-complete and Decidable, then AMAPSAT must be Decidable and in NP, as seen in Problem 2(a). ■

7 Classification (Mixed) I - 12 points

Same type of problem as before, but now you have to determine decidability in addition to algorithmic complexity.

Given the language:

$$\text{PayAttention}_{TM} = \{\langle M \rangle \mid M \text{ accepts on strings "ECE374" and "KANI" }\}$$

Which of the following classes does this language belong to? Whatever you choose, *succinctly* prove it!

P NP NP-hard NP-complete decidable undecidable

Solution: P NP (NP-hard) NP-complete Decidable (Undecidable)

NP-Hardness Proof

- Assuming we have the black box $\text{ALG}_{\text{PayAttention}}$, we reduce (polynomial time) 3SAT to PayAttention as below:

```
3SATSolver(φ):
  Encode the following Turing machine M:
      M(x):
        For all 2^n assignments of 0/1 to variables in φ:
            If φ evaluates to true under the current assignment:
                If x == "KANI", or x == "ECE374", :
                    return True.
        return False
  if ALG_PayAttention
        return True
  else
        return False
```

- If ϕ is satisfiable:
  - M encodes a TM that only accepts only "KANI" or "ECE374"
  - Therefore, $\text{ALG}_{\text{PayAttention}}$ returns True.
- If ϕ is not satisfiable:
  - M encodes a TM that always returns False.
  - M is not a TM that only accepts "KANI" and "ECE374"
  - Therefore, $\text{ALG}_{\text{PayAttention}}$ returns False.

As we have showed that 3SAT is solvable by PayAttention, and thus PayAttention is proven NP-hard.

Decidability Proof

- Assuming we have the decider PayAttention, we reduce (polynomial time) HaltOnInput to PayAttention as below:

```
HaltOnInput(⟨M′, w⟩):
  Encode the following Turing machine M′:
      M′(x):
        Run w on M
        return x = "Kani" or x = "ECE374"
  if PayAttention(⟨M′⟩)
      return True
  else
      return False
```

- If M halts on w:
  - M' is a TM that only accepts "KANI" or "ECE374"
  - PayAttention($\langle M' \rangle$) returns True.
- If M hangs on w:
  - M' is not a TM that accepts "KANI" or "ECE374"
  - PayAttention($\langle M' \rangle$) returns False.

So we have showed that Halt is decidable if we have the decider for PayAttention, and as halt is known to be undecidable, PayAttention is proven undecidable.

■

8 Classification (Mixed) II - 12 points

Same type of problem as before, but now you have to determine decidability in addition to algorithmic complexity.

Given the language:

$$\textsc{Seriously}_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA that accepts atleast one string } w \text{ where } |w| \leq 374\}$$

Which of the following classes does this language belong to? Whatever you choose, *succinctly* prove it!

P NP NP-hard NP-complete decidable undecidable

Solution: (P) (NP) NP-hard NP-complete (decidable)

8.1 P Proof

If a DFA accepts a string with less than 374 characters, it means that there must be a path from the start node to a accept state that is no more than 372 edges long. So use BFS from the start vertex and look at the distance for every accept state. If any of them have a distance <374, then the DFA is part of the language.

8.2 NP Proof

If a problem is in P, it is in NP.

8.3 Decidability Proof

if a problem is in P, it is decidable.

■

EXTRA CREDIT (1 pt)

Give an example of a problem that is not in NP, but is in EXPTIME time.

Solution: Lots of correct answers. Some I liked:

- Tautology - it's in co-NP-complete and as far as we know, there are no problems that are in both NP and co-NP-hard.
- Optimization version of longest path or traveling salesman.
- List powerset output.

■

EXTRA CREDIT (? pt)

What will your pre-curve grade, not including extra credit, be on this exam? If you guess your integer score, I'll add 20 points to your exam grade. You must guess a integer between [20-100] (yes you must score atleast 20 points to be eligible for this EC). We'll round down your actual score to the nearest integer.

Solution: 10 (out of 253) people got it! ■

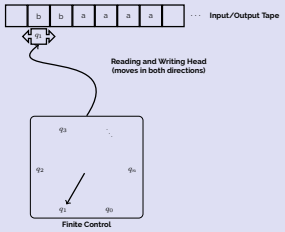
This page is for additional scratch work!

ECE 374 B Reductions, P/NP, and Decidability: Cheatsheet

Turing Machines

Turing machine is the simplest model of computation.

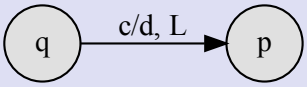
- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Ever step: Read character under head, write character out, move the head right or left (or stay).
- Every TM **M** can be encoded as a string $\langle M \rangle$



Transition Function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow, \square\}$

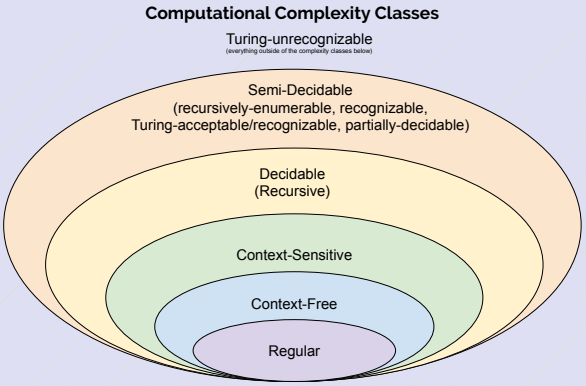
$\delta(q, c) = (p, d, \leftarrow)$

- q: current state.
- c: character under tape head.
- p: new state.
- d: character to write under tape head
- $\leftarrow$: Move tape head left.

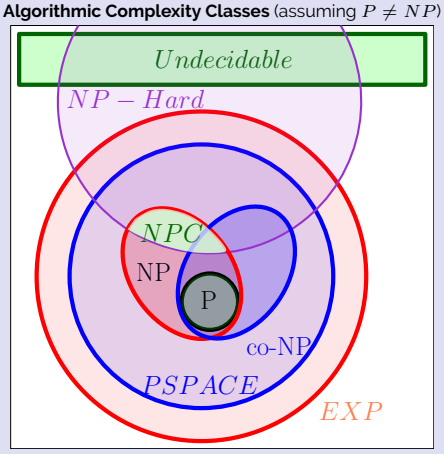


Complexity Classes

Computational Complexity Classes



Algorithmic Complexity Classes (assuming $P \neq NP$)



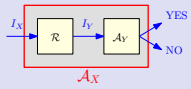
Reductions

A general methodology to prove impossibility results.

- Start with some *known* hard problem X
- *Reduce* X to your favorite problem Y

If Y can be solved then so can $X \implies Y$. But we know X is hard so Y has to be hard too. On the other hand if we know Y is easy, then X has to be easy too.

The Karp reduction, $X \leq_P Y$ suggests that there is a polynomial time reduction from X to Y.



Assuming

- $R(n)$: running time of $\mathcal{R}$
- $Q(n)$: running time of $\mathcal{A}_Y$

Running time of $\mathcal{A}_X$ is $O(Q(R(n)))$

Sample NP-complete problems

CircuitSat:	Given a boolean circuit, are there any input values that make the circuit output True?
3Sat:	Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?
IndependentSet:	Given an undirected graph G and integer k, what is there a subset of vertices $\geq k$ in G that have no edges among them?
Clique:	Given an undirected graph G and integer k, is there a complete complete subgraph of G with more than k vertices?
kPartition:	Given a set X of kn positive integers and an integer k, can X be partitioned into n, k-element subsets, all with the same sum?
3Color:	Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?
HamiltonianPath:	Given graph G (either directed or undirected), is there a path in G that visits every vertex exactly once?
HamiltonianCycle:	Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?
LongestPath:	Given a graph G (either directed or undirected, possibly with weighted edges) and an integer k, does G have a path $\geq k$ length?

- Remember a **path** is a sequence of distinct vertices $[v_1, v_2, \ldots v_k]$ such that an edge exists between any two vertices in the sequence. A **cycle** is the same with the addition of a edge $(v_k, v_1) \in E$. A **walk** is a path except the vertices can be repeated.
- A formula is in conjunction normal form if variables are or'ed together inside a clause and then clauses are and'ed together: $((x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_4 \vee x_5))$. Disjunctive normal form is the opposite $((x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_2} \wedge x_4 \wedge x_5))$.

Sample undecidable problems

AcceptOnInput:	$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts on } w\}$
HaltsOnInput:	$Halt_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and halts on input } w\}$
HaltOnBlank:	$HaltB_{TM} = \{\langle M\rangle \mid M \text{ is a TM \& } M \text{ halts on blank input}\}$
Emptiness:	$E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \varnothing\}$
Equality:	$EQ_{TM} = \left\{\langle M_A, M_B\rangle \;\middle|\; \begin{array}{l} M_A \text{ and } M_B \text{ are TM's} \\ \text{and } L(M_A) = L(M_B) \end{array}\right\}$