1a 1b 2 3a 3b 4 5a 5b 5c 6d 5e

1a) First, we will rewrite this problem as: $\exists a_1 \forall a_2 \exists a_3 \forall a_4 \exists a_5 \ldots Q_{a_n} \left(\sum_{i=1}^{n} M[i, a_i] = 0\right)$ where Q is $\exists$ if n is odd and $\forall$ if n is even. To solve this problem, we can use a recursive algorithm that alternates between quantifiers. If the current quantifier is $\exists$, we recursively check whether there exists an integer $a_i \in \{1, 2 \ldots n\}$ such that the sum of the selected integers up to row i, and considering player 2's moves from following rows can cause the sum to be zero. If the current quantifier is $\forall$, we check for all $a_i \in \{1, 2, \ldots n\}$ whether there exist moves by player 1 such that the sum of integers is zero, no matter what player 2 does. At each recursive level, we check one row at a time, which results in a branching factor n for each players turn. It recurses until the final row is reached, where the sum of selected integers can be checked. The recursion depth is proportional to the number of rows n and we keep a polynomial amount of space to store the current state and the players' choices at each level. Since each recursive call only requires a fixed amount of space to store the current state, the overall space used is $O(n \cdot \text{poly}(|\langle M \rangle|))$, where $|\langle M \rangle|$ is the size of the matrix M. This shows that the space complexity is polynomial, so the problem can be solved in space proportional to $\text{poly}(|\langle M \rangle|)$. Thus, L is in PSPACE since we can decide whether player 1 has a winning strategy in polynomial space.

```
start
↓
check row 1 (∃)
  ↳ for each choice a1 in row 1: check row 2 (∀)
      ↳ for each choice a2 in row 2: check row 3 (∃)
          ↳ ... check row n (Q)
              ↳ if sum of selected integers == 0:
                    player 1 wins
              ↳ else:
                    player 1 loses
```

1b) We will prove L is PSPACE-hard by reducing from Q-SAT. Consider the instance of Q-SAT with $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots Q x_n \phi(x_1, x_2 \dots x_n)$ where the CNF ϕ has k clauses and we assume n is even so that x_n matches our pattern. In the first n rows of M, we populate each row with two values corresponding to x^i_{true} and x^i_{false} where these values are derived from SUBSET SUM reduction for ϕ. Each row corresponds to a variable x_i with x^i_{true} and x^i_{false} representing the assignments of x_i. The next 2k-1 rows, for the i^{th} clause, a row contains the values $0, A1_i, 2A1_i$ where A1 is the value from the SUBSET SUM reduction. After each of these clause rows, there is a zero row, alternating between rows with nonzero values and ones filled with zeros until all 2k-1 rows are filled. The last row has -B values and B is the target value from ϕ's SUBSET SUM reduction. This reduction runs in polynomial time. In the QSAT game, player 1 alternates choosing truth values for $x_1, x_2 \dots x_n$ trying to end up with a satisfying assignment. A YES instance of QSAT implies the existence of a satisfying assignment for ϕ, meaning player 1 can select these truth values in the first phase of the game. After the first n rows, player 1 can pick the values (0, 1, or 2) for each of k clauses to make sure the sum so far is B, because player 2 has to pick a row of zeros so they don't change the sum. In the last row, player 2 must pick -B, so the sum is zero and player 1 wins, so yes maps to yes. For a NO instance, player 2 can prevent the assignment from the first phase from satisfying. There will be at least one clause where the sum of the selected values so far is zero because all the literals of that clause are false. Thus, player 1 cannot do anything to make the sum B and as a result, the total sum will not be zero. Player 1 does not win and no maps to no. By reducing QSAT to this game in polynomial time, we show that solving this problem is at least as hard as solving QSAT. Since QSAT is PSPACE-complete, the problem is PSPACE-hard.

x^1_{true}	x^1_{false}	x^2_{true}	x^2_{false}	x^3_{true}	x^3_{false}
x^2_{true}	x^2_{false}	x^3_{true}	x^3_{false}	x^1_{true}	x^1_{false}
x^3_{true}	x^1_{false}	x^3_{true}	x^1_{false}	x^3_{true}	x^1_{false}
0	$A1_1$	$2A1_1$	0	0	0
0	0	0	0	0	0
0	$A1_2$	$2A1_2$	0	0	0
0	0	0	0	0	0
0	$A1_3$	$2A1_3$	0	0	0
0	0	0	0	0	0
-B	-B	-B	-B	-B	-B

2) We will start by showing that L is not regular. We'll do proof by contradiction with the pumping lemma by showing that a string $xyz \in L$, but $xy^iz \notin L$. Assume L is regular and let p be pumping length and consider the string $w = a^p b^{p!+p}$ such that $w \in L$. The power of b >> a. We can split w to xyz. Given the conditions $|y| > 0$ and $|xy| \leq p$, the only way to split $w = xyz$ is as follows:

- $x = a^{p-m}$ where $1 \leq m < p$
- $y = a^n$ where $1 \leq n < p$
- $z = a^k b^{p!+p}$ where $0 \leq k < p$ and $m + n + k = p$ such that if we pump on y, we have $xy^iz = a^{p-m} a^{in} a^k a^{p!+p} = a^{p-m+k+in} b^{p!+p}$.

Note that since $1 \leq n < p$, n is a factor of $p!$ such that if we have $i = \frac{p!}{n} + 1$ and since $p - m + k + n = p$, we get $xy^iz = a^{p-m+k+in} b^{p!+p} = a^{p-m+k+n+(i-1)n} b^{p!+p} = a^{p+(i-1)n} b^{p!+p} = a^{p+((p!/n)+1-1)n} b^{p!+p} = a^{p+p!} b^{p+p!}$.

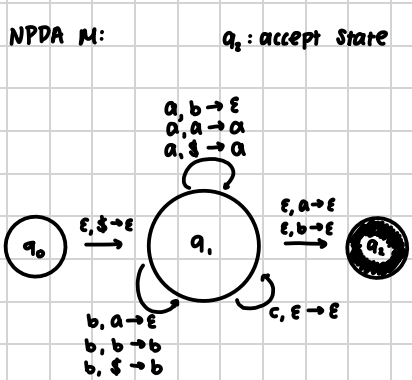
$\Rightarrow\Leftarrow$ xy^iz is a string with an equal number of a's and b's such that $xy^iz \notin L$, so L is not regular.

We will now show L is context free by building an NPDA M that recognizes L. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2\}$ as all states, $\Sigma = \{a, b, c\}$ represents the tape alphabet, $\Gamma = \{\$, a, b\}$ represents the stack alphabet, q_0 is the start state, and $F = \{q_2\}$ is accept. δ is our transition function defined by:

- start by pushing \$, so it's at the bottom of the stack
- if we read an a from the tape and b is at the top of the stack, pop a b
- if we read an a and \$ or b is at the top of the stack, push an a
- if we read a b and a is at the top of the stack, pop an a
- if we read a b and \$ or b is at the top of the stack, push a b
- If we read a c, the stack doesn't change
- when we reach the end of the tape, we read a ε, if the top of the stack is an a or b, pop and transition to the accept state. Thus, we don't transition to q_2 if \$ is at the top of the stack

Our NPDA only accepts if the number of a's does not equal the number of b's. Since we have built an NPDA that recognizes L, L is context free. L is context free but not regular.

NPDA M: q_2: accept state



3a) Since L is regular, there exists a DFA $M=(Q,\Sigma,\delta,q_0,F)$ that recognizes L. We will also construct NFA M' that decides L_{-w}. Let M' have $n+1$ copies of M for some $w=w_1w_2w_3\dots w_n$ where $|w|=n$. We let the first and last copy of M contain all of M's transitions and the other $n-1$ copies will only have states — no normal transitions from M, just ε-transitions. The start state of M' is the start state of the first copy of M. The accept states of M' are the accept states of the $(n+1)^{th}$ copy of M. For transitions in M between two states q_1 and q_2 labeled with w_i, we add an ε transition linking q_1 in the i^{th} copy of M to q_2 in the $(i+1)^{th}$ copy of M. This ensures that when reading w_i, we transition from copy i of M to $i+1$ to show how we track progress through w.

Now, consider a string $xwy \in L$ where $w=w_1w_2\dots w_n$. We follow the normal transitions of M in the first copy, processing x. When reading w_1, we use the ε-transition to move to the second copy of M, then continue for $w_2,w_3\dots w_n$. After reaching the $n+1^{th}$ copy of M, we follow the normal transitions of M to read y, where we reach an accept state of M as $xwy\in L$, which corresponds to an accept state in M', so $xy\in L_{-w}$.

If xy is accepted by M', then there exists a path through M' from the first copy of M to the last. The ε transitions taken in M' can construct a string w. Thus, we can construct xwy, where $w=w_1w_2\dots w_n$ which must be accepted by M. Therefore, if L is regular, then L_{-w} is regular as well.

b) If L is RE, there exists a turing machine M that recognizes L. We define a TM M' that recognizes L_{-w}. Consider a string w where we define M' as a decider of L_{-w} where $M=$ "on input s: "

1. we create $|s|+1$ new strings x_i where each $x_i=s_1s_2\dots s_i w s_{i+1}s_{i+2}\dots s_{|s|}$ for all i between 0 and $|s|$ such that we simulate xwy where $x=s_1s_2\dots s_i$ and $y=s_{i+1}s_{i+2}\dots s_{|s|}$
2. we run M for each x_i such that we are running M $|s|+1$ times simultaneously
3. if any instance of M halts and accepts, then we also halt and accept. Otherwise, if we reach $|x_i|$ steps and all instances of M haven't halted, we halt and reject

Our turing machine accepts if there exists a partition of $s=xy$ that we can insert w into the partition to form $xwy\in L$. Since we can build a TM that decides L_{-w}, then L_{-w} is RE.

4) From the Time Hierarchy Theorem, for any k there exists languages decidable in $O(n^{3k})$ time that cannot be decided in $O(n^a)$ time. For $k=6$, we get a language P that cannot be in $O(n^b)$ time, but is in NP even if an NP-complete problem could be solved in $O(n^b)$. Also, an NP problem might require a reduction that runs in $O(n^b)$ time before solving the NP-complete problem in $O(n^3)$. Thus, the proof is not possible as we would need to decide a language by reduction.

5a) Vertices are an independent set if none of them are adjacent. For (G, k) and subset S of vertices, we can check in polynomial time that $|S| \geq k$ and S is independent. Thus, INDEPENDENT SET is in NP. We will prove it is NP-hard by reducing from 3-SAT. Given a boolean formula ϕ with M clauses and n variables, where each clause has at most 3 literals, we construct G_1 such that it has an independent set of size k iff ϕ is satisfiable. For each variable x_i, create two vertices: one for x_i and one for $\neg x_i$. Add an edge between these vertices as only one can be chosen in an independent set. For each clause $C_j = (a \vee b \vee c)$, create a triangle where each vertex corresponds to a literal. If a literal x_i appears in a clause and $\neg x_i$ appears in another clause, their corresponding vertices in the graph stay separate. Each variable node has degree ≤ 3 and each clause node has degree ≤ 3, so the graph has a max degree 4. Set $k = M+n$, where M is the number of clauses and n is the number of variables. The independent set will contain at least one vertex from each clause and one vertex per variable. If ϕ is satisfiable, there exists a truth assignment where at least one literal in each clause is true. For each x_i, pick either x_i or $\neg x_i$ based on the satisfying assignment. For each clause, choose one literal that is true. Since these literals belong to different triangles, the set stays independent. This gives an independent set of size $n+M$, which is $\geq k$. Thus, yes maps to yes. If ϕ is not satisfiable, then some clause will have no satisfied literal for every possible assignment. The corresponding clause triangle cannot contribute a vertex to the independent set, so it can't reach size k. If ϕ is unsatisfiable, there is no independent set with size k in G_1, so no maps to no. Thus, INDEPENDENT SET is NP-complete.

b) A forest is a graph with no cycles. We will use DFS to do this. During the DFS, each vertex is marked as unvisited, visiting, or visited. During the DFS, we look for edges that point to a vertex in the recursion stack (visiting). If such an edge is found, then there is a cycle as there is a path back to an ancestor. If none are found, the graph is a forest. The DFS runs in $O(|V|+|E|)$ where $|V|$ is the number of vertices and $|E|$ is edges. The cycle check is done in constant time during the DFS for each edge. The overall time complexity is polynomial in the size of G_1. After checking G_1 is a forest, we show that any forest is 3-colorable. A forest is a collection of trees and any tree is 2-colorable. So start at vertex V and color it color #1, then its neighbors are color #2. Continue and alternate color. As there are no cycles, two neighbors will never be the same color. We can apply this scheme to each tree in the forest, so each forest is 2-colorable. 2-colorable is just a case of 3-colorable, so the forest is 3-colorable. Thus, if the graph is a forest, we accept the graph and reject otherwise. Since the DFS and 3-colorability can be checked in polynomial time, the problem is in P.

d) First, note that there are a polynomial number of ways to choose at least 750 clauses from the first 1000 clauses of the formula. For each selection of at least 750 clauses from the first 1000, we combine these selected clauses with the the remaining clauses after the first 1000 to form a new 2-CNF formula. Since 2-SAT is solvable in polynomial time, we can check the satisfiability of each 2-CNF instance formed by combining the chosen clauses from the first 1000 with the remaining clauses in polynomial time. Basically for each combination of at least 750 clauses from the first 1000, we create a 2-SAT instance and solve it in polynomial time. There are at most 2^{1000} instances, which is bounded by a polynomial in 1000. Since each 2-SAT can be solved in polynomial time, the total time required to solve all 2-SAT instances is polynomial. If any of the at most 2^{1000} 2-SAT instances is satisfiable, we return YES. If all are NO instances, we return NO. Since the largest possible number of combinations is polynomial and each check takes polynomial time, the whole runtime is polynomial. Thus the language is in P.

e) To show L is in NP, all of the first 1000 clauses must be satisfied by the assignment and $3/4$ of the remaining clauses must be satisfied given a truth assignment This check can be done in linear time with respect to the number of clauses. L is in NP. To show L is NP-hard, we reduce from MAX-2-SAT. In MAX-2-SAT, we are given a 2-CNF formula ϕ with m clauses and our goal is to determine if there exists a truth assignment that satisfies at least K clauses. We add the clause $(x_i, \overline{x_i})$ 1000 times and this will form the first 1000 clauses of the formula in L. The remaining $10m$ clauses of the formula in L correspond directly to the clauses in ϕ. MAX-2-SAT guarantees that if there's an assignment that satisfies $\geq K = 7m$ clauses, the corresponding instance in L will satisfy all of the first 1000 trivially satisfied clauses and $\geq 7m$ of the remaining $10m$ clauses. We add $n = 2m$ more trivially satisfied clauses. Now, the total number of clauses is $12m$. The truth assignment that satisfies $\geq 7m$ clauses in MAX-2-SAT will ensure that $\geq 9m$ of the $12m$ clauses are satisfied in L since the first 1000 clauses are satisfied, $\geq 7m$ of the next $10m$ clauses from ϕ are satisfied, and the remaining $2m$ are trivially satisfied. Thus, the formula in L will satisfy at least $3/4$ of the remaining $12m - 1000$ clauses iff the MAX-2-SAT instance has a YES solution where $\geq 7m$ clauses are satisfied. Therefore L is NP-complete.