5. For problem 3:
Time and Space Complexity Analysis

1. Constructor: Election()
Time Complexity: O(1)
Space Complexity: O(1)
Explanation: Just initializes empty data structures with constant time operations.

2. initializeCandidates(List<String> candidates)
Time Complexity: O(n)
Space Complexity: O(n)
Explanation:
Clears existing structures: O(1)
Iterates through n candidates: O(n)
Each put and offer operation is O(1) (for HashMap) and O(log n) (for heap), but since we're doing n of them, it's O(n log n) for the heap operations
Dominant term is O(n log n), but since we're initializing with all candidates at once, some implementations consider this O(n) due to heap construction optimizations

3. setTotalVotes(int p)
Time Complexity: O(1)
Space Complexity: O(1)
Explanation: Simple assignment operation

4. castVote(String candidate)
Time Complexity: O(log n)
Space Complexity: O(1)
Explanation:
HashMap get/put: O(1)
Heap insertion (offer): O(log n) where n is number of candidates
Total is dominated by heap operation

5. castRandomVote()
Time Complexity: O(n) for worst case, O(1) average case
Space Complexity: O(1)
Explanation:
candidates.keySet() to ArrayList: O(n)
Random selection: O(1)
Then calls castVote: O(log n)
Dominated by O(n) conversion to ArrayList

6. rigElection(String candidate)
Time Complexity: O(n log n)
Space Complexity: O(1)

Explanation:
Reset all votes in HashMap: O(n)
Clear heap: O(n)
Add rigged candidate: O(log n)
Filter others: O(n)
Add specific votes: O(log n) per addition (but constant number)
Dominated by O(n) operations and heap operations

7. getTopKCandidates(int k)
Time Complexity: O(n log n)
Space Complexity: O(n)
Explanation:
Stream operations:
sorted(): O(n log n) (using TimSort)
limit(): O(1)
map/collect: O(n)
Dominated by sorting

8. auditElection()
Time Complexity: O(n log n)
Space Complexity: O(n)
Explanation:
Same as getTopKCandidates but prints instead of returning
Dominated by sorting operation