Question 1:

The solution's time complexity is primarily linear with respect to the number of rows in the input CSV files. Most operations, such as reading CSVs, filtering with conditions, grouping data with groupby, and merging datasets with merge, operate in O(n) time, where *n* is the number of rows. However, for reasonably sized datasets, these operations are efficient due to Pandas' optimized backend in C.

A key tradeoff made in this solution was prioritizing simplicity, readability, and development speed over absolute performance and scalability. By using Pandas, the code remains concise and easy to follow, which is ideal for a proof of concept or local analysis. That said, this approach is memory-bound and large CSVs can cause memory pressure or slowdowns on limited environments.

If the dataset were to grow significantly, migrating to a lightweight SQL database such as PostgreSQL would be a practical improvement. SQL excels at handling relational data joins, filters, and aggregations, all core tasks in this solution, and offers better performance on larger datasets, particularly when proper indexing is used. With SQL, we also gain the ability to write more maintainable and declarative queries, leverage query planners, and reduce memory usage by streaming data from disk instead of loading everything into memory.

Question 2:

To support future columns like "Bill Voted On Date" or "Co-Sponsors", I would modularize the data processing pipeline to make it more extensible and readable. For instance, "Bill Voted On Date" is a field that naturally belongs to the votes dataset, so I would ensure that the date column is preserved when merging votes with bills. If multiple vote dates exist for the same bill, we could decide to display the first (min), last (max), or even all dates, depending on the reporting requirement.

For "Co-Sponsors", we are dealing with a many-to-many (N:M) relationship: a single bill can have multiple co-sponsors, and a legislator can co-sponsor multiple bills. This would be best represented using an associative table (e.g., bill_cosponsors.csv) that maps bill_id to legislator_id. The implementation would involve joining this table with the legislators' names and grouping by bill_id, then concatenating the names into a comma-separated string for each bill. This string could then be added to the final CSV export.

In general, anticipating future requirements would involve designing the processing steps to be composable and clearly separated by function, which makes adding new columns or transformations more straightforward.

Question 3:

If instead of reading the full datasets from CSV files, the program received a list of specific legislators or bills to generate reports for, I would refactor the solution to support filtered data processing from the start.

This change would require accepting inputs either via command-line arguments, a configuration file, or a JSON list containing the desired legislator_ids or bill_ids. After loading the full datasets (or ideally querying a database), the program would filter the data early in the pipeline to only process and output information related to the specified entities.

The main benefit of this change is performance and flexibility — especially when generating reports for selected individuals or bills, without loading or computing unnecessary data.

Question 4:

Approximately 3 hours, including:

- Data exploration
- Script development and testing
- Refactoring for clarity
- Writing unit tests
- Creating documentation (README)
- Responding to edge cases and errors (e.g., missing columns, NaNs)