Criterion	**Solution 1 (Pipes and Filters)**	**Solution 2 (Main/Subroutine)**
Ease of modifying the algorithm	Easier to modify individual parts.	Harder to modify since the algorithm is integrated into one solution
Ease of changing data representation	Data representation (list of permutations) is separated from processing logic, making it easier to change.	Harder to change, as data representation is directly tied to logic
Ease of adding additional functions	Easier to add functions, as new modules can be added without interfering with existing logic.	Adding new functions is more challenging
Performance	Less efficient	More efficient, as it uses step-by-step checking
Reusability	Code is modular and easily extendable for other tasks	Easier to reuse for tasks requiring stepwise recursive problem-solving

Ease of modifying the algorithm: In *pipes and filters*, each stage is separate, so individual components can be modified without affecting the whole structure. In *Main/Subroutine*, changing one function could require a review of other related parts due to close interdependencies.

Ease of changing data representation: In the first solution, data representation is isolated in GenerateColumnChoices, making it simpler to adjust. In the second solution, data is closely tied to the solution-checking algorithm, so any changes impact the safety-checking (is_safe_to_place_queen) logic as well.

Ease of adding additional functions: The *pipes and filters* approach allows adding modules to process data, such as filtering or exporting results. In the *Main/Subroutine* approach, adding functions is less straightforward because the recursive algorithm is more tightly integrated.

Performance: The second solution is more efficient as it checks the validity of queen positions during solution construction and doesn't generate unnecessary permutations.

Reusability: *pipes and filters* is useful for tasks with modular, step-by-step processing needs. However, for recursive, depth-first tasks like this, the

Main/Subroutine approach is preferable, as it's more concise and naturally suited to this type of problem.

Criterion	**Abstract Data Types (ADT) Solution**	**Implicit Invocation (Event-Driven) Solution**
Ease of modifying the algorithm	Changing algorithms in specific modules is relatively straightforward	More complex
Ease of changing data representation	Moderately easy	Harder
Ease of adding additional functions	Easier	Slightly harder
Performance	Potentially more performant due to the synchronous flow	Generally less performant because of event dispatching and observer pattern overhead
Reusability	More straightforward for small to medium-scale tasks where direct control and simplicity are preferred	Better suited for larger, more modular systems that may need more dynamic event-driven handling, although the complexity could be unnecessary for simpler tasks.

Changeability of Algorithms: The ADT approach enables direct access and management within each class, so changes in algorithms are more isolated. In contrast, the event-driven approach interconnects modules through events, making it more complex to update one module without considering its impact on the rest of the event chain.

Data Representation Changes: With the ADT approach, each module independently manages its data, so modifying one representation does not necessarily impact others. The event-driven approach, however, requires a shared understanding of data formats across observers, making it more challenging to adapt to changes in data structures.

Adding New Functions: In the ADT solution, adding functionality is generally as simple as adding new methods. However, in the event-driven approach, adding new functions requires creating new event types or observers and managing how these events interact with other existing modules.

Performance: The ADT approach avoids the additional complexity of dispatching and handling events, likely resulting in better performance, especially for simpler workflows. The event-driven approach can introduce overhead in dispatching events and updating observers, which can slow down the process, especially as the system scales.

Preferred Reusable Solution: For simpler, straightforward tasks, the ADT solution provides a more structured, easily understandable framework. The event-driven solution could be more beneficial in large, modular systems where extensibility and flexibility are prioritized. However, given the current context and requirements, the ADT-based solution would likely be easier to reuse.