Gas Optimization Tips Checklist

1. Optimize Storage Usage

- **Use Local Memory Variables:** Transfer storage data to memory variables when working within a function, then write back only if needed, as storage read/write is far more expensive.
- **Avoid Repeated Storage Access:** Store values read from storage in a local variable if accessed multiple times within the same function.

2. Optimize Data Types and Struct Packing

- **Use Smaller Data Types:** Where values allow, choose the smallest data types possible (e.g., `uint8`, `uint16`). Compact data saves gas.
- **Struct Packing:** Arrange struct variables to fit within 32-byte slots. Solidity packs multiple minor variables into a single storage slot, reducing cost.
- **Avoid Strings and Dynamic Arrays in Storage:** Use them sparingly, as they can significantly increase gas costs; if suitable, try `bytes32` or smaller fixed-size arrays.

3. Reduce Loop Costs with Efficient Logic

- **Limit Loop Operations in Storage:** Minimize storage read/write operations within loops. Use local variables to hold interim results before writing to storage.
- **Batch Processing:** For large loops, split tasks across multiple transactions to avoid hitting the block gas limit and excessive costs.

4. Use Constants and `immutable` Variables

- **Define Constants:** Declare values that won't change as `constant` to save gas. Constants are stored directly in bytecode, avoiding storage costs.
- **`Immutable` Variables:** Use `immutable` for variables initialized only once (like constructor parameters), allowing changes without the extra storage overhead.

5. Structure Efficient Contract Architecture

- **Modularize with Caution:** Avoid deep nesting, which can increase gas costs for function calls.
- **Library Use:** Offload reusable logic into libraries instead of duplicating code, reducing contract size and deployment cost.

6. Minimize External and Cross-Contract Calls

- **Consolidate External Calls:** Minimize the number of calls to external contracts and batch where possible. Each external call incurs additional gas overhead.
- **Interface Optimization:** Ensure external calls are optimized only to retrieve or send required data, keeping functions streamlined.

7. Utilize `view` and `pure` Functions

- **Design Read-Only Operations:** Use `view` for reading from the blockchain without changing state, costing zero gas when called externally.
- **Pure for Calculations:** Use `pure` functions for computations without state access, reducing overall gas if logic is reused.

8. Optimize Event Logging and Parameterization

- **Log Essential Data Only:** Each additional parameter logged in an event increases gas costs, so only log necessary data.
- **Avoid Excessive Events:** Use events to log significant actions but avoid frequently changing or redundant information.

9. Implement Upgradable Patterns via Proxy Contracts

- **Proxy Contract Use:** Employ proxy patterns to avoid redeploying the entire contract for upgrades, reducing costs over time.
- **Separate Logic and Data:** To update logic without modifying storage, keep the logic in one contract and data in a separate proxy contract.

10. Eliminate Redundant Code and Streamline Operations

- **Remove Unused Code:** Eliminating unused variables, functions, or unnecessary calculations saves gas and reduces complexity.
- **Combine Conditionals and Avoid Unnecessary Checks:** Streamline conditionals to minimize steps, reducing the gas cost for each function call.