### Computational Inefficiencies and Anti-patterns:

1. **getPriority Function**:

* **Issue**: The getPriority function uses a switch statement to return a priority number based on blockchain type. While functional, it is not scalable or maintainable for large numbers of blockchain types.
* **Improvement**: Use a data structure like an object or Map for blockchain priorities to improve readability and maintainability.

1. **Sorting and Filtering Logic**:

* **Issue**: In useMemo, the sortedBalances array is filtered and sorted based on balances and prices dependencies. However, the filtering logic is not clear and contains errors (lhsPriority is undefined).
* **Improvement**: Review and correct the filtering logic to ensure it filters correctly based on getPriority and amount conditions. Use clear variable names (leftPriority, rightPriority) to avoid confusion.

1. **Formatted Balances Calculation**:

* **Issue**: The formattedBalances array is created using sortedBalances.map, which computes toFixed() on each balance.amount. This operation is performed redundantly if sortedBalances doesn't change.
* **Improvement**: Memoize the formattedBalances using useMemo to avoid recomputation on every render if sortedBalances and prices haven't changed.

1. **Rendering Rows**:

* **Issue**: Rendering each WalletRow component directly inside the return statement without handling potential large lists efficiently.
* **Improvement**: Consider using a virtualized list or pagination for rendering large lists of WalletRow components to improve performance and avoid rendering all rows at once.

1. **Dependency Array in useMemo**:

* **Issue**: The dependency array in useMemo includes balances and prices, which may lead to unnecessary recalculations if prices changes frequently.
* **Improvement**: Ensure that useMemo dependencies are correctly chosen to optimize performance and prevent unnecessary recalculations.

### Explanation of Improvements

* **Data Structure for Priorities**: Introduced blockchainPriorities as an object to store blockchain priorities. This enhances readability and allows easy additions or modifications of priorities.
* **Correct Sorting and Filtering**: Simplified the sorting logic using blockchainPriorities object directly in sorting function. Ensured filtering logic accurately filters out balances based on priority and amount conditions.
* **Memoization**: Utilized useMemo effectively to memoize sortedBalances, formattedBalances, and rows. This prevents unnecessary recalculations on re-renders when dependencies (balances, blockchainPriorities, prices) haven't changed.
* **Improved Rendering**: Rendered WalletRow components efficiently using map inside useMemo, ensuring that the rendering logic is separated from data processing logic.
* **Dependency Optimization**: Carefully selected dependencies for useMemo to optimize performance by preventing unnecessary recalculations. This ensures that expensive operations like sorting and formatting are only performed when necessary.