ERROR REPORT and IMPROVEMENT

I. Inefficiencies and anti-patterns:

1. **Inconsistent Function Return and Conditions:**

   - In ***sortedBalances***, the conditions for filtering and sorting are confusing and potentially flawed. The ***lhsPriority*** is **undefined**, which should be **balancePriority**.
   - The filtering logic within ***useMemo*** is inefficient and incorrect. It filters balances with **non-positive** amounts but also returns true when the priority is greater than -99, which is not coherent.
2. **Incorrect Sorting and Filtering Logic:**

   - The sorting and filtering logic in ***useMemo*** is **mixed**, which should be separated for clarity and efficiency.
   - The comparison in the ***sort*** function does **not** correctly handle **equality**.
     - The return values in the ***sort*** method comparison are 1 and -1, which is correct, but **equality** should return **0**.
3. **Inefficient Data Mapping:**

   - The sortedBalances array is created and then formattedBalances and rows arrays are separately created based on it. These can be combined into a **single** mapping operation.
4. **Potential Redundant State Updates:**

   - ***Prices and balances*** are dependencies for ***useMemo***, causing **recalculations** even if only **one** of them **changes**.
5. **Missing Error Handling and Type Checking:**

   - Error handling for **missing** or **undefined** values in prices is **missing**, which can cause runtime errors.

II. Improvement:

- Filtering and Sorting Logic:
  - The ***filter*** function now correctly filters balances that have a valid priority and a positive amount.
  - The ***sort*** function has been updated to return **0** when the priorities are **equal**, ensuring proper comparison.

```
const sortedBalances = useMemo(() => {
   return balances
     .filter((balance: WalletBalance) => {
       const priority = getPriority(balance.blockchain);
       return priority > -99 && balance.amount > 0;
```

```
    })
    .sort((lhs: WalletBalance, rhs: WalletBalance) => {
      const leftPriority = getPriority(lhs.blockchain);
      const rightPriority = getPriority(rhs.blockchain);
      if (leftPriority > rightPriority) return -1;
      if (leftPriority < rightPriority) return 1;
      return 0;  // When priorities are equal
    });
}, [balances]);
```

- State Dependency Clarification:
  - ***sortedBalances*** and rows are recalculated only when balances or prices change.
  - The recalculation of rows only happens if ***sortedBalances*** or **prices** are updated, avoiding unnecessary computations.
  - The dependency on **prices** has been **removed** since sorting and filtering only depend on **balances**

```
const sortedBalances = useMemo(() => {
  return balances
    .filter((balance: WalletBalance) => {
      const priority = getPriority(balance.blockchain);
      return priority > -99 && balance.amount > 0;
    })
    .sort((lhs: WalletBalance, rhs: WalletBalance) => {
      const leftPriority = getPriority(lhs.blockchain);
      const rightPriority = getPriority(rhs.blockchain);
      if (leftPriority > rightPriority) return -1;
      if (leftPriority < rightPriority) return 1;
      return 0;  // When priorities are equal
    });
}, [balances]);
```

- Proper Price Handling:
  - Checked if the **price** exists before calculating **usdValue** to avoid runtime errors when the price is **undefined**.

```
const rows = useMemo(() => {
  return sortedBalances.map((balance: WalletBalance, index: number) => {
    const usdValue = prices[balance.currency] ? prices[balance.currency]
* balance.amount : 0;
    return (
      <WalletRow
        className={classes.row}
```

```
          key={index}
          amount={balance.amount}
          usdValue={usdValue}
          formattedAmount={balance.amount.toFixed(2)}
        />
      );
    });
  }, [sortedBalances, prices]);
```

- Combined Mapping Operations:
  - Combined mapping and formatting operations for **balances** into a **single *useMemo*** to improve efficiency and clarity.

```
const rows = useMemo(() => {
  return sortedBalances.map((balance: WalletBalance, index: number) => {
    const usdValue = prices[balance.currency] ? prices[balance.currency]
* balance.amount : 0;
    return (
      <WalletRow
        className={classes.row}
        key={index}
        amount={balance.amount}
        usdValue={usdValue}
        formattedAmount={balance.amount.toFixed(2)}
      />
    );
  });
}, [sortedBalances, prices]);
```