Problem 3: Messy React - with comments

1. Use of `any` type for blockchain parameter

```
const getPriority = (blockchain: any): number => {
        switch (blockchain) {
            case 'Osmosis':
                return 100
            case 'Ethereum':
                return 50
            case 'Arbitrum':
                return 30
            case 'Zilliqa':
                return 20
            case 'Neo':
                return 20
            default:
                return -99
        }
    }
```

Using any type defeats the purpose of TypeScript. We should use specific type if necessary:

```
const getPriority = (blockchain: string): number => { ... };
```

2. Improve the `getPriority()` function for more efficient and readable

By using an object for the mapping instead of a `switch` statement. This change can improve lookup performance and maintainability:

```
const priorities = {
  'Osmosis': 100,
  'Ethereum': 50,
  'Arbitrum': 30,
  'Zilliqa': 20,
  'Neo': 20
};

const getPriority = (blockchain: string): number => {
  return priorities[blockchain] ?? -99;
}
```

3. Redundant `if` Condition

The nested `if` statement when we filter balances can be combined for clarity and efficiency:

```
const sortedBalances = useMemo(() => {
    return balances.filter((balance: WalletBalance) => {
        const balancePriority = getPriority(balance.blockchain);
        return balancePriority > -99 && balance.amount <= 0;
    }).sort(
    // ....sort section
    );
}, [balances, prices]);
```

4. Missing Return Statement in `sort()` method

The `sort` comparator should always return a value, even when priorities are equal.

```
    const sortedBalances = useMemo(() => {
        return balances.filter((balance: WalletBalance) => {
        //...filter section
        }).sort((lhs: WalletBalance, rhs: WalletBalance) => {
            const leftPriority = getPriority(lhs.blockchain);
            const rightPriority = getPriority(rhs.blockchain);
            if (leftPriority > rightPriority) {
                return -1;
            } else if (rightPriority > leftPriority) {
                return 1;
            }
            return 0;
        });
    }, [balances, prices]);
```

5. Confusing use of `prices`

The dependency array includes `prices`, but `prices` is not used in the computation. If `prices` are not needed, they should be removed from the dependency array to avoid unnecessary re-computations.

6. Repeated Calculations

The mapping over `sortedBalances` to create `formattedBalances` and then mapping over `formattedBalances` to create `rows` can be combined into a single map operation to avoid iterating twice.

```
const rows = sortedBalances.map((balance: WalletBalance, index:
    const formatted = balance.amount.toFixed();
    const usdValue = prices[balance.currency] * balance.amount;
    return (
        <WalletRow
            className={classes.row}
            key={index}
            amount={balance.amount}
```

```
            usdValue={usdValue}
            formattedAmount={formatted}
        />
    );
});
```