Problem 3

Given code:

```
interface WalletBalance {
  currency: string;
  amount: number;
}
interface FormattedWalletBalance {
  currency: string;
  amount: number;
  formatted: string;
}

class Datasource {
  // TODO: Implement datasource class
}

interface Props extends BoxProps {

}
const WalletPage: React.FC<Props> = (props: Props) => {
  const { children, ...rest } = props;
  const balances = useWalletBalances();
  const [prices, setPrices] = useState({});

  useEffect(() => {
    const datasource = new Datasource("https://interview.switcheo.com/prices.json");
    datasource.getPrices().then(prices => {
      setPrices(prices);
    }).catch(error => {
      console.err(error);
    });
  }, []);

  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
    }
  }
```

```
  const sortedBalances = useMemo(() => {
    return balances.filter((balance: WalletBalance) => {
      const balancePriority = getPriority(balance.blockchain);
      if (lhsPriority > -99) {
         if (balance.amount <= 0) {
           return true;
         }
      }
      return false
    }).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;
      }
    });
  }, [balances, prices]);

  const formattedBalances = sortedBalances.map((balance: WalletBalance) => {
    return {
      ...balance,
      formatted: balance.amount.toFixed()
    }
  })

  const rows = sortedBalances.map((balance: FormattedWalletBalance, index: number) =>
{
    const usdValue = prices[balance.currency] * balance.amount;
    return (
      <WalletRow
        className={classes.row}
        key={index}
        amount={balance.amount}
        usdValue={usdValue}
        formattedAmount={balance.formatted}
      />
    )
  })

  return (
    <div {...rest}>
      {rows}
    </div>
  )
}
```

Computational inefficiencies and anti-patterns found:

- The ‘FormattedWalletBalance’ interface is defined but not used in the code —> can be removed.
- The ‘formattedBalances’ array is created by mapping over the ‘sortedBalances’ array, but it is never used. —> can be removed.
- ‘lhsPriority’ is referenced in the ‘sortedBalances’ hook but it’s not defined before. This might be an incorrect variable name.
- ‘getPriority’ is implemented imperatively using switch/case —> can be improved by using a priority map object

```
// improved version
const priorityMap = {
  Osmosis: 100,
  Ethereum: 50,
  //...
}

const getPriority = (blockchain) => priorityMap[blockchain] || -99
```

- The ‘getPriority()’ function is called for every balance in the sortedBalances array —> computationally inefficient

```
// Improved version
const priority = getPriority(balances[0].blockchain);

  const sortedBalances = useMemo(() => {
    return balances.filter((balance: WalletBalance) => {
      return balance.amount > 0;
    }).sort((lhs: WalletBalance, rhs: WalletBalance) => {
      return priority - getPriority(rhs.blockchain);
    });
  }, [balances, prices]);

//'priority' = the priority of the first item in the balances array, assuming that all
balances have the same blockchain. // --> no need to calculate the priority repeatedly
in the filter and sort functions.

// filter function is simplified to only include balances with amount greater than 0
// sort function uses the priority variable to determine the sorting order
```