Explain the virtual DOM and how React uses it to improve performance.

- **Answer:** The virtual DOM is an in-memory representation of the real DOM elements generated by React components. When the state or props of a component change, React first updates the virtual DOM. It then compares the updated virtual DOM with the previous version using a diffing algorithm, identifying the minimum number of changes required to update the real DOM. This process, known as reconciliation, helps improve performance by minimizing direct manipulations of the real DOM.

What are higher-order components (HOCs) in React? Provide an example.

- **Answer:** Higher-order components (HOCs) are functions that take a component and return a new component with additional props or functionality. They are a pattern for reusing component logic.

What is the purpose of useMemo and useCallback Hooks?

- **Answer:** useMemo and useCallback are optimization hooks in React:
  - useMemo is used to memoize expensive calculations, ensuring they are only recomputed when dependencies change. It returns a memoized value.
  - useCallback is used to memoize functions, preventing them from being recreated on every render unless their dependencies change. It returns a memoized callback.

Explain the difference between useState and useReducer. When would you use one over the other?

- **Answer:** useState is a Hook that lets you add state to functional components. It is suitable for managing simple state logic. useReducer is a Hook that is used for state management when the state logic is complex and involves multiple sub-values or when the next state depends on the previous one. useReducer is often used for managing global state in large applications or for complex state transitions.

What is the difference between React.memo and useMemo?

- **Answer:** React.memo is a higher-order component that memoizes the component itself, preventing unnecessary re-renders when the props have not changed. useMemo, on the other hand, is a Hook that memoizes a value or a function's return value to optimize performance.

Task:

Objective: Share data both from parent to child and from child to parent.

Parent Component:

```
import React, { useState } from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const [childData, setChildData] = useState('');
  const parentData = 'Data from Parent';

  const handleDataFromChild = (data) => {
    setChildData(data);
  };
  return (
    <div>
      <h1>Parent Component</h1>
      <p>Data from Child: {childData}</p>
      <ChildComponent data={parentData} sendDataToParent={handleDataFromChild} />
    </div>
  );
}

export default ParentComponent;
```

Child Component:

```
import React from 'react';

function ChildComponent(props) {
  const data = 'Data from Child';

  const handleClick = () => {
    props.sendDataToParent(data);
  };
  return (
    <div>
      <h2>Child Component</h2>
      <p>{props.data}</p>
      <button onClick={handleClick}>Send Data to Parent</button>
    </div>
  );
}

export default ChildComponent;
```

Task: Fetch and Display User Data

Objective: Fetch data from a public API and display the user information on the UI. The data should be fetched using the useEffect hook in a React functional component.

Requirements:

1. Use the useEffect hook to fetch data from the API when the component mounts.
2. Use the useState hook to manage the fetched data and any loading state.
3. Display the fetched user information on the UI.
4. Handle any errors that occur during the fetch process and display an appropriate message.

API Endpoint: You can use a public API like JSONPlaceholder to fetch user data.

- API URL: https://jsonplaceholder.typicode.com/users

```
import React, { useState, useEffect } from 'react';

function UserList() {
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        fetch('https://jsonplaceholder.typicode.com/users')
            .then((response) => {
                if (!response.ok) {
                    throw new Error('Network response was not ok');
                }
                return response.json();
            })
            .then((data) => {
                setUsers(data);
                setLoading(false);
            })
            .catch((error) => {
                setError(error);
                setLoading(false);
            });
    }, []);

    if (loading) {
        return <div>Loading...</div>;
    }

    if (error) {
```

```
    return <div>Error: {error.message}</div>;
  }

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            {user.name} ({user.email})
          </li>
        ))}
      </ul>
    </div>
  );
}

export default UserList;
```