1. What is Memoization, and How Does It Improve Performance in React Applications?

- What is Memoization?

Memoization is an optimization technique that caches the results of expensive function calls so that if the same inputs are provided again, the cached result is returned instead of recalculating the value.

In React, memoization helps prevent unnecessary re-renders and improves performance, especially in components with expensive calculations or frequent re-renders.

- How to Use Memoization in React?
  1. <u>useMemo (Memoizing Computations)</u>

- Prevents recalculating expensive values unless dependencies change.

```
import { useState, useMemo } from "react";
const ExpensiveComponent = ({ num }) => {
  const computeFactorial = (n) => {
    console.log("Computing factorial...");
    return n <= 1 ? 1 : n * computeFactorial(n - 1);
  };
  const factorial = useMemo(() => computeFactorial(num), [num]); // Memoized
  return <p>Factorial of {num} is {factorial}</p>;
};
```

  2. <u>useCallback (Memoizing Functions)</u>

- Prevents unnecessary re-creation of callback functions, useful when passing functions as props.

```
import { useCallback, useState } from "react";
const Button = ({ handleClick }) => {
  console.log("Button rendered");
  return <button onClick={handleClick}>Click Me</button>;
};
const Parent = () => {
  const [count, setCount] = useState(0);
  const memoizedHandleClick = useCallback(() => {
    console.log("Button clicked!");
  }, []); // Memoized function
```

```
 return (
  <div>
   <p>Count: {count}</p>
   <button onClick={() => setCount(count + 1)}>Increment</button>
   <Button handleClick={memoizedHandleClick} />
  </div>
 );
};
```

3. React.memo (Memoizing Components)

- Prevents unnecessary re-renders of components if props have not changed.

```
import React from "react";
const MemoizedComponent = React.memo(({ name }) => {
 console.log("Rendering...");
 return <h1>Hello, {name}</h1>;
});
```

2. What is Code Splitting in React and How Does It Improve Performance?

- What is Code Splitting?

Code splitting is a technique in React that splits the JavaScript bundle into smaller chunks, loading only the necessary code when needed.

It improves performance by reducing initial load time and delivering only required code to the user.

- How to Implement Code Splitting in React?

◆ Using React.lazy() and Suspense (for Lazy Loading Components)

```
import React, { lazy, Suspense } from "react";
const HeavyComponent = lazy(() => import("./HeavyComponent"));
const App = () => {
 return (
  <Suspense fallback={<div>Loading...</div>}>
   <HeavyComponent />
  </Suspense>
```

```
  );
};
export default App;
```

1. Dynamic Import with Webpack (for Route-Based Splitting)

```
import { lazy, Suspense } from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
const Home = lazy(() => import("./Home"));
const About = lazy(() => import("./About"));
const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Suspense>
  </Router>
);
```

2. Code Splitting Using Webpack (Automatic Optimization)

If you're using Webpack, it automatically splits large dependencies into separate chunks.

```
// Webpack config: Enable chunking
optimization: {
  splitChunks: {
    chunks: "all",
  },
}
```