Q.1 What's React and its pros and cons?

React is a popular JavaScript library for building user interfaces. It allows developers to create reusable UI components and efficiently update and render the user interface when the underlying data changes. Some pros and cons of React are:

Pros:

- Component-Based: React promotes a modular and component-based approach to UI development, making it easier to manage and reuse code.

- Virtual DOM: React uses a virtual DOM, which allows for efficient updates to the actual DOM by minimizing direct manipulations.

- Unidirectional Data Flow: React follows a unidirectional data flow, making it easier to understand and track data changes.

- Rich Ecosystem: React has a vast ecosystem with numerous libraries, tools, and community support.

- Performance: React's efficient diffing algorithm and virtual DOM implementation contribute to high-performance UI rendering.

Cons:

- Learning Curve: React has a learning curve, especially for developers new to component-based architectures and JSX syntax.

- Tooling: The React ecosystem relies on various build tools, which can sometimes lead to a complex configuration setup.

- Boilerplate Code: React can require writing additional code for managing state, handling lifecycle methods, and dealing with prop validation.

Q.2 What do you understand by Virtual DOM?

The Virtual DOM is a concept in React where a lightweight copy of the actual DOM is created and maintained in memory. It acts as a representation of the UI and allows React to efficiently update and render the real DOM when the underlying data changes. The Virtual DOM enables React to minimize direct manipulations to the DOM, which can be costly in terms of performance. React compares the Virtual DOM with the previous state of the Virtual DOM and performs the minimum required changes to update the real DOM, resulting in faster and more efficient UI updates.

Q.3 Difference between Virtual DOM vs Real DOM:

- Real DOM: The real DOM is the actual browser representation of the HTML elements on a web page. It is built and maintained by the browser and consists of a tree structure where each element is an object with properties and methods. Manipulating the real DOM directly can be computationally expensive and slow, especially when dealing with frequent updates.

- Virtual DOM: The Virtual DOM is an abstraction or lightweight copy of the real DOM that React maintains in memory. It is a representation of the UI and consists of a tree structure similar to the real DOM. React uses the Virtual DOM to track and manage updates to the UI. When the state or props of a component change, React creates a new Virtual DOM, compares it with the previous Virtual DOM, and determines the minimal set of changes needed to update the real DOM.

The key differences between the Virtual DOM and the real DOM are that the Virtual DOM is a lightweight representation in memory, while the real DOM is the actual browser representation. Manipulating the Virtual DOM is faster and more efficient than manipulating the real DOM directly, as React performs the diffing and updates on the Virtual DOM before applying the changes to the real DOM.

Q.4 What's a component? Types of components:

In React, a component is a reusable and self-contained piece of code that represents a part of the user interface. Components allow developers to encapsulate UI logic and structure into modular and independent units, making it easier to build and maintain complex UIs. There are two main types of components in React:

1. Functional Components: Functional components, also known as stateless functional components, are defined as JavaScript functions that take in `props` as arguments and return JSX to describe the UI. They are simpler, lightweight, and primarily used for presenting UI based on the provided props. Functional components don't have their own internal state.
2. Class Components: Class components are defined as ES6 classes that extend the `React.Component` class. They are more feature-rich and flexible compared to functional components. Class components have their own internal state and can utilize lifecycle methods. They are commonly used for components that require state management, event handling, and complex logic.

Additionally, React introduces the concept of Pure Components and Higher-Order Components (HOCs):

- Pure Components: Pure Components are class components that automatically implement a shallow prop and state comparison to determine if re-rendering is necessary. They optimize performance by preventing unnecessary re-renders when the props or state have not changed.

- Higher-Order Components (HOCs): HOCs are functions that take a component as an argument and return a new component. They enable code reuse and add additional behavior or data to the wrapped component. HOCs are a way to enhance or extend the capabilities of components.

Q.5 Difference between class-based and function-based components:

- Class-based components: Class-based components in React are defined as JavaScript classes that extend the `React.Component` class. They provide more features and flexibility, such as the ability to define and manage state using `this.state`, access lifecycle methods like `componentDidMount`, handle events, and implement complex logic. Class components are suitable when you need internal state management or more advanced functionality.

- Function-based components: Function-based components, also known as stateless functional components, are defined as JavaScript functions that take `props` as an argument and return JSX to describe the UI. They are simpler and lightweight compared to class components. Function-based components don't have their own internal state and can be used for presenting UI based on the provided props. They are recommended for simple, presentational components without the need for lifecycle methods or state management.

With the introduction of React Hooks in React 16.8, function-based components can also handle state and lifecycle-like functionality using built-in hooks like `useState`, `useEffect`, and more. This makes function-based components more powerful and allows them to handle more complex scenarios previously handled by class components.

Q.6 Explain React component lifecycle:

React components go through different lifecycle phases from their creation to removal. The lifecycle consists of three main phases:

- Mounting: Occurs when an instance of a component is being created and inserted into the DOM.

  - `constructor()`: The constructor method is called before the component is mounted. It is used to initialize state and bind event handlers.

  - `render()`: The render method is responsible for returning the JSX representation of the component's UI.

  - `componentDidMount()`: This method is called immediately after the component is mounted and added to the DOM. It is commonly used to initialize data fetching or subscriptions.

- Updating: Occurs when a component is re-rendered due to changes in props or state.

  - `componentDidUpdate(prevProps, prevState)`: This method is called after the component has updated and re-rendered. It allows you to perform additional actions based on changes in props or state.

- Unmounting: Occurs when a component is being removed from the DOM.

  - `componentWillUnmount()`: This method is called just before the component is unmounted and removed from the DOM. It is used for cleanup tasks like canceling subscriptions or removing event listeners.

These are some of the commonly used lifecycle methods, but React provides more lifecycle methods that can be used for specific purposes.

Q.7 Explain Prop Drilling in React & Ways to avoid it:

Prop Drilling is a term used in React when props are passed through multiple levels of nested components that don't need the props directly. It happens when an intermediate component receives a prop but doesn't use it, but it needs to pass it down

to its child components. This can lead to a cluttered and less maintainable codebase.

To avoid prop drilling, there are a few approaches:

1. Context API: React's Context API allows you to create a context that can be accessed by nested components without passing props manually. It provides a way to share data or state across components without going through all the intermediate components.

2. Component Composition: Instead of passing props down through multiple levels, consider restructuring your components to have a parent component that wraps the child components that actually

need the props. This way, the props can be passed directly to the relevant child components, avoiding the unnecessary passing of props through intermediate components.

3. Use Redux or MobX: Implementing a state management library like Redux or MobX can help avoid prop drilling. These libraries allow you to maintain a global state accessible by any component in the application, eliminating the need to pass props through multiple levels.