Q.1 Whats React and its pros and cons?

Pros of React:

1. Component-Based Architecture:

React promotes a modular approach to building UIs through its component-based architecture. Components are self-contained, reusable pieces of code that can be composed together to create complex user interfaces. This makes the codebase more maintainable, scalable, and easier to understand.

2. Virtual DOM:
   React utilizes a virtual DOM (Document Object Model) to efficiently update and render components. The virtual DOM is a lightweight representation of the actual DOM, and React uses it to perform minimal updates to the real DOM when there are changes in the data or state. This approach improves performance and makes React applications fast and responsive.

3. Declarative Syntax:
   React uses a declarative syntax, which means you describe what the UI should look like based on the current state, and React takes care of updating the DOM to reflect that state. This makes the code more predictable and easier to reason about, as you don't have to manually manipulate the DOM.

4. Rich Ecosystem and Community Support:

React has a large and active community that continuously contributes to its ecosystem. There are numerous third-party libraries, tools, and resources available to enhance development productivity, solve common problems, and provide support.

Cons of React:

- Learning Curve:
  React has a learning curve, especially for developers who are new to JavaScript or frontend development.It introduces new concepts like JSX (JavaScript XML) and the component-based approach, which may require some time to grasp fully.
- Boilerplate Code:
  React itself is a library focused on building user interfaces, so it doesn't provide solutions for all aspects of application development. As a result, developers may need to set up additional tools and libraries for state management, routing, and other functionalities. This can lead to a certain amount of boilerplate code and configuration.

- Tooling Complexity:
  React's ecosystem includes various tools and build configurations, such as Babel and Webpack, which can add complexity to the development setup. Managing tooling configurations and keeping up with updates and dependencies can be challenging for beginners.

- Steep Learning Curve for Advanced Features:
  While React's core concepts are relatively easy to grasp, mastering more advanced features, such as context, hooks, and advanced state management patterns, may require additional effort and learning.

Q.2 What do you understand by Virtual Dom?

The Virtual DOM (Document Object Model) is a concept and technique used by libraries like React to optimize
the performance of updating and rendering user interfaces. It acts as a lightweight representation of the actual DOM,
which is the tree-like structure that represents the HTML elements on a webpage.

Here's how the Virtual DOM works:

- Whenever there is a change in the state or data of a React component, the Virtual DOM gets updated with the new state and data.
- React then performs a process called reconciliation, where it compares the previous Virtual DOM with the updated Virtual DOM.
- During the reconciliation process, React identifies the differences or changes between the two Virtual DOM representations.
- Once React identifies the changes, it updates only the necessary parts of the actual DOM to reflect those changes, rather than re-rendering the entire DOM.
- React optimizes the process by performing batch updates and efficiently updating the minimal set of DOM nodes required.

Q.3 Difference between Virtual Dom vs Real Dom

Virtual Dom

The Virtual DOM (Document Object Model) is a concept and technique used by libraries like React to optimize
the performance of updating and rendering user interfaces. It acts as a lightweight representation of the actual DOM,
which is the tree-like structure that represents the HTML elements on a webpage.

Here's how the Virtual DOM works:

- Whenever there is a change in the state or data of a React component, the Virtual DOM gets updated with the new state and data.
- React then performs a process called reconciliation, where it compares the previous Virtual DOM with the updated Virtual DOM.
- During the reconciliation process, React identifies the differences or changes between the two Virtual DOM representations.
- Once React identifies the changes, it updates only the necessary parts of the actual DOM to reflect those changes, rather than re-rendering the entire DOM.
- React optimizes the process by performing batch updates and efficiently updating the minimal set of DOM nodes required.

Real DOM

In React, the Real DOM (Document Object Model) refers to the actual browser DOM that represents the structure of HTML elements in a web page. It is the standard interface provided by web browsers to interact with and manipulate web page elements using JavaScript.

When using React, the Real DOM is updated efficiently by utilizing the Virtual DOM (as explained in the previous answer). Here's how React interacts with the Real DOM:

- Initial Render: When a React component is first rendered, it generates a Virtual DOM representation of the component's UI based on the component's state and props. This Virtual DOM is then used to create the corresponding HTML elements in the Real DOM.
- Updating the Real DOM: When the state or props of a React component change, React re-renders the component. It generates a new Virtual DOM representation of the updated UI. React then performs a diffing algorithm to identify the differences between the previous Virtual DOM and the updated Virtual DOM.
- Efficient DOM Updates: After identifying the differences, React determines the minimal set of changes required to update the Real DOM. It applies these changes by manipulating the actual HTML elements in the Real DOM, adding, removing, or modifying specific elements as needed.
- Reconciliation: React optimizes the process by batching the DOM updates and performing them in an optimized manner. It updates only the specific parts of the Real DOM that have changed, rather than re-rendering the entire page.

Q.4 Whats component? Types of component

In React, a component is a reusable and self-contained piece of code that encapsulates the logic, structure, and styling of a specific part of a user interface. Components allow you to break down the UI into smaller, modular pieces, making it easier to manage and build complex applications.

In React, components can be classified into two main types:

1.Function Components
2. Class Components.

Q.5 Difference between class & function based component

1. Syntax:

Class-based components are defined as JavaScript classes that extend the React.Component base class. They use the render() method to define the component's UI.Function-based components are defined as JavaScript functions. They receive props as input and return JSX (JavaScript XML) as output, representing the component's UI.

2. State and Lifecycle:

- Class-based components have their own internal state, which allows them to manage and update data independently. They can define lifecycle methods such as componentDidMount, componentDidUpdate, etc., for handling component initialization and updates.
- Function-based components do not have their own internal state by default. They receive data via props and focus on presenting the UI based on the given props. Prior to React 16.8, function-based components didn't have a built-in way to manage state or lifecycle methods.

3. Code Complexity:

- Class-based components can handle more complex logic and state management, making them suitable for larger, stateful components.
- Function-based components are simpler and more lightweight, suitable for smaller and presentational components that don't require state or lifecycle methods. They are often preferred for their simplicity and ease of understanding.

4. React Hooks:

Prior to React 16.8, function-based components didn't have access to state or lifecycle methods. Hooks were introduced in React 16.8 as a way to add state and other React features to function components. With hooks, function components can now have their own state and lifecycle management, similar to class components.

React Hooks allow function-based components to use features like useState, useEffect, useContext, and more, enabling more advanced functionality while maintaining the simplicity and conciseness of function components.

Q.6 Explain react component life cycle

The React component lifecycle refers to a series of methods that are invoked at different stages of a component's existence, from its creation to its removal from the DOM. Understanding the component lifecycle is crucial for managing component initialization, state updates, and cleanup operations. In React versions before 16.3, the lifecycle was primarily based on class components. With the introduction of React Hooks in React 16.8, function components can also utilize lifecycle-like functionality. Here is an overview of the different phases of the React component lifecycle for class components:

Mounting:

constructor(): This is the first method called during the creation of a component. It is used to initialize state and bind event handlers.

static getDerivedStateFromProps(): This method is invoked before rendering and allows a component to update its internal state based on changes in props.

render(): This method is responsible for returning the JSX representation of the component's UI. It is called each time the component updates.

componentDidMount(): This method is called after the component is mounted in the DOM. It is commonly used for making API calls, setting up subscriptions, or initializing third-party libraries.

Updating:

static getDerivedStateFromProps(): Similar to the mounting phase, this method can be used to update the component's state based on changes in props.

shouldComponentUpdate(): This method determines whether the component should re-render or not. It can be used for performance optimization by preventing unnecessary updates.

render(): This method is called to re-render the component with updated props or state.

componentDidUpdate(): This method is invoked after the component has been re-rendered. It is used for handling side effects or performing additional updates.

Unmounting:componentWillUnmount(): This method is called just before the component is removed from the DOM. It is commonly used for cleaning up event listeners, subscriptions, or other resources.Additionally, there are some other lifecycle methods that are less commonly used:

shouldComponentUpdate(): As mentioned earlier, this method can be used to optimize rendering by preventing unnecessary updates.

getSnapshotBeforeUpdate(): This method allows the component to capture some information from the DOM before it is potentially changed. It is rarely used in practice.It's worth noting that with the introduction of React Hooks, function components have alternative lifecycle methods:

useEffect(): This hook combines functionalities of componentDidMount, componentDidUpdate, and componentWillUnmount. It allows you to perform side effects in a function component.

useLayoutEffect(): Similar to useEffect(), but it fires synchronously after all DOM mutations. It can be used for tasks that require immediate DOM measurements or mutations.

Understanding the React component lifecycle is crucial for managing component behavior and performing actions at the appropriate stages of a component's existence.

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

Prop drilling in React refers to the process of passing down props through multiple intermediate components that don't actually use or need those props. It can occur when a component needs to pass data to a deeply nested child component, but it has to pass through several levels of components in between. This can lead to code complexity and reduced maintainability.

Here's an example to illustrate prop drilling:

```
function ParentComponent() {
  const data = " Prop Drilling!";
  return (
    <div>
      <IntermediateComponent data={data} />
    </div>
  );}function IntermediateComponent({ data }) {
  return <ChildComponent data={data} />;
}function ChildComponent({ data }) {
  return <GrandchildComponent data={data} />;
}function GrandchildComponent({ data }) {
  return <div>{data}</div>;
}
```

- Use React Context: React Context provides a way to share data between components without explicitly passing props through each level. It allows you to define a "context" that can be accessed by any component in its descendant tree. Components can consume the context using the useContext hook or the Context.Consumer component. This helps to avoid the need for prop drilling in certain scenarios.
- Component Composition: Instead of passing props through intermediate components, you can create a wrapper component that encapsulates the hierarchy of components that need the prop. This wrapper component can then handle passing the prop down directly to the relevant child component, without involving the intermediate components.
- State Management Libraries: Utilize state management libraries like Redux or MobX. These libraries centralize the application's state and provide a way to access the state from any component without the need for prop drilling.

- React Hooks: With React Hooks, specifically the useReducer hook, you can manage state in a centralized manner and make it accessible to components that need it. This eliminates the need for prop drilling to pass down state through multiple levels of components.
- Higher-Order Components (HOCs): HOCs are functions that take a component and return a new component with additional props. By wrapping components with HOCs, you can inject the required props without the need for explicit prop drilling.