Q.1 Whats React and its pros and cons?
React is a popular JavaScript library used for building user interfaces. It allows developers to create reusable UI components and efficiently update and render the user interface based on changes in data. React follows a component-based architecture, where the UI is divided into small, reusable components that manage their own state and can be composed together to create complex UI structures.

Pros of React:
1. **Virtual DOM**: React uses a virtual DOM, which is an in-memory representation of the actual DOM. This allows React to efficiently update and render only the necessary parts of the UI when the underlying data changes, resulting in improved performance.
2. **Reusability**: React encourages component reusability, which helps in writing modular and maintainable code. Components can be easily composed together to build complex UI structures.
3. **Unidirectional Data Flow**: React follows a unidirectional data flow, where data flows in a single direction from parent components to child components. This makes the application easier to understand and debug.
4. **React Native**: React can be used to build native mobile applications using React Native. This allows developers to write code once and deploy it on multiple platforms, saving development time and effort.
5. **Active Community**: React has a large and active community of developers, which means there are plenty of resources, tutorials, and libraries available to support React development.

Cons of React:
1. **Learning Curve**: React has a learning curve, especially for developers who are new to JavaScript frameworks or component-based architectures.
2. **Tooling Complexity**: React ecosystem has various tools and libraries associated with it, such as webpack, Babel, and ESLint. Setting up and configuring these tools correctly can be challenging for beginners.
3. **JSX**: React uses JSX (JavaScript XML) syntax, which combines JavaScript and HTML-like syntax. While JSX is powerful and expressive, it may require developers to learn a new syntax and tooling setup.

Q.2 What do you understand by Virtual Dom?
The Virtual DOM (Virtual Document Object Model) is a concept used by React to optimize the updating and rendering of the user interface. It is a lightweight copy or representation of the actual DOM that React maintains in memory.

When the state or props of a component change, React creates a new virtual DOM tree by comparing it with the previous virtual DOM tree. React then efficiently updates only the differences between the two virtual DOM trees in the actual DOM. This process is known as reconciliation.

By using the virtual DOM, React minimizes direct manipulation of the real DOM, which can be slow and inefficient. The virtual DOM allows React to batch multiple DOM updates together and perform them in a more optimized manner, resulting in improved performance and a smoother user experience.

Q.3 Difference between Virtual Dom vs Real Dom
 The main difference between the Virtual DOM and the Real DOM is the way they handle updates and rendering.

Real DOM:
- The Real DOM is the actual representation of the HTML structure of a web page.
- Any change in the state or props of a component triggers a re-rendering of the entire DOM tree.
- Re-rendering the entire DOM tree can be slow and resource-intensive, especially for complex applications with frequent updates.
- Direct manipulation of the Real DOM can lead to performance bottlenecks.

Virtual DOM:
- The Virtual DOM is an in-memory representation of the Real DOM.
- When a component's state or props change, React updates the Virtual DOM instead of the Real DOM.
- React compares the previous Virtual DOM with the updated Virtual DOM and calculates the minimal set of changes needed to update the Real DOM.
- Only the necessary changes are applied to the Real DOM, resulting in better performance compared to re-rendering the entire DOM tree.
- React's diffing algorithm optimizes the update process by minimizing the number of operations required to synchronize the Virtual DOM with the Real DOM.

In summary, the Virtual DOM provides a more efficient way of updating and rendering the user interface by minimizing direct manipulation of the Real DOM and optimizing the update process.

Q.4 Whats component? Types of component
In React, a component is a reusable, self-contained building block for creating user interfaces. Components can be thought of as JavaScript functions or classes that return a piece of UI based on their input, known as props (short for properties).

There are two types of components in React:

1. **Function Components**: Function components are JavaScript functions that receive props as input and return JSX (JavaScript XML) as output. They are also known as stateless functional components because they don't have their own internal state. Here's an example of a function component:

```jsx
```

```
function MyComponent(props) {
  return <div>{props.message}</div>;
}
```

2. **Class Components**: Class components are JavaScript classes that extend the `React.Component` base class. They have their own internal state and can also receive props. Class components use the `render()` method to return JSX. Here's an example of a class component:

```jsx
class MyComponent extends React.Component {
  render() {
    return <div>{this.props.message}</div>;
  }
}
```

Both function components and class components can have child components and can be composed together to form a tree-like structure, representing the UI hierarchy of the application.

Q.5 Difference between class & function based component
The main difference between class-based components and function-based components in React is the syntax and the way they handle state and lifecycle methods.

Class-based components:
- Are JavaScript classes that extend the `React.Component` base class.
- Use the `render()` method to return JSX.
- Have their own internal state, which can be managed using `this.state`.
- Can use lifecycle methods like `componentDidMount()`, `componentDidUpdate()`, and `componentWillUnmount()` for managing side effects and handling component lifecycle events.
- Allow access to `this.props` and `this.state` within the component's methods.
- Can have more complex logic and are suitable for managing stateful components.

Function-based components:
- Are JavaScript functions that receive props as input and return JSX.
- Don't have their own internal state.
- Don't use lifecycle methods directly, but can use React Hooks (introduced in React 16.8) to manage side effects and state.
- Use the `useState` and other hooks provided by React to manage state within the function component.
- Don't use `this` keyword, as there is no instance associated with function components.
- Have a simpler syntax and are recommended for most use cases, especially for simple components that don't require state management or lifecycle methods.

In general, function-based components are considered the modern and recommended approach in React because they promote simpler and more concise code. However, class-based components are still supported and useful in certain situations, such as when working with older codebases or when more advanced component features are needed.

Q.6 Explain react component life cycle
React component lifecycle refers to the series of methods that are invoked at different stages of a component's existence. These methods allow you to perform actions and manage the component's behavior during its lifecycle.

The lifecycle of a class component can be divided into three main phases:

1. **Mounting**: This phase occurs when an instance of a component is being created and inserted into the DOM. The following lifecycle methods are called in the mounting phase:
   - `constructor()`: Used for initializing the component's state and binding event handlers.
   - `static getDerivedStateFromProps()`: Used for updating the component's state based on changes in props.
   - `render()`: Returns the JSX representation of the component.
   - `component

DidMount()`: Invoked after the component is mounted in the DOM. It's commonly used for performing side effects like fetching data from an API or setting up event listeners.

2. **Updating**: This phase occurs when a component's state or props change. The following lifecycle methods are called in the updating phase:
   - `static getDerivedStateFromProps()`: Similar to the mounting phase, it's called when the component receives new props.
   - `shouldComponentUpdate()`: Allows you to optimize component rendering by determining if a re-render is necessary.
   - `render()`: Updates the JSX representation of the component.
   - `componentDidUpdate()`: Invoked after the component is updated in the DOM. It's used for performing side effects based on the updated state or props.

3. **Unmounting**: This phase occurs when a component is being removed from the DOM. The following lifecycle method is called in the unmounting phase:
   - `componentWillUnmount()`: Invoked just before the component is unmounted and destroyed. It's used for cleaning up resources like canceling timers or removing event listeners.

It's important to note that with the introduction of React Hooks, there is an alternative and recommended approach to managing component lifecycle using hooks like `useEffect()`. Hooks provide a more intuitive way of handling side effects and managing component lifecycle in function-based components.

Q.7 Explain Prop Drilling in React & Ways to avoid it
Prop drilling in React refers to the process of passing props down through multiple layers of components, even if some intermediate components don't actually need those props. This can lead to code clutter and make it harder to maintain and refactor the codebase.

To avoid prop drilling and make the code more maintainable, there are a few approaches you can take:

1. **Context API**: React's Context API allows you to share data across components without explicitly passing props through each level. You can create a context using `React.createContext()` and wrap the relevant components with the `Context.Provider` component to make the data available to the consuming components. This way, you can access the context value directly without the need for prop drilling.

2. **Redux**: Redux is a state management library that provides a global store for managing application state. Instead of passing props through multiple components, you can store the shared data in the Redux store and access it from any component in the application. Redux eliminates the need for prop drilling and provides a centralized way of managing shared data.

3. **React Router**: If the prop drilling issue is related to routing-related props, you can use React Router. React Router allows you to define routes and pass route parameters to components without explicitly drilling the props. The components can access the route parameters using hooks or higher-order components provided by React Router.

4. **Component Composition**: Instead of passing props through multiple layers, you can break down your UI into smaller, reusable components and compose them together. This way, only the necessary props are passed to the immediate child components, reducing the amount of prop drilling.