React useMemo Hook - Complete Notes

1️⃣ What is useMemo?

Definition: `useMemo` is a React hook that memoizes a computed value.

- Avoids expensive recalculations on every render by caching the result until dependencies change.
- Useful for performance-critical components.

Syntax:

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

- Returns the memoized value - Recomputes only when dependencies change

---

2️⃣ Why useMemo Exists

- React components re-render frequently
- Any computation inside the component runs on every render, even if inputs didn't change
- `useMemo` caches the result to skip unnecessary recalculation

Example:

```
function Component({ num }) {
  const fib = fibonacci(num); // recalculated on every render ❌
  return <div>{fib}</div>;
}
```

✅Using `useMemo`:

```
const fib = useMemo(() => fibonacci(num),
[num]); // recomputed only when num changes
```

3️⃣ Internal Working

- React keeps a fiber node cache of `useMemo` hooks
- On render:
- React compares the dependencies array with previous render
- If no dependency changed → returns cached value
- If any dependency changed → recomputes and stores new value

- Key: `useMemo` is lazy and dependency-based

4️⃣ When to Use useMemo

- **Expensive calculations:**

```
const sortedUsers = useMemo(() => users.sort((a, b) => a.age - b.age),
[users]);
```

- **Derived state that's costly to compute:**

```
const filteredItems = useMemo(() => items.filter(i => i.active),
[items]);
```

- **Prevent recalculation for stable references:**

```
const options = useMemo(() => computeOptions(config), [config]);
<Dropdown options={options} />
```

5️⃣ Gotchas

- Do **not** overuse: memoization has memory and CPU cost
- Only worth it for expensive calculations or large arrays/objects
- Dependencies must be correct:

```
const value = useMemo(() => compute(a, b), [a]); // ❌ missing b → may
produce stale value
```

- `useMemo` does **not** prevent re-renders; use React.memo with memoized props for that

---

6️⃣ Difference Between useMemo and useCallback

Feature	useMemo	useCallback
Returns	memoized value	memoized function
Use case	expensive calculation	stable function reference
Dependencies	[dep1, dep2]	[dep1, dep2]

Feature	useMemo	useCallback
Optimization	Avoid recalculating values	Avoid recreating functions for children props

Example useCallback:

```
const handleClick = useCallback(() => doSomething(count), [count]);
```

7️⃣ Common Production Patterns

1️⃣ Expensive list computations:

```
const visibleTodos = useMemo(() => {
  return todos.filter(todo => todo.completed === false);
}, [todos]);
```

2️⃣ Stable object / array references:

```
const tableColumns = useMemo(() => [
  { Header: "Name", accessor: "name" },
  { Header: "Age", accessor: "age" }
], []); // stable reference → prevents child re-renders
```

3️⃣ Combination with React.memo:

```
const memoizedColumns = useMemo(() => columns, [columns]);
<Table columns={memoizedColumns} data={data} />; // avoids unnecessary render
```

4️⃣ Derived calculations in dashboards / analytics:

```
const totalSales = useMemo(() => sales.reduce((sum, s) => sum + s.amount,
0), [sales]);
```

8️⃣ Senior-Level Best Practices

- Only memoize expensive computations → don't use for trivial calculations
- Always provide correct dependencies → avoid stale values
- Use with React.memo for child component optimization
- Avoid unnecessary nesting → memoize only high-cost operations

• Understand reference equality → memoized object/array ensures stable prop references

---

9️⃣ Example — Full Pattern

```
function Dashboard({ salesData, filter }) {
  // Expensive filtered calculation
  const filteredSales = useMemo(() => {
    console.log("Filtering sales..."); // runs only when salesData or filter
change
    return salesData.filter(s => s.region === filter.region);
  }, [salesData, filter]);

  // Stable object reference for child props
  const tableColumns = useMemo(() => [
    { Header: "Product", accessor: "product" },
    { Header: "Revenue", accessor: "revenue" }
  ], []);

  return <Table data={filteredSales} columns={tableColumns} />;
}
```

🔟 Senior-Level Mental Model

- `useMemo` = cache value between renders
- Dependencies = “when to recompute”
- Does **not** prevent rendering, only recomputations
- Works best with expensive calculations or stable prop references
- Combine with React.memo → full render optimization

✅ **Key Takeaways:** - Memoize values, not functions (useCallback for functions) - Only use for performance-critical computations - Always specify dependencies - Great for large lists, derived state, stable props, dashboards, analytics