**Redux**

**Part 1:**

1. Can you provide a brief summary of what is happening in this function code?

The countReducer function is a reducer function for managing the state of a counter.

It takes two parameters: state (representing the current state) and action (representing the action dispatched).

If the action.type is 'increment', it returns a new state object with the value incremented by 1.

2. Add one action that tells the reducer to reduce the state value by 1

function countReducer(state = initialState, action) {

    if (action.type === 'increment') {

      return {

        value: state.value + 1

      };

    } else if (action.type === 'reduce') {

      return {

        value: state.value - 1

      };

    }

  }

3. Add one action that tells the reducer to reset the state

function countReducer(state = initialState, action) {

    if (action.type === 'increment') {

      return {

        value: state.value + 1

      };

    } else if (action.type === 'reduce') {

      return {

        value: state.value - 1

      };

    } else if (action.type === 'reset') {

      return {

        value: initialState.value

      };

    }

}

**Part 2:**

1. Can you provide a brief summary on what is happening on line 34, 39?

**Line 34**:

This line uses the useState hook to initialize a state variable studentsCount with an initial value of 0. setStudentsCount is the function that will be used to update this state.

**Line 39**:

This line creates a button with an onClick event handler. When the button is clicked, it invokes the setStudentsCount function to update the studentsCount state by incrementing its value by 1.

2. When a user clicks on the “Add student” button update the state (studentsCount) to

include only the total number of students who are present. Using the data provided

below:a.

// handling the "Add Student" button click

function handleAddStudent() {

    // Filter students

    const presentStudents = students.filter((student) => student.present);

    // Get the count of present students

    const presentStudentsCount = presentStudents.length;

    // Update the state with the count of present students

    setStudentsCount(presentStudentsCount);

  }

b.

Attach the handleAddStudent function to the onClick event of the "Add Student" button.

Here is an example:

<button onClick={handleAddStudent}>Add Student</button>

c.

Use the setStudentsCount function to update the studentsCount state with the count of present students (presentStudentsCount).

Here is an example (as in ‘a’):

setStudentsCount(presentStudentsCount);

**Part 3:**

1. A change of code was made on line 174 (figure 4), can you briefly explain what that

would do?

The code change on line 174 introduces an action payload (action.payload) to the "increment" action. This payload is added to the current state value when updating the state. It allows dynamic incrementing by a specified value.

2. Let’s say we don’t want to set the state locally anymore and want to use dispatch.

How would you ensure that an “increment” action that also contains the result of the

studentCount is dispatched on button click? According to your answer in part 2.2b

what would need to be changed？

Dispatch an action of type "increment" with the payload being the count of present students.

Like: dispatch({ type: 'increment', payload: presentStudentsCount });

And the yellow marked line ‘dispatch’ should be done inside the handleAddStudent function when the button is clicked.

3. Which code do you think is best suited to ensure that the “increment” action updates

the state with the correct total number of students who are present. Is it Figure 4 code? Or

Figure 5 code? Explain the code difference and your reasoning？

Both Figure 4 and Figure 5 codes can be used for updating the state based on the "increment" action. The difference lies in how they handle the state update. Figure 4 increments the current state value by the payload, while Figure 5 sets the state value directly to the payload.

If the requirement is accumulating the count of present students over multiple button clicks, Figure 4 is more suitable. If need to set the state value to the latest count of present students without accumulation, Figure 5 is more suitable.

**Algorithm 1:**

what is its Big O Time & Space complexity? Why have you chosen this approach? Could there be a more efficient way (and if so, how)

Time complexity: O(N); Space complexity: O(1)

The reason for this is that each character in the string is examined once, and the while loop iterates through the string from both ends, effectively traversing the entire string once.

Why this approach:

The two-pointer approach is chosen because it allows for linear traversal of the string without the need for additional data structures.

The algorithm efficiently skips non-alphanumeric characters, making it suitable for handling strings with punctuation, spaces, and other non-alphanumeric characters.

Could there be a more efficient way: I can think of any better efficient way to solve it for now.

**Algorithm 2:**

what is its Big O Time & Space complexity? Why have you chosen this approach? Could there be a more efficient way (and if so, how)

Time complexity: O(N); Space complexity: O(1)

Why this approach:

The algorithm efficiently identifies non-numeric values and negative numbers in the array, providing meaningful error messages for such cases. It iterates through the array only once for each check, keeping the time complexity linear with respect to the array size.

Could there be a more efficient way: I can think of any better efficient way to solve it for now.