DOAnswers

PART 1

1. This is a function that tells the reducer to change the state by increasing the value associated with it. So if the value is =5, it will be increasing to 6, and if that's not case the state is returned with no alterations to it

2. // else if (action.type === “decrement”) {
   // return {
   // value: state.value -1
   }
   }
   return state
   }

3. // case RESET:
   // return initialState;
   // default:
   // return state;

PART 2

1. In line 34, 39, use State hook is being utilised to create a state named “studentsCount” with an initial value of 0. There is also a setStudentsCount function to update the aforementioned state. Inside the components’s JSX, the current student count is being displayed using the studentsCount state variable. The button element with the text “Add Student”, exists to increment the student count when clicked.

2a.
```
import React, { useState } from 'react';

function Classroom() {
const students = [
 // { name: “Nrupul”, present: false},
 // { name: "Prateek”, present: true},
 // { name: “Jane”, present: true },
//{ name: “Paul”, present: false},
//{ name: “Jane”, present: true}
];

// const [studentsCount, setStudentsCount] = useState(
  students.filter(student => student.present).length
);

//const addStudent = () => {
  setStudentsCount(studentsCount + 1);
};


export default Classroom;
```

b.
```
//   //return (
    <div>
      <p>Number of students in classroom: {studentsCount}</p>
      <button onClick={addStudent}>Add Student</button>
    </div>
  );
}
```

c.
```
const calculatePresentStudentCount = () => {
 return students.filter(student => student.present).length
  );
// const [studentsCount, setStudentsCount] = useState(calculatedCount);
```

PART 3

1. The updated code checks if the action type is "increment", and if it is, the state's value property is updated by adding the value provided in the action.payload. However, if the action type is not "increment", it returns the current state without making any changes.

2. Through “redux state management”. This involves dispatching actions to update the state stored in the Redux store. The Redux store handles the state updates.

```
// // reducer.js
import { INCREMENT_STUDENT_COUNT } from './actions';

//const initialState = {
  studentsCount: 0,
};

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case INCREMENT_STUDENT_COUNT:
      return { ...state, studentsCount: action.payload };
    default:
      return state;
  }
};

export default reducer;
```

3. Figure 4 because I understand what it’s doing. I am familiar with that syntax. I know when the program runs, because the action type is already an increment and the state value is provided, the value property would be updated. There is no state value specified in figure 5 so there is a high probability that code would not produce the output that’s needed

ALGORITHM 1:

The provided solution converts the input string to lowercase and removes spaces to make the comparison case-insensitive and space-agnostic. Then, it compares the modified string with its reverse to determine if it's a palindrome.

Time Complexity:

- Converting the string to lowercase and removing spaces takes O(n), where n is the length of the input string.
- Splitting the string into an array, reversing it, and joining it back into a string also takes O(n).
- Overall, the time complexity is O(n).

Space Complexity:

- The space complexity depends on the space used by the modified string and the intermediate array used during the array operations.
- The modified string space complexity is O(n) since it's the length of the input string.
- The intermediate array used for splitting the string and reversing it also contributes to O(n) space complexity.
- Overall, the space complexity is O(n).

Approach Rationale:

- The approach of converting to lowercase and removing spaces ensures that the comparison is case-insensitive and ignores spaces.
- Reversing the string with array manipulation provides a simple way to compare the reversed string with the original.

Possible Improvements:

- The provided solution is straightforward and efficient for most cases.
- However, an alternative approach could involve comparing characters directly from the beginning and end of the string, skipping non-alphanumeric characters.
- This approach would require two pointers that move towards each other until they meet, skipping non-alphanumeric characters.

Comparison with Alternative:

- The alternative approach of comparing characters directly would have a similar time complexity of O(n), where n is the length of the string.
- The space complexity would be lower since it doesn't involve creating additional modified strings or arrays.
- This approach could potentially be more efficient in terms of space but might require more complex code.
  Calculate Actual Sum: Calculate the sum of the elements in the array.
  Calculate Missing Number: The missing number is the difference between the expected sum and the actual sum.
  Return Result: If the missing number is not negative or greater than `n`, return the missing number. If the missing number is negative or greater than `n`, return an appropriate error message.

In conclusion, the chosen approach is efficient and straightforward for solving this problem while also handling various input scenarios. It has a good balance between time and space complexity. While there might be more specialised algorithms, they could be more complex without significant improvements in performance for typical input sizes.

ALGORITHM 2:

Complexity Analysis:

Time Complexity: The algorithm iterates through the input array twice: once to populate the seenNumbers set and once to find the missing number. Therefore, the time complexity is O(n), where n is the length of the input array.

Space Complexity: The primary space usage comes from the seenNumbers set, which can store all the unique numbers in the input array. In the worst case, the set might contain all n numbers. Thus, the space complexity is O(n). Additionally, the algorithm uses a constant amount of extra space for variables and temporary storage, which doesn't contribute significantly to the space complexity.

.