Task 1

My approach to task one was efficiency, in both the number of variables created and the speed in which the code executed. To achieve this, I made sure to use a for loop that only executed once, so the code would run in O(n) time. Additionally, I created only one variable, the new reversed string, and returned it directly. I chose to use a for loop instead of a while because I feel it is clearer in conveying what the code does. A for loop shows that the loop will activate until the full length of the word is done. Furthermore, because the string parameter is stored as an array which starts at zero, I need to have *i* be the length of the parameter minus 1, or else the code will end on an undefined. For example, instead of doing (hello) -> (olleh), it would be (hello) -> (undefinedolleh), which I want to avoid.

Task 2

For the task two problem, there are many ways to solve it. Initially, I used a series of if/else statements to print the correct output depending on whether the number was divisible by 3 or 5. However, I had problems with this approach. For one thing, I would have to repeat the same if statement at least twice for either divisible by three or 5. For example, I would have had:

```
…
if (i % 3 == 0) {
        if (i % 5 == 0) {
                console.log(“FizzBuzz)
        }
        console.log(“Fizz)
}
else if (i % 5 == 0) {
        console.log(“Buzz)
}
Else {
        console.log(i)
}
```

I must repeat the I % 5 == 0 check because I have to check if the number is divisible by 5 both after confirming 3 is divisible and after confirming 3 was not divisible. This is a repeat process, so it is a waste of time and confusing. I decided a better way of visualizing the problem was using a series of Booleans to represent the state of each component of fizzbuzz. A Boolean to represent divisible by 3, divisible by 5, and divisible by both. By using Booleans to represent states of fizzbuzz, I reduce the number of if statements needed to confirm whether a number is divisible down to three.

After this, I chose to use a switch statement to decide which case should be printed. Initially, I ran into trouble with having fizzbuzz never appearing in the results. I solved this problem by observing the order of the cases, and realizing I had the fizzbuzz case as my final case. Since for fizzbuzz to activate, both fizz and buzz are active, whenever I had the fizzbuzz case I would instead activate fizz, because it was first in the case order, and thus selected by the switch before reaching the correct fizzbuzz case. This can be seen in the chart below:

1. Is the number divisible by 3?
    - 1.1. Yes! Print Fizz <- By succeeding here, it is impossible to reach fizzbuzz since fizzbuzz will also succeed on number divisible by 3
    - 1.2. No! The number must be divisible by either 5 or neither <- By failing here, impossible to reach fizzbuzz
2. Is the number divisible by 5?
    - 2.1. Yes! Print Fizz
    - 2.2. No! The number is divisible by 5 OR not at all
3. Is the number divisible by 3 and 5?
    - 3.1. Yes! Print Buzz
    - 3.2. No! Go to default case

To solve this problem, I made fizzbuzz the first case that the switch checks. This causes the switch statement to check like so:

1. Is the number divisible by both 3 and 5?
    - a. Yes! Print Fizzbuzz
    - b. No! The number must be divisible by either 3 OR 5 (but not both), or neither
2. Is the number divisible by 3?
    - a. Yes! Print Fizz
    - b. No! The number is divisible by 5 OR not by 5
3. Is the number divisible by 5?
    - a. Yes! Print Buzz
    - b. No! Go to default case

I used a switch statement because I feel it is more readable than a series of if/else statements, while additionally allowing for a lower number of if/else statements than what I would have needed if I did not use a switch statement. While the creation of Booleans does cost a small amount of extra memory, because they are created outside the scope of the for loop, I am only creating three extra variables. Three extra variables will have a negligible impact on memory use and runtime speed. As a result, while being somewhat lengthier than a series of nested if statements, I feel my solution is equivalently efficient and much more readable.

Task 3

Task three is simple, but there are ways to maintain efficiency. The easiest is to directly return the “Even” or “Odd” string from the if statement, instead of assigning the string to a variable and returning that, which is something I have seen my students do many times when solving this problem. By returning the direct result and not creating a new variable, memory is saved.

Task 4

Task four required using JavaScript to change the text of a header in html by pressing a button. I decided the best way to do this was by changing the color and text of the header. In order to change the color, I created a css class for the text color, which I would then assign to the header on the button press event. I felt assigning a css class instead of directly changing the color was a better coding practice, since it decouples the color from the JavaScript. For example, if I wanted the button press to change the color to orange instead of red, I would not need to go into the JavaScript and change that code. Instead, I could just change the associated css class, and now all javascript functions that use the new class will automatically change, without having to manually change every Javascript onEvent function. Additionally, styling such as color falls under the purview of the css style sheet, so it is more readable to place it there.

One struggle I had was selecting the h1 content, to then change its color and text. Initially, I tried selecting by class=”black”, but this caused issues when then immediately setting the class to something else. As a result, I gave the header its own unique id, and then searched that in the script.js to implement changes. This worked perfectly, and allowed me to change the class of the associated text with id=”heading_span”, along with its text content. To change the text content, I just reset it by using.textContent = “Insert new text here” in script.js.