STEP 1:

The gas optimization techniques applied to the `contracts/gasChallenge.sol` file

```
uint[10] numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

In the code above, the **numbers** array was defined as a dynamic array **uint[10].** In the original code, the **numbers** array was defined as a dynamic array **uint[].** To utilize the fixed-size array technique, we changed it to a fixed-size array **uint[10].** This restricts the array to have a fixed length of 10 elements.

```
function optimizedFunction() public {
        unchecked {
            uint len = numbers.length;
            for (uint i = 0; i < len; ++i) {
                numbers[i] = 0;
            }
        }
    }
```

In the code above, the **optimizedFunction()** function was completed.

The **unchecked** block in the **optimizedFunction()** was use to skip certain checks, such as integer overflows. This can help reduce gas consumption by omitting unnecessary checks in the loop.

The **for** loop increment syntax was changed from **i++** (postfix increment) to **++i** (prefix increment). This alternative syntax does the same operation but in a slightly more optimized way, as it eliminates the need for a duplicate operation for incrementing the loop variable, resulting in reduced gas consumption during execution.

STEP 2:

The unit test under the describe block to check that after running the gas optimized function, the sum of array is 0.

```
// Write test block here to check sum of array equals 0
      await gas_contract.optimizedFunction();
      const sum = await gas_contract.getSumOfArray();
      expect(sum).to.equal(0);
```

The code above was written under the **describe** block.

In the "Check Sum Of Array" test block, the **optimizedFunction()** was called first to set all elements of the **numbers** array to 0. Then, the **getSumOfArray()** function was used to retrieve the sum of the array and store it in the **sum** variable. Finally, the **expect** statement from the Chai assertion library was used to assert that the **sum** variable is equal to 0, verifying that the optimized function correctly sets the array elements to 0, and the sum becomes 0 as expected.