1. **A word is encrypted using below fashion Decrypted word: gamer Encrypted word: rmgae Given such a encrypted word. task is to print it's encrypted form**

Method:1

```
function encryptWord(word) {
    // Reverse the word
    let reversedWord = word.split('').reverse().join('');

    // Move the first character of the reversed word to the end
    let encryptedWord = reversedWord.slice(1) + reversedWord[0];

    return encryptedWord;
}

// Example usage
let decryptedWord = 'gamer';
let encryptedWord = encryptWord(decryptedWord);
console.log(encryptedWord);  // Output: rmgae
```

Method:2

```
function encryptWord(word) {
    // Step 1: Reverse the word
    let reversedWord = '';
    for (let i = word.length - 1; i >= 0; i--) {
        reversedWord += word[i];
    }

    // Step 2: Move the first character of the reversed word to the end
    let encryptedWord = '';
    for (let i = 0; i < reversedWord.length; i++) {
      if(i%2==0){
          encryptedWord += reversedWord[i];
      }
    }
     for (let i =reversedWord.length-1 ; i > 0; i--) {
      if(i%2 != 0){
          encryptedWord += reversedWord[i];
      }
    }

    return encryptedWord;
```

```
}

// Example usage
let decryptedWord = 'gamer';
let encryptedWord = encryptWord(decryptedWord);
console.log(encryptedWord);  // Output: rmgae
```

Time Complexity: O(n)
Space Complexity: O(n)

2.Given an integer array. Print the numbers which are powers of 4.

```
function isPowerOfFour(num) {
    if (num <= 0) {
        return false;
    }
    while (num % 4 === 0) {
        num /= 4;
    }
    return num === 1;
}

function printPowersOfFour(arr) {
    arr.forEach(num => {
        if (isPowerOfFour(num)) {
            console.log(num);
        }
    });
}

// Example usage:
const arr = [1, 4, 16, 20, 64, 100, 256];
printPowersOfFour(arr);
```

Time Complexity: O(m · log(n))
Space Complexity: O(1)

3.Rearrange letters of the word such that the resultant word is not a palindrome.

```
function isPalindrome(word) {
    const len = word.length;
    for (let i = 0; i < len / 2; i++) {
```

```
        if (word[i] !== word[len - 1 - i]) {
            return false;
        }
    }
    return true;
}

function rearrangeWord(word) {
    if (!isPalindrome(word)) {
        return word;
    }

    let chars = word.split('');

    // Swap the first two characters if the word is a palindrome
    if (chars.length > 1) {
        let temp = chars[0];
        chars[0] = chars[1];
        chars[1] = temp;
    }

    // If the resultant word is still a palindrome (like in case of words with all
same characters)
    // Swap any two adjacent characters which are different
    if (isPalindrome(chars.join(''))) {
        for (let i = 1; i < chars.length; i++) {
            if (chars[i] !== chars[0]) {
                let temp = chars[0];
                chars[0] = chars[i];
                chars[i] = temp;
                break;
            }
        }
    }

    return chars.join('');
}

// Example usage:
const word = "aabbaa";
const result = rearrangeWord(word);
console.log(result);
```

space complexity and time complexity: **O(n)**

4. .Write a function that prints out a breakdown of an integer into a sum of numbers that have just one non-zero digit which is at the first position. Input: 12345, Output: Print 10000 + 2000 + 300 + 40 + 5 Input: 43018, Output: Print 40000 + 3000 + 10 + 8
123 - 100 + 20

```
function breakdownInteger(num) {
    let digits = num.toString().split('');
    let result = [];

    for (let i = 0; i < digits.length; i++) {
        if (digits[i] !== '0') {
            let power = digits.length - 1 - i;
            let value = parseInt(digits[i]) * Math.pow(10, power);
            result.push(value);
        }
    }

    console.log(result.join(' + '));
}

// Examples:
breakdownInteger(12345); // Output: 10000 + 2000 + 300 + 40 + 5
breakdownInteger(43018); // Output: 40000 + 3000 + 10 + 8
```

Time Complexity: O(d), where d is the number of digits in the input number `num`.
Space Complexity: O(d), where d is the number of digits in the input number `num`.

5.first non repeating character in string

```
function firstNonRepeatingCharacter(str) {
    // Step 1: Create an array to store frequency counts of each character
    const charCount = new Array(26).fill(0); // Assuming only lowercase English letters

    // Step 2: Count frequencies of each character
    for (let i = 0; i < str.length; i++) {
        let index = str.charCodeAt(i) - 'a'.charCodeAt(0);
        charCount[index]++;
```

```
    }

    // Step 3: Find the first non-repeating character
    for (let i = 0; i < str.length; i++) {
        let index = str.charCodeAt(i) - 'a'.charCodeAt(0);
        if (charCount[index] === 1) {
            return str[i];
        }
    }

    // Step 4: If no non-repeating character found, return null or handle
accordingly
    return null;
}

// Example usage:
const str1 = "leetcode";
const str2 = "loveleetcode";
console.log(firstNonRepeatingCharacter(str1)); // Output: "l"
console.log(firstNonRepeatingCharacter(str2)); // Output: "v"
```

Time Complexity: O(n)
Space Complexity: O(1)

6.ind the 2 indexes where sum is given target (Two Sum)

7.String anagrams

```
function areAnagrams(str1, str2) {
    // Check if the lengths of the strings are different
    if (str1.length !== str2.length) {
        return false;
    }

    // Create frequency counters for both strings
    const charCount1 = {};
    const charCount2 = {};

    // Populate frequency counters
    for (let char of str1) {
        charCount1[char] = (charCount1[char] || 0) + 1;
    }
```

```
    for (let char of str2) {
        charCount2[char] = (charCount2[char] || 0) + 1;
    }

    // Compare frequency counters
    for (let key in charCount1) {
        if (charCount1[key] !== charCount2[key]) {
            return false;
        }
    }

    // If all characters and their frequencies match, they are anagrams
    return true;
}

// Example usage:
const str1 = "listen";
const str2 = "silent";
console.log(areAnagrams(str1, str2)); // Output: true

const str3 = "hello";
const str4 = "world";
console.log(areAnagrams(str3, str4)); // Output: false
```

Time: O(n)
Space: O(1)

8.find the maximum occurrences of elements in two arrays and their intersection

```
function maxOccurrences(arr1, arr2) {
    // Step 1: Create frequency maps for both arrays
    const freqMap1 = createFrequencyMap(arr1);
    const freqMap2 = createFrequencyMap(arr2);

    // Step 2: Find maximum occurrences in the first array
    let maxCount1 = 0;
    let maxElement1 = null;
```

```
    for (let [key, count] of freqMap1.entries()) {
        if (count > maxCount1) {
            maxCount1 = count;
            maxElement1 = key;
        }
    }

    // Step 3: Find maximum occurrences in the second array
    let maxCount2 = 0;
    let maxElement2 = null;

    for (let [key, count] of freqMap2.entries()) {
        if (count > maxCount2) {
            maxCount2 = count;
            maxElement2 = key;
        }
    }

    // Step 4: Find maximum occurrences in the intersection of both arrays
    let intersection = arr1.filter(value => arr2.includes(value));
    const freqMapIntersection = createFrequencyMap(intersection);

    let maxCountIntersection = 0;
    let maxElementIntersection = null;

    for (let [key, count] of freqMapIntersection.entries()) {
        if (count > maxCountIntersection) {
            maxCountIntersection = count;
            maxElementIntersection = key;
        }
    }

    return {
        maxCount1,
        maxElement1,
        maxCount2,
        maxElement2,
        maxCountIntersection,
        maxElementIntersection
    };
}

function createFrequencyMap(arr) {
    const freqMap = new Map();
```

```
        arr.forEach(item => {
            freqMap.set(item, (freqMap.get(item) || 0) + 1);
        });
        return freqMap;
    }

    // Example usage:
    const arr1 = [1, 2, 3, 2, 2, 3, 4];
    const arr2 = [2, 3, 4, 5, 2, 3, 5, 6];
    const result = maxOccurrences(arr1, arr2);
    console.log(result);
```

Time complexity: Linear (O(n)) Space complexity: Linear (O(n))

9.Binary array (swap 1 to 0)

```
function swapBinaryArray(arr) {
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === 0) {
            arr[i] = 1;
        } else if (arr[i] === 1) {
            arr[i] = 0;
        }
        // If arr[i] is neither 0 nor 1, it remains unchanged
    }
    return arr;
}

// Example usage:
const binaryArray = [1, 0, 1, 1, 0, 0, 1];
console.log("Original array:", binaryArray);
const swappedArray = swapBinaryArray(binaryArray.slice()); // Create a copy to
avoid modifying the original array
console.log("Swapped array:", swappedArray);
```

Time complexity: Linear (O(n)) Space complexity: Linear (O(1))

10.Check whether a year is a Leap year or not. Find the next leap year of n in O(1) time complexity.
Input: n=4

Output: 8

```
function isLeapYear(year) {
    // Leap year condition:
    // 1. Year must be divisible by 4
    // 2. If the year is divisible by 100, it must also be divisible by 400
    return (year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0);
}

function nextLeapYear(n) {
    // Check if n is already a leap year
    if (isLeapYear(n)) {
        return n + 4;
    } else {
        // Calculate the next leap year
        let remainder = n % 4;
        return n + (4 - remainder);
    }
}

// Example usage:
const n = 4;
const nextLeap = nextLeapYear(n);
console.log("Next leap year after", n, "is:", nextLeap);
```

14.Write a function that takes input an integer number and prints the closest prime integer to that number. The closest prime can be greater or smaller than the passed input integer. If there are equi-distant prime-numbers, print both. "32": Closest prime number is "31", so print "31" "30": Closest prime numbers are "29" and “31”, so print both

```
function isPrime(num) {
    // Check if num is less than 2 or equal to 3
    if (num <= 1) {
        return false;
    }
    if (num <= 3) {
        return true;
    }
    // Check if num is divisible by 2 or 3
    if (num % 2 === 0 || num % 3 === 0) {
        return false;
    }
```

```
    // Check for prime numbers in the form of 6k ± 1
    for (let i = 5; i * i <= num; i += 6) {
        if (num % i === 0 || num % (i + 2) === 0) {
            return false;
        }
    }
    return true;
}

function closestPrimes(n) {
    let lowerPrime = null;
    let higherPrime = null;
    let distance = 1;
    let found = false;

    // Check primes in both directions from n
    while (!found) {
        if (isPrime(n - distance) && !lowerPrime) {
            lowerPrime = n - distance;
        }
        if (isPrime(n + distance) && !higherPrime) {
            higherPrime = n + distance;
        }
        if (lowerPrime && higherPrime) {
            found = true;
        }
        distance++;
    }

    // Output results
    if (lowerPrime && higherPrime) {
        console.log(`Closest prime numbers to ${n} are ${lowerPrime} and
${higherPrime}`);
    } else if (lowerPrime) {
        console.log(`Closest prime number to ${n} is ${lowerPrime}`);
    } else if (higherPrime) {
        console.log(`Closest prime number to ${n} is ${higherPrime}`);
    } else {
        console.log(`No prime numbers found.`);
    }
}

// Example usage:
closestPrimes(32); // Output: Closest prime number to 32 is 31
```

```
closestPrimes(30); // Output: Closest prime numbers to 30 are 29 and 31
```

Time complexity: Linear (O(sq. root of n)) Space complexity: Linear (O(1))

15.Write a function that takes input as an array of integers and returns the third largest integer inside the array. Do not SORT the array for solving this problem.
Input: {3, 5, 7, 1, 10, 14, 6}, Output: third

```
function thirdLargest(arr) {
    let first = -Infinity;
    let second = -Infinity;
    let third = -Infinity;

    for (let i = 0; i < arr.length; i++) {
        if (arr[i] > first) {
            third = second;
            second = first;
            first = arr[i];
        } else if (arr[i] > second && arr[i] !== first) {
            third = second;
            second = arr[i];
        } else if (arr[i] > third && arr[i] !== second && arr[i] !== first) {
            third = arr[i];
        }
    }

    return third !== -Infinity ? third : "No third largest element found";
}

// Example usage:
const inputArray = [3, 5, 7, 1, 10, 14, 6];
const result = thirdLargest(inputArray);
console.log("Third largest element:", result); // Output: Third largest element:
10
```

Time complexity: Linear (O(n)) Space complexity: Linear (O(1))

16. Question# Variation of Pell Sequence

The Pell sequence is constructed by adding the second last number of the sequence and twice of the last number of the sequence so far to get the next

number in the sequence. The first and the second numbers of the sequence are defined as 0 and 1. We get:

Write a function which takes input as a number:
If the given number is a Pell sequence number, print the sum of all odd Pell sequence numbers less than or equal to the given number.
If the given number is NOT a Pell sequence number, print the sum of all even Pell sequence numbers less than the given number.

0, 1, 2, 5, 12, 29, 70, 169, 408…

Test cases for candidate

Input: 29 Output: 35
Input: 30 Output: 14

```
function isPellNumber(n) {
    let a = 0, b = 1, c;

    if (n === 0 || n === 1) {
        return true;
    }

    while (true) {
        c = 2 * b + a;
        if (c === n) {
            return true;
        }
        if (c > n) {
            return false;
        }
        a = b;
        b = c;
    }
}

function pellSequenceSum(number) {
    if (isPellNumber(number)) {
        // Sum of all odd Pell sequence numbers <= number
        let sum = 0;
        let a = 0, b = 1, c;

        if (number >= 0) {
            sum += 0; // First number in Pell sequence is 0 (even)
```

```
        }
        if (number >= 1) {
            sum += 1; // Second number in Pell sequence is 1 (odd)
        }

        while (true) {
            c = 2 * b + a;
            if (c > number) {
                break;
            }
            if (c % 2 !== 0) {
                sum += c;
            }
            a = b;
            b = c;
        }

        console.log(`Sum of all odd Pell sequence numbers <= ${number}:
${sum}`);
    } else {
        // Sum of all even Pell sequence numbers < number
        let sum = 0;
        let a = 0, b = 1, c;

        while (true) {
            c = 2 * b + a;
            if (c >= number) {
                break;
            }
            if (c % 2 === 0) {
                sum += c;
            }
            a = b;
            b = c;
        }

        console.log(`Sum of all even Pell sequence numbers < ${number}:
${sum}`);
    }
}

// Test cases
pellSequenceSum(29); // Output: Sum of all odd Pell sequence numbers <=
29: 35
```

```
pellSequenceSum(30); // Output: Sum of all even Pell sequence numbers <
30: 14
```

Time complexity: Linear (O(sq. root of n)) Space complexity: Linear (O(1))

Question#. String compression

Details:
Given a string containing only lowercase alphabets, we have to compress it using character frequencies.

Examples:
INPUT: "bbbbb"
OUTPUT: "b5"
INPUT: "abc"
OUTPUT: "a1b1c1"
INPUT: "amazon"
OUTPUT: "a2m1z1o1n1"

```
function firstNonRepeatingCharacter(str) {
    const charCount = new Array(26).fill(0); // Assuming only lowercase English letters

    for (let i = 0; i < str.length; i++) {
        let index = str.charCodeAt(i) - 'a'.charCodeAt(0);
        charCount[index]++;
    }

    let result = '';

    for (let i = 0; i < str.length; i++) {
        let index = str.charCodeAt(i) - 'a'.charCodeAt(0);
        if (charCount[index] > 0) {
            result += str[i] + charCount[index];
            charCount[index] = 0; // Mark as counted to avoid repetition
        }
    }

    return result;
}
```

```
// Example usage:
const str1 = "abc";
console.log(firstNonRepeatingCharacter(str1)); // Output: "a2m1z1o1n1"
```

Time complexity: Linear (O(n)) Space complexity: Linear (O(1))

Question#. Number of minimum coins of 2s and 5s

Details:
Given an amount, write a program such that the program should return a minimum number of coins as a tender exact exchange against the given amount. The change should be only in the form of coins of 2s and 5s.

Test cases for candidate

Input: 30
Output: 6 coins of 5s

Input: 32
Output: 6 coins of 5s and 1 coin of 2s

Input: 33
Output 5 coins of 5s and 4 coins of 2s

```
function minimumCoins(amount) {
    if (amount < 0) {
        return "Invalid amount";
    }

    let count5 = 0;
    let count2 = 0;

    // First, use as many 5s as possible
    while (amount >= 5) {
        count5++;
        amount -= 5;
    }

    // Then, use 2s for the remaining amount
    while (amount >= 2) {
```

```
        count2++;
        amount -= 2;
    }

    // Check if exact change is possible
    if (amount !== 0) {
        return "Exact change not possible with coins of 2s and 5s";
    }

    let result = "";
    if (count5 > 0) {
        result += `${count5} coin${count5 > 1 ? 's' : ''} of 5s`;
    }
    if (count2 > 0) {
        if (count5 > 0) {
            result += ` and `;
        }
        result += `${count2} coin${count2 > 1 ? 's' : ''} of 2s`;
    }

    return result;
}

// Test cases
console.log(minimumCoins(30)); // Output: "6 coins of 5s"
console.log(minimumCoins(32)); // Output: "6 coins of 5s and 1 coin of 2s"
console.log(minimumCoins(33)); // Output: "5 coins of 5s and 4 coins of 2s"
```

Time complexity: Linear (O(n)) Space complexity: Linear (O(1)

Given a string str, the task is to convert the given string into the number without using any inbuilt function.

```
function convert(s) {
let num = 0;
let n = s.length;

for (let i = 0; i < n; i++) {
        num = num * 10 + (s.charCodeAt(i) - 48);
}
```

```
console.log(num);
}

let s = "583";
convert(s);
```

We are given two sorted arrays. We need to merge these two arrays such that the initial numbers (after complete sorting) are in the first array and the remaining numbers are in the second array

```
let arr1=[1, 5, 9, 10, 15, 20];
  let arr2=[2, 3, 8, 13];

  function merge(m,n)
  {
     for (let i=n-1; i>=0; i--)
     {

        let j, last = arr1[m-1];
        for (j=m-2; j >= 0 && arr1[j] > arr2[i]; j--)
           arr1[j+1] = arr1[j];

        // If there was a greater element
        if (last > arr2[i])
        {
           arr1[j+1] = arr2[i];
           arr2[i] = last;
        }
     }
  }

  merge(arr1.length,arr2.length);
  document.write("After Merging <br>First Array: ");
```

```
for(let i=0;i<arr1.length;i++)
{
   document.write(arr1[i]+" ");
}
document.write("<br>Second Array:  ");
for(let i=0;i<arr2.length;i++)
{
   document.write(arr2[i]+" ");
}
```

Given a number N, The task is to find the length of the longest consecutive 1s series in its binary representation.

```
function maxConsecutiveOnes(x)
{

        let count = 0;
        while (x != 0)
        {
                x = (x & (x << 1));
                count++;
        }
        return count;
}

console.log(maxConsecutiveOnes(14) );
console.log(maxConsecutiveOnes(222));
```

Print prime numbers 1to 100

```
    function isPrime(num) {
    if (num <= 1) return false; // 1 and below are not prime numbers
    if (num <= 3) return true;  // 2 and 3 are prime numbers

    if (num % 2 === 0 || num % 3 === 0) return false;

    for (let i = 5; i * i <= num; i += 6) {
        if (num % i === 0 || num % (i + 2) === 0) return false;
    }
    return true;
}

function printPrimes(from, to) {
    for (let num = from; num <= to; num++) {
        if (isPrime(num)) {
            console.log(num);
        }
    }
}

printPrimes(1, 100);
```

Bubble Sort :

```
        function bubbleSort(arr, n)
        {
            var i, j, temp;
            var swapped;
            for (i = 0; i < n - 1; i++)
            {
                swapped = false;
```

```
        for (j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                // Swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        // IF no two elements were
        // swapped by inner loop, then break
        if (swapped == false)
        break;
    }
}

// Function to print an array
function printArray(arr, size)
{
  var i;
  for (i = 0; i < size; i++)
      console.log(arr[i] + " ");
}

// Driver program
var arr = [ 64, 34, 25, 12, 22, 11, 90 ];
var n = arr.length;
bubbleSort(arr, n);
console.log("Sorted array: ");
printArray(arr, n);
```

Insertion Sort:

```
function insertionSort(arr, n)
{
    let i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
  function printArray(arr, n)
{
    let i;
    for (i = 0; i < n; i++)
        document.write(arr[i] + " ");

}
```

```
// Driver code
    let arr = [12, 11, 13, 5, 6 ];
    let n = arr.length;

    insertionSort(arr, n);
    printArray(arr, n);
```