**Main Topics:**

* **Second largest number in an array efficient way.**
* **Get factorial of number**
* **Singleton Pattern**
* **find-the-frequencies-of-all-duplicates-elements-in-the-array**
* **Next greater element**
* **Sorting Techniques**<http://www.differencebetween.info/difference-between-quick-sort-and-bubble-sort>
* Sort an array:

<https://www.geeksforgeeks.org/how-to-sort-an-array-in-a-single-loop/>

**Since all the known sorting methods use more than 1 loop, it is hard to imagine to do the same with a single loop. Practically, it is not impossible to do so. But doing so won’t be the most efficient.**

**Second largest number efficient way:**

  getSecondLargest() {

    let arr = [1000, 200, 105, 100, 201, 50];

    let max = -Infinity, SecondMax = -Infinity;

    for (let i = 0; i < arr.length; i++)

    {

      if (arr[i] > max)

      {

        SecondMax = max;

        max = arr[i];

      } else if (arr[i] > SecondMax)

      {

        SecondMax = arr[i]

      }

    }

    console.log(SecondMax);

  }

* JavaScript curring

[https://theanubhav.com/2019/02/03/js-currying-in-interview/#3-explicit-call-to-function-with-no-arguments-for-final-result](https://theanubhav.com/2019/02/03/js-currying-in-interview/)

See step 3. Explicit call to function with no arguments for final result

Practice it also.

* Flatten arr = [1,2,3, [[[[4]]]]] => [1,2,3,4]

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/flat>

Inside above link see use a stack

* Find non- repetitive element in string.

nonRepetetiveChar(str = "abdhsasabhcwe") {

let set = new Set(str);

set.forEach(val => {

const regex = new RegExp(`[${val}]`, 'gi');

const matches = str.match(regex);

if (matches && matches.length === 1)

{

console.log(val);

}

})

}

* <https://stackoverflow.com/questions/19395257/how-to-count-duplicate-value-in-an-array-in-javascript>
* <https://www.geeksforgeeks.org/find-the-frequencies-of-all-duplicates-elements-in-the-array/>
* <https://www.geeksforgeeks.org/print-all-the-duplicates-in-the-input-string/>
* Reverse a string without reverse function:
* <script>

      const str = "qwerty";

      const s = str.split("");

      let i = 0, j = s.length - 1;

      while (j > i) {

         const t = s[i];

         s[i] = s[j];

         s[j] = t;

         console.log(s[i]);

         i++;

         j--;

      }

      console.log(s);

</script>

* <https://www.geeksforgeeks.org/longest-consecutive-subsequence/>
* <https://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html>
* Qno. 3  
  <https://www.javatpoint.com/program-to-find-all-permutations-of-a-string>
* <https://www.geeksforgeeks.org/tcs-coding-practice-question-checking-prime-number/>
* <https://www.geeksforgeeks.org/tcs-coding-practice-question-prime-numbers-upto-n/>
* <https://www.geeksforgeeks.org/tcs-coding-practice-question-sum-of-digits-of-a-number/>
* <https://www.geeksforgeeks.org/return-maximum-occurring-character-in-the-input-string/>
* Sum of natural numbers n(n+1)/2
* **Time complexity:**

<https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/>

Instead of measuring actual time required in executing each statement in the code, we consider how many times each statement execute.

<https://www.geeksforgeeks.org/practice-questions-time-complexity-analysis/>

Which is better o(n) or o(log n)?

O(log n)

<https://stackoverflow.com/questions/10369563/difference-between-on-and-ologn-which-is-better-and-what-exactly-is-olo>

* Get decoded string from reversed Ascii code

001801411111782311180180110127=> 721011081081113287111114108100 => Hello World

  getDecodedCode(encoded: string) {

    let arr = encoded.split("").reverse();

    let split = 2;

    let val;

    let chr = "";

    while (arr.length > 2)

    {

      val = arr.splice(0, split).join("");

      if (!((parseInt(val) > 65 && parseInt(val) < 100) || parseInt(val) === 32))

      {

        split = 3;

        val += arr.splice(0, 1).join("");

      }

      chr += String.fromCharCode(parseInt(val));

      split = 2;

    }

    console.log(chr);

  }

 getFactorial(num: number) {

    let res = num;

    while (num > 1)

    {

      res \*= num - 1;

      num--;

    }

    return res;

  }

* <https://medium.com/@rbiswas596/hackerrank-javascript-challenges-f2e822160df6>
* <https://programs.programmingoneonone.com/2021/03/hackerrank-minimum-swaps-2-solution.html>
* A *left rotation* operation on an array shifts each of the array's elements  unit to the left. For example, if  left rotations are performed on array , then the array would become . Note that the lowest index item moves to the highest index in a rotation. This is called a *circular array*.

function rotLeft(a, d) {

    const l = a.length;

    let diff;

    const res = [];

    if(d>=l){

        d = d%l;

        if(d === 0){

            return a;

        }

    }

    for(let i = l-1;i>=0;i--){

        diff = i-d;

          if(diff<0){

            diff = diff + l

        }

        res[diff] = a[i]

    }

    return res;

    // Write your code here

}

* **Optum shared questions:**
* Program to find the frequency of each element in the array OR write a program to count the occurrence of each number in a given string

  getFrequencyInArray(arr: number[]) {

    arr.sort((x, y) => x - y);

    let a = arr[0];

    let count = 0;

    for (let i = 0; i < arr.length; i++)

    {

      if (a === arr[i])

      {

        count++;

      } else

      {

        console.log(`${a} => ${count}`);

        count = 1;

        a = arr[i];

      }

    }

    console.log(`${a} => ${count}`);

    console.log(arr);

  }

2. Verify if a given string is an anagram

e.g. string1: LISTEN string 2: SILENT

3. Remove 3 or more consecutive characters from a string, repeat until there are no more.

e.g.ABCCCCBBA -> ABBBA ->AA

  removeConsecutiveChars(str = "ABCCCCBCBCCCCCCCCBBCCA") {

    let arr = str.split("");

    let dis = new Set(arr);

    let disLength = 0;

    while (dis.size !== disLength)

    {

      disLength = dis.size;

      dis.forEach((val) => {

        let regex = new RegExp(`[${val}]{3,}`, 'gi');

        let matches = str.match(regex);

        if (matches && matches.length > 0)

        {

          matches.forEach((val) => {

            str = str.replace(val, "");

          })

          dis = new Set(str);

          disLength = 0;

        }

      });

      console.log(str);

    }

  }

4. Given a binary {0,1,1,0,0,1,0,0,1}, sort the array in a way that all zeros come to the left and all the ones come to the right side of the array.

* Generate 10 buttons dynamically with 1,2,3,4…

function clickMe(elem){

let btnElem;

for(let i = 0; i< 11;i++){

btnElem = document.createElement("button");

textNote = document.createTextNode(i);

btnElem.appendChild(textNote);

document.body.appendChild(btnElem);

btnElem.addEventListener('click', () => {console.log(i)})

}

}

* Print 10 to 1 in the interval of 1 seconds

    for (let i = n; i > 0; i--)

    {

      setTimeout(() => {

        console.log(i);

      }, (n - i) \* 1000);

    }

// using Promise

function printAfterDelay(inpArr) {

let arr = [...inpArr];

return async function y() {

if (arr.length > 0)

{

const next = arr.pop();

let fn = await delayFn(next, y);

return fn();

}

return 1;

}

}

function delayFn(num, cb){

return new Promise(resolve => {

setTimeout(() => {

console.log(num);

resolve(cb);

}, 1000);

})

}

printAfterDelay(inpArr)();

* <https://codeburst.io/100-coding-interview-questions-for-programmers-b1cf74885fb7>
* Use Hashset in c# and Set in javascript.
* Given a string possibly containing three types of braces ({}, [], ()), write a function that returns a Boolean indicating whether the given string contains a valid nesting of braces.

[https://www.educative.io/blog/level-up-javascript-coding-challenges#balanced-brackets](https://www.educative.io/blog/level-up-javascript-coding-challenges)

{ab} // true

{(}) // false

See Armstrong number in next question

* <https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/>

See all the related articles

* <https://www.geeksforgeeks.org/c-program-find-gcd-hcf-two-numbers/>
* <https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/>
* <https://www.educative.io/edpresso/how-to-implement-infinite-scrolling-in-javascript>

Hard level

* <https://www.geeksforgeeks.org/length-of-the-longest-substring-without-repeating-characters/>

See below solution:

Time Complexity: O(n + d) where n is length of the input string and d is number of characters in input string alphabet. For example, if string consists of lowercase English characters then value of d is 26.

Auxiliary Space: O(d)

  // geeksforgeeks

  longestUniqueSubsttr(str = "geeksforgeeks") {

    // let m = new Map();

    let m = new Set();

    let arr = str.split('');

    let max = 0, count = 0;

    for (let i = 0; i < arr.length; i++)

    {

      if (m.has(arr[i]))

      {

        count = 1

        m.clear();

      } else

      {

        count++;

      }

      m.add(arr[i]);

      // m.set(arr[i], i);

      if (count > max)

      {

        max = count;

      }

    }

    console.log(max);

  }

* <https://www.geeksforgeeks.org/next-greater-element/?ref=lbp>
* <https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/>
* <https://www.ibm.com/in-en/cloud/learn/three-tier-architecture>

N-tier architecture - also called or multi-tier architecture - refers to any application architecture with more than one tier. But applications with more than three layers are rare, because additional layers offer few benefits and can make the application slower, harder to manage and more expensive to run.

* SOC:

Separation of concerns is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. For example the business logic of the application is a concern and the user interface is another concern.

<https://github.com/abhishek18011995/study-material.git>