Question: Implement own bind

Function.prototype.bind = function(context){

    const fn = this;

    const mainArguments = [].slice.call(arguments, 1);

    return function() {

        const currentArgs = [].slice.call(arguments);

        return fn.apply(context, [...mainArguments, ...currentArgs]);

    }

}

Question: Debounce

function debounce(func, interval, callfirst){

    let timeout;

    return function () {

        const context = this, args = arguments;

        let delay = function() {

            timeout = null;

            if(!callfirst){

            func.apply(context, args);

            }

        }

        const isCallFirst =  callfirst && !timeout;

        clearTimeout(timeout);

        timeout = setTimeout(delay, interval|| 500);

        if(isCallFirst) {

          func.apply(context, args);

        }

    }

}

Question : currying

function curry(func){

return  function (){

          const arguments1 = arguments;

        if(arguments1.length >= func.length){

            return func.apply(this, arguments1);

        }else{

        return function(){

             return curry.apply(this, func.bind(argument1.concat(arguments)))

         }

        }

    }

}

curriedSum(10, 20, 30) => 60

curriedSum(10, 20)(30) => 60

curriedSum(10)(20)(30) => 60

let curriedSum = curry(sum);

Question: implement loadsh get

\_\_get(object, keys, defaultVal = null): any {

keys = Array.isArray(keys) ? keys : keys.replace(/(\[(\d)\])/g, '.$2').split('.'); // split by dot of array

object = object[keys[0]];

if (object && keys.length > 1) {

return this.\_\_get(object, keys.slice(1), defaultVal);

}

return object === undefined ? defaultVal : object;

}

Question Link list

// adds an element at the end

// of list

add(element)

{

// creates a new node

var node = new Node(element);

// to store current node

var current;

// if list is Empty add the

// element and make it head

if (this.head == null)

this.head = node;

else {

current = this.head;

// iterate to the end of the

// list

while (current.next) {

current = current.next;

}

// add node

current.next = node;

}

this.size++;

}

// insert element at the position index

// of the list

insertAt(element, index)

{

if (index > 0 && index > this.size)

return false;

else {

// creates a new node

var node = new Node(element);

var curr, prev;

curr = this.head;

// add the element to the

// first index

if (index == 0) {

node.next = head;

this.head = node;

} else {

curr = this.head;

var it = 0;

// iterate over the list to find

// the position to insert

while (it < index) {

it++;

prev = curr;

curr = curr.next;

}

// adding an element

node.next = curr;

prev.next = node;

}

this.size++;

}

}

// removes an element from the

// specified location

removeFrom(index)

{

if (index > 0 && index > this.size)

return -1;

else {

var curr, prev, it = 0;

curr = this.head;

prev = curr;

// deleting first element

if (index == = 0) {

this.head = curr.next;

} else {

// iterate over the list to the

// position to removce an element

while (it < index) {

it++;

prev = curr;

curr = curr.next;

}

// remove the element

prev.next = curr.next;

}

this.size--;

// return the remove element

return curr.element;

}

}

// removes a given element from the

// list

removeElement(element)

{

var current = this.head;

var prev = null;

// iterate over the list

while (current != null) {

// comparing element with current

// element if found then remove the

// and return true

if (current.element == = element) {

if (prev == null) {

this.head = current.next;

} else {

prev.next = current.next;

}

this.size--;

return current.element;

}

prev = current;

current = current.next;

}

return -1;

}

// finds the index of element

indexOf(element)

{

var count = 0;

var current = this.head;

// iterae over the list

while (current != null) {

// compare each element of the list

// with given element

if (current.element == = element)

return count;

count++;

current = current.next;

}

// not found

return -1;

}

// gives the size of the list

size\_of\_list()

{

console.log(this.size);

}

class Node{

    constructor(data, next = null){

        this.data = data,

        this.next = next

    }

}

Question : memorization

const memoize = (callback, threshold = 1000) => {

let memo = new LRU(threshold);

return(args) => {

if(memo.get(args) != undefined){

console.log("from cache");

console.log(memo.getSizeOfCache())

return memo.get(args);

}else{

memo.set(args, callback(args))

return  memo.get(args);

}

}

}

Question : LRU

class LRU{

constructor(threshold = 100){

    this.max = threshold;

    this.cache = new Map();

}

get(key){

    // console.log("iun get", key)

    let item = this.cache.get(key);

     console.log(key,item)

    if(item != undefined){

        this.cache.delete(key);

        this.cache.set(key, item);

    }

   return item;

}

set(key, val){

    if(this.cache.has(key)){

         this.cache.delete(key);

    }

    if(this.cache.size === this.max){

        this.cache.delete(this.getFirst())

    }

    this.cache.set(key, val);

}

getFirst(){

    return this.cache.keys().next().value;

}

getSizeOfCache(){

    return this.cache.keys()

}

}

Question : chocolate/wrapper

static int countMaxChoco(int money,

                        int price, int wrap)

    {

        // Corner case

        if (money < price)

            return 0;

        // First find number of chocolates

        // that can be purchased with the

        // given amount

        int choc = money / price;

        // Now just add number of chocolates

        // with the chocolates gained by

        // wrapprices

        choc = choc + (choc - 1) / (wrap - 1);

        return choc;

    }

// recursion

int countRec(int choc, int wrap)

{

    // If number of chocolates is less than

    // number of wrappers required.

    if (choc < wrap)

        return 0;

    // We can immediatly get newChoc using

    // wrappers of choc.

    int newChoc = choc/wrap;

    // Now we have "newChoc + choc%wrap" wrappers.

    return newChoc + countRec(newChoc + choc%wrap,

                              wrap);

}

Question : maxDiff from an array

function getMaxiff (arr){

const length = arr.length;

let maxDiff = -100

if(!length){

return maxDiff;

}

let maximumRightValue = arr[length-1];

for(let i = length-2; i>=0 ; i--){

if(arr[i] > maximumRightValue){

maximumRightValue = arr[i];

}else{

const currentDiff = maximumRightValue - arr[i];

// console.log(currentDiff, arr[i])

if(currentDiff > maxDiff){

maxDiff = currentDiff;

}

}

}

return maxDiff;

}

console.log(getMaxiff([2, 3, 10, 6, 4, 8, 1]))

Question: Implement map function

const hoMyMap = function(callback, arr) {

    let resultArray = [];

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

        resultArray.push(callback(arr[i], i, this))

    }

    return resultArray;

}

Memorization for async

function memo(func, isAsync){

    let memo = {};

    const x = function(args){

        memo[args] = memo[args] || func.apply(this, args);

        return  memo[args];

    }

    x.store = memo;

    if(isAsync){

        return async function(){

            const args = JSON.stringify(arguments)

            x(arguments);

        }

    }else{

        return function(){

            const args =[].slice.call(arguments);

            x(args);

        }

    }

}

Memoization with cache return

function memo(func){

    let memo = {};

    let a = function(){

        const args = [].slice.call(arguments);

        if(memo[args]){

            return memo[args];

        }else{

            memo[args] = func.apply(this, args);

            return  memo[args];

        }

    };

    a.store = memo;

    return a;

}

Stock profit maximize (Single trans)

var maxProfit = function(prices) {

    let result = 0;

    let min = prices[0];

    for(let i = 1; i < prices.length; i++) {

        min = Math.min(prices[i], min);

        result = Math.max(result, prices[i] - min);

    }

    return result;

};

Stock profit maximize multiple transaction

var maxProfit = function(prices) {

    let profit = 0;

    for (let i = 0; i < prices.length - 1; i++) {

        const possibleProfit = prices[i + 1] - prices[i];

        profit = Math.max(profit + possibleProfit, profit);

    }

    return profit;

};

var maxProfit = function(prices) {

    if(prices.length < 2) return 0;

    let min = prices[0], sum = 0;

    for(let i = 1 ; i < prices.length ; i++){

        if(prices[i]  >= prices[i - 1]){

            sum += (prices[i] - prices[i-1]);

        }

    }

    return sum;

};

Stock profit(sell before buying)

var maxProfit = function(prices) {

    let haveOne = -prices[0]

    let haveTwo = -Infinity

    let notHaveOne = -Infinity

    let notHaveTwo = -Infinity

    for (let i = 1; i < prices.length; i++) {

        haveOne = Math.max(haveOne, -prices[i])

        haveTwo = Math.max(haveTwo, notHaveOne-prices[i])

        notHaveOne = Math.max(notHaveOne, haveOne+prices[i])

        notHaveTwo = Math.max(notHaveTwo, haveTwo+prices[i])

    }

    return Math.max(0, notHaveOne, notHaveTwo)

};

**Longest Word in Dictionary**

var longestWord = function(words) {

    let set = new Set(words);

    let res = "";

    words.forEach(a => {

        if(a.length < res.length) return;

        if(a.length == res.length && a > res) return;

        for(let i = a.length - 1; i> 0 ; i--) {

            if( !set.has( a.substring(0, i))) return

        }

        res = a;

    })

    return res;

};

**Jump Game**

var jump = function(nums) {

    let currFarest = 0

    let currEnd = 0

    let jump = 0

    for (let i = 0; i < nums.length - 1; i++) {

      currFarest = Math.max(currFarest, i + nums[i])

      // Improvement

      if (currFarest >= nums.length -1) return jump+1

      if (currEnd === i) {

        jump++

        currEnd = currFarest

      }

    }

    return jump

  };

function jump(nums) {

    var max = 0;

    var nextMax = 0;

    var jumps = 0;

    nums.some((v, i) => {

        if (max >= nums.length - 1) {

            return true;

        }

        nextMax = Math.max(i + v, nextMax);

        if (i === max) {

            max = nextMax;

            jumps++;

        }

    });

    return jumps;

}

Rotten eggs problem :

var orangesRotting = function(grid) {

    const height = grid.length;

    const width = grid[0].length;

    let fresh = 0;

    const queue = [];

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

      for (let j = 0; j < width; j++) {

        if (grid[i][j] === 2) queue.push([i, j]);

        if (grid[i][j] === 1) fresh++;

      }

    }

    let minute = 0;

    while (queue.length) {

      const size = queue.length;

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

        const [x, y] = queue.shift();

        if (x - 1 >= 0 && grid[x - 1][y] === 1) {

          grid[x - 1][y] = 2;

          fresh--;

          queue.push([x - 1, y]);

        }

        if (x + 1 < height && grid[x + 1][y] === 1) {

          grid[x + 1][y] = 2;

          fresh--;

          queue.push([x + 1, y]);

        }

        if (y - 1 >= 0 && grid[x][y - 1] === 1) {

          grid[x][y - 1] = 2;

          fresh--;

          queue.push([x, y - 1]);

        }

        if (y + 1 < width && grid[x][y + 1] === 1) {

          grid[x][y + 1] = 2;

          fresh--;

          queue.push([x, y + 1]);

        }

      }

      if (queue.length > 0) minute++;

    }

    return fresh === 0 ? minute : -1;

  };

BFS

var orangesRotting = function(grid) {

    const height = grid.length;

    const width = grid[0].length;

    let fresh = 0;

    const queue = [];

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

      for (let j = 0; j < width; j++) {

        if (grid[i][j] === 2) queue.push([i, j]);

        if (grid[i][j] === 1) fresh++;

      }

    }

    let minute = 0;

    while (queue.length) {

      const size = queue.length;

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

        const [x, y] = queue.shift();

        if (x - 1 >= 0 && grid[x - 1][y] === 1) {

          grid[x - 1][y] = 2;

          fresh--;

          queue.push([x - 1, y]);

        }

        if (x + 1 < height && grid[x + 1][y] === 1) {

          grid[x + 1][y] = 2;

          fresh--;

          queue.push([x + 1, y]);

        }

        if (y - 1 >= 0 && grid[x][y - 1] === 1) {

          grid[x][y - 1] = 2;

          fresh--;

          queue.push([x, y - 1]);

        }

        if (y + 1 < width && grid[x][y + 1] === 1) {

          grid[x][y + 1] = 2;

          fresh--;

          queue.push([x, y + 1]);

        }

      }

      if (queue.length > 0) minute++;

    }

    return fresh === 0 ? minute : -1;

  };

Task runner

const exampleTaskA = (name) => new Promise(resolve => setTimeout(function() {

    console.log(`Task ${name} Done`);

    resolve()

  }, Math.floor(Math.random() \* 2000)))

  function TaskRunner(concurrency) {

    this.limit = concurrency;

    this.store = [];

    this.active = 0;

  }

  TaskRunner.prototype.next = function() {

    if (this.store.length) this.runTask(...this.store.shift())

  }

  TaskRunner.prototype.runTask = function(task, name) {

    this.active++

    console.log(`Scheduling task ${name} current active: ${this.active}`)

    task(name).then(() => {

      this.active--

      console.log(`Task ${name} returned, current active: ${this.active}`)

      this.next()

    })

  }

  TaskRunner.prototype.push = function(task, name) {

    if (this.active < this.limit) this.runTask(task, name)

    else {

      console.log(`queuing task ${name}`)

      this.store.push([task, name])

    }

  }

  var task = new TaskRunner(2);

  task.push(exampleTaskA, 1)

  task.push(exampleTaskA, 2)

Sort character by frequency

var frequencySort = function(s) {

    var map = {};

    var result = '';

    var stringArray = s.split('');

    //put the character count into a map

    for(var i=0;i<stringArray.length;i++){

        map[stringArray[i]] = map[stringArray[i]] + 1 || 1;

    }

    //sort the map first, then push into the result

    Object.keys(map).sort((a,b)=>map[b]-map[a]).forEach(function(v){

        for(var j=0;j<map[v];j++){

            result += v;

        }

    });

    return result;

};

**Two Sum**

const twoSum = (nums, target) => {

    const map = {};

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

      const another = target - nums[i];

      if (another in map) {

        return [map[another], i];

      }

      map[nums[i]] = i;

    }

    return null;

  };

Median of two sorted array

var findMedianSortedArrays = function(nums1, nums2) {

    //ES6 syntax to break down the 2 arrays

       let num = [...nums1, ...nums2];

    //Sort in numerical order

     num.sort((a,b) => a-b);

     if(num.length % 2 != 0){

         //Return the middle value if number of total elements is odd

          return (num[Math.floor(num.length/2)]);

     }else{

         //Return the average of the middle values if number of total elements is even

          return (num[Math.floor(num.length/2)-1] + num[Math.floor(num.length/2)])/2;

     }

 };

**Container With Most Water**

var maxArea = function(height) {

    let max = 0;

    let i = 0;

    let j = height.length - 1;

    while(i < j){

        let cur = (j - i) \* Math.min(height[i], height[j]);

        max = Math.max(cur, max);

        height[i] <= height[j] ? i ++ : j --;

    }

    return max;

};

Triplet sum

var threeSum = function(nums) {

    var rtn = [];

    if (nums.length < 3) {

        return rtn;

    }

    nums = nums.sort(function(a, b) {

        return a - b;

    });

    for (var i = 0; i < nums.length - 2; i++) {

        if (nums[i] > 0) {

            return rtn;

        }

        if (i > 0 && nums[i] == nums[i - 1]) {

            continue;

        }

        for (var j = i + 1, k = nums.length - 1; j < k;) {

            if (nums[i] + nums[j] + nums[k] === 0) {

                rtn.push([nums[i], nums[j], nums[k]]);

                j++;

                k--;

                while (j < k && nums[j] == nums[j - 1]) {

                    j++;

                }

                while (j < k && nums[k] == nums[k + 1]) {

                    k--;

                }

            } else if (nums[i] + nums[j] + nums[k] > 0) {

                k--;

            } else {

                j++;

            }

        }

    }

    return rtn;

};

[Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array)

var removeDuplicates = function(nums) {

    let lastIndex = 0;

    nums.forEach(n => {

        if(nums[lastIndex] != n){

            lastIndex++;

            nums[lastIndex] = n;

        }

     });

    return lastIndex +1;

};

**Next Permutation**

var nextPermutation = function(nums) {

    // find the first descending element, nums[i-1] that satisfies nums[i-1] < nums[i] in the array from the right

    let i = nums.length - 1;

    while (i > 0) {

        if (nums[i-1] >= nums[i]) {

            i--;

        } else {

            break;

        }

    }

    // swap nums[i-1] with the smallest element between nums[i] and nums[length-1] that is larger than nums[i-1] (only do this step if the current array is not entirely descending like [4, 3, 2, 1])

    let j = nums.length - 1;

    while (j > i) {

        if (nums[j] <= nums[i-1]) {

            j--;

        } else {

            break;

        }

    }

    if (i !== 0) {

        swap(nums, i-1, j);

    }

    // reverse the part between nums[i] and nums[length-1] by using swap

    let mid = Math.floor((i+nums.length)/2);

    for (let k = i; k < mid; k++) {

        swap(nums, k, nums.length - k + i - 1);

    }

};

function swap(arr, i, j) {

    let temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

Find index in rotated sorted array

ar search = function(nums, target) {

    let start = 0,

      end = nums.length - 1;

    while (start < end) {

      let mid = Math.floor((start + end) / 2);

      // fast path, early returns...

      if (nums[mid] === target) return mid;

      if (nums[start] === target) return start;

      if (nums[end] === target) return end;

      // left part is sorted...

      if (nums[mid] > nums[start]) {

        if (target > nums[start] && target < nums[mid]) {

          end = mid - 1;

        } else {

          start = mid + 1;

        }

      }

      // right part is sorted...

      else if (nums[mid] < nums[end]) {

        if (target > nums[mid] && target < nums[end]) {

          start = mid + 1;

        } else {

          end = mid - 1;

        }

      } else {

        return -1;

      }

    }

    return nums[start] === target ? start : -1;

  };

Find the set of combination of sum

var combinationSum = function(candidates, target) {

    if (!candidates || !candidates.length) { return []; }

    candidates.sort((a,b) => a - b);

    const solutions = [];

    const findCombos = function(candIdx, subtotal, solution) {

        for (let i = candIdx; i < candidates.length; i++) {

            if (subtotal + candidates[i] === target) {

                solutions.push(solution.concat(candidates[i]));

            } else if (subtotal + candidates[i] < target) {

                findCombos(i, subtotal + candidates[i], solution.concat(candidates[i]));

            }

        };

    };

    findCombos(0, 0, []);

    return solutions;

};

First missing positive

var firstMissingPositive = function(nums) {

    let next = 1;

    let index = 0;

    while(index < nums.length) {

        if(nums[index] == next) {

            next += 1;

            index = 0;

        } else {

            index += 1;

        }

    }

    return next;

};

Rotate image by 90 degree

Transpose the matrix

Reverse each row

var rotate = function(matrix) {

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

        for (let j=i;j<matrix[0].length;j++) {

            let temp = matrix[i][j];

            matrix[i][j] = matrix[j][i];

            matrix[j][i] = temp;

        }

    }

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

        for (let j=0;j<matrix[0].length/2;j++) {

            let temp = matrix[i][j];

            matrix[i][j] = matrix[i][matrix[0].length-j-1];

            matrix[i][matrix[0].length-j-1] = temp;

        }

    }

};

Maximum sum subarray

var maxSubArray = function(nums) {

    if(nums.length == 0) return 0;

    let result = Number.MIN\_SAFE\_INTEGER;

    let sum = 0;

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

        sum += nums[i];

        result = Math.max(sum, result);

        sum = sum < 0 ? 0 : sum;

    }

    return result;

};

Spiral matrix

var spiralOrder = function(matrix) {

    let res=[];

    if(matrix.length==0) return [];

    res.push(...matrix.shift());

    while(matrix.length>0){

        //rotate left

        matrix=matrix[0].map((item,index,arr)=>{return matrix.map(x=> x[arr.length-1-index])});

        //shift

        res.push(...matrix.shift());

    }

    return res

};

Merge intervals

function merge(intervals) {

    if (!intervals.length) return intervals

    intervals.sort((a, b) => a.start !== b.start ? a.start - b.start : a.end - b.end)

    var prev = intervals[0]

    var res = [prev]

    for (var curr of intervals) {

      if (curr.start <= prev.end) {

        prev.end = Math.max(prev.end, curr.end)

      } else {

        res.push(curr)

        prev = curr

      }

    }

    return res

  }

Unique path in matrix

var uniquePaths = function(m, n) {

    var arr = new Array(m);

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

        arr[i] = new Array(n);

      }

      let res = numberOfPathsRecDP(arr,m-1,n-1);

      //console.log(arr2D)

      return res

    };

    function numberOfPathsRecDP(memo,m,n){

      //console.log('index',m,n)

      if(m === 0|| n === 0) {

        memo[m][n] = 1

        return 1;

      }

      if(memo[m][n] === undefined){

        //console.log(n)

        memo[m][n] = numberOfPathsRecDP(memo,m-1,n) + numberOfPathsRecDP(memo,m,n-1)

      }

      return memo[m][n];

    }

Plus one to the array

var plusOne = function(digits) {

    for(let i = digits.length - 1; i >= 0; i --){

      if(digits[i] === 9){

        digits[i] = 0;

      }

      else {

        digits[i] ++;

        return digits;

      }

    }

    return [1, ...digits];

  };

Sort colors inplace

function sortColors (nums) {

    let low = 0, high = nums.length - 1

    for (let i = 0; i <= high;i++) {

        if (nums[i] === 0) {

            [nums[i], nums[low]] = [nums[low], nums[i]]

            low++;

        } else if (nums[i] == 2) {

            [nums[i], nums[high]] = [nums[high], nums[i]]

            high--;i--

        }

    }

};

Number of subset

var subsets = function(nums) {

    let result = [];

    dfs([], 0);

    function dfs(current, index){

        result.push(current);

        for(let i = index; i < nums.length; i++) {

            dfs(current.concat(nums[i]), i + 1);

        }

    }

    return result;

};

const subsets = nums => {

    let res = [[]],

        curr;

    for(let num of nums) {

        curr = res.map(x => [...x, num]);

        res = [...res, ...curr];

    }

    return res;

};

Maximum rectangle

var maximalRectangle = function(matrix) {

    if (matrix.length === 0) {

        return 0

    }

    const heights = new Array(matrix[0].length + 1).fill(0)

    let ret = 0

    matrix.forEach(line => {

        line.forEach((flag, i) => {

            heights[i] = flag === '1' ? heights[i] + 1 : 0

        })

        const stack = [[0, -1]]

        let top = 0

        heights.forEach((height, index) => {

            let memoIndex = index

            while (stack[top][0] > height) {

                const [h, i] = stack.pop()

                ret = Math.max(ret, (index - i) \* h)

                memoIndex = i

                top--

            }

            if (stack[top][0] < height) {

                stack.push([height, memoIndex])

                top++

            }

        })

    })

    return ret

};

Parenthesis check

var isValid = function(s) {

    let map = {

        ")": "(",

        "]": "[",

        "}": "{"

    }

    let arr = [];

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

        if(s[i] === "(" || s[i] === "[" || s[i] === "{"){

            arr.push(s[i]);

        }

        else{

            if(arr[arr.length - 1] === map[s[i]]){

                arr.pop();

            }

            else return false;

        }

    }

    return arr.length === 0 ? true : false;

};

Reverse Link list

var reverseList = function(head) {

    let pre = null

    while(head){

        const next = head.next

        head.next = pre

        pre = head

        head = next

    }

    return pre

};

Group Anagram

const groupAnagrams = strs => {

    const map = {};

    for (let str of strs) {

        const key = [...str].sort().join('');

        if (!map[key]) {

            map[key] = [];

        }

        map[key].push(str);

    }

    return Object.values(map);

};

Intersection of two array

function intersection(nums1, nums2) {

    const set = new Set(nums1);

    return [...new Set(nums2.filter(n => set.has(n)))];

}

const intersection = (nums1, nums2) => {

    return [...new Set(nums1.filter(num => nums2.includes(num)))]

  };

Set properties

function isSuperset(set, subset) {

    for (let elem of subset) {

        if (!set.has(elem)) {

            return false

        }

    }

    return true

}

function union(setA, setB) {

    let \_union = new Set(setA)

    for (let elem of setB) {

        \_union.add(elem)

    }

    return \_union

}

function intersection(setA, setB) {

    let \_intersection = new Set()

    for (let elem of setB) {

        if (setA.has(elem)) {

            \_intersection.add(elem)

        }

    }

    return \_intersection

}

function symmetricDifference(setA, setB) {

    let \_difference = new Set(setA)

    for (let elem of setB) {

        if (\_difference.has(elem)) {

            \_difference.delete(elem)

        } else {

            \_difference.add(elem)

        }

    }

    return \_difference

}

function difference(setA, setB) {

    let \_difference = new Set(setA)

    for (let elem of setB) {

        \_difference.delete(elem)

    }

    return \_difference

}

// Examples

let setA = new Set([1, 2, 3, 4])

let setB = new Set([2, 3])

let setC = new Set([3, 4, 5, 6])

isSuperset(setA, setB)          // => true

union(setA, setC)               // => Set [1, 2, 3, 4, 5, 6]

intersection(setA, setC)        // => Set [3, 4]

symmetricDifference(setA, setC) // => Set [1, 2, 5, 6]

difference(setA, setC)          // => Set [1, 2]

Unique email address

var numUniqueEmails = function(emails) {

    return new Set(emails.map(function(item, index) {

        var [name, domain] = item.split("@")

        var [nameNoPlus, namePlus] = name.split("+");

        var nameSimplified = nameNoPlus.split(".").join("")

        var emailSimplified = nameSimplified + "@" + domain;

        return emailSimplified;

    })).size;

};

First unique character index

var firstUniqChar = function(s) {

    var map=new Map();

    let deletehash = {};

    for(i=0;i<s.length;i++){

         if(map.has(s[i])){

             map.delete(s[i]);

             deletehash[s[i]] = i;

         }

         else{

             if(!deletehash[s[i]]){

                 map.set(s[i],i);

             }

         }

     }

    if( map.values().next().value ||  map.values().next().value === 0){

        return  map.values().next().value;

    }

    return -1;

};

**Subarray Sum Equals K**

var subarraySum = function(nums, k) {

    let map = {0: 1};

    let sum = 0;

    let count = 0;

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

         console.log(sum)

        console.log(map)

        sum += nums[i];

        if (map[sum - k]) {

            count += map[sum - k];

            console.log("in", sum-k)

        }

        map[sum] = map[sum] ? map[sum] + 1 : 1;

    }

    return count;

};

Convert dec to binary

function binary(n){

let s = "";

while(n > 0){

    s = n % 2 + s

    n = parseInt(n / 2);

}

return s;

}

parseInt(num, radix);

Convert binary to dec

A picture containing clock

Description automatically generated

**Pow(x, n)**

var myPow = function(x, n) {

    if (n === 0) return 1

    else if (n === 1) return x;

    else if (n === -1) return 1 / x;

    else if (n % 2 === 0) {

        const m = myPow(x, n/2);

        return m \* m

    }

    else return x \* myPow(x, n - 1);

};

**K-th Symbol in Grammar**

var kthGrammar = function(N, K) {

    if (N === 1) return 0;

    const mother = kthGrammar(N-1, parseInt((K+1)/2));

    return K % 2 === 0 ? !mother : mother;

};

// This function returns the complementary value

var cpl = function(n){

    if(n==0){

        return 1

    }

    if(n==1){

        return 0

    }

}

var kthGrammar = function(N, K) {

// If N is 1 or K is 1 the first value has to be 0 so we check that

   if(N==1 || K==1){

       return 0

   }

// If N is 2 and K is 2 is the simplest occurrence of 1

   if(N==2 && K==2){

       return 1

   }

// When K is smaller or equal than 2^N-1 the solution is the same as the kthGrammar(N-1, K) problem

   if(K<=Math.pow(2, N-1)){

       return kthGrammar(N-1, K)

   }

// When K is larger than 2^N-1 the solution is the complementary value (opposite) of the kthGrammar(N, K-Math.pow(2, N-1))

   return cpl(kthGrammar(N, K-Math.pow(2, N-1)))

}

Permutation of array

var permute = function(nums, n = 0) {

    if (n >= nums.length) return [[]];

    const res = [];

    const prevs = permute(nums, n + 1);  // permutations of elements after n

    for (let prev of prevs) {

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

            let p = prev.slice(0);

            p.splice(i, 0, nums[n]);  // successively insert element n

            res.push(p);

        }

    }

    return res;

};

**Generate Parentheses**

var generateParenthesis = function(n) {

    var arr = [];

    compose(n, n, '');

    return arr;

    function compose(left, right, str) {

      if (!left && !right && str.length) return arr.push(str);

      if (left) compose(left - 1, right, str + '(');

      if (right > left) compose(left, right - 1, str + ')');

    }

  };

**Longest Substring Without Repeating Characters**

var lengthOfLongestSubstring = function(s) {

    if (!s) return 0;

    const positions = {};

    let startIndex = 0;

    let maxLength = 0;

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

        const char = s.charAt(i);

        if (positions[char] !== undefined && positions[char] >= startIndex) {

            maxLength = Math.max(maxLength, i - startIndex);

            startIndex = positions[char] + 1;

        }

        positions[char] = i;

    }

    maxLength = Math.max(maxLength, s.length - startIndex);

    return maxLength;

};

Add to list in reverse

var addTwoNumbers = function(l1, l2) {

    var List = new ListNode(0);

    var head = List;

    var sum = 0;

    var carry = 0;

    while(l1!==null||l2!==null||sum>0){

        if(l1!==null){

            sum = sum + l1.val;

            l1 = l1.next;

        }

        if(l2!==null){

            sum = sum + l2.val;

            l2 = l2.next;

        }

        if(sum>=10){

            carry = 1;

            sum = sum - 10;

        }

        head.next = new ListNode(sum);

        head = head.next;

        sum = carry;

        carry = 0;

    }

    return List.next;

};

[ZigZag Conversion](https://leetcode.com/problems/zigzag-conversion)

function convert(s, numRows) {

    if (numRows === 1) {

      return s;

    }

    const N = s.length;

    const arr = [...Array(numRows)].map(r => []);

    for (let i = 0; i < N; i++) {

      const pos = i % (2\*numRows-2);

      const ii = pos < numRows ? pos : 2\*numRows-2-pos;

      arr[ii].push(s[i]);

    }

    return arr.map(r => r.join('')).join('');

  }

[String to Integer (atoi)](https://leetcode.com/problems/string-to-integer-atoi)

The Idea

Trim

Get the sign

Get the digit

var myAtoi = function(str) {

    let i=0, sign = 1, num = 0, MIN = -2147483648, MAX = 2147483647;

    str = str.trim();

    if (str[i]=='-' || str[i]=='+') sign = str[i++]=='-'?-1:1;

    while (str[i] && str[i].charCodeAt(0)-48 <= 9 && str[i].charCodeAt(0)-48 >= 0) {

        num = num\*10 + (str[i++].charCodeAt(0)-48);

    }

    num = sign\*num;

    return num<=MIN?MIN:num>=MAX?MAX:num;

};

**Search Insert Position**

var searchInsert = function(nums, target) {

    let index = -1;

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

        if(nums[i] === target || nums[i] > target){

            index = i;

            break;

        }

    }

    return index === -1 ? nums.length : index;

};

[Unique Paths in grid](https://leetcode.com/problems/unique-paths)

const uniquePaths = (m, n) => {

    const makeMatrix = (m, n) => Array(m).fill(Array(n).fill(1));

    let matrix = makeMatrix(m, n);

    for(let i = 1; i < m; i++) {

        for(let j = 1; j < n; j++) {

            matrix[i][j] = matrix[i-1][j]+matrix[i][j-1];

        }

    }

    return matrix[m-1][n-1];

};

Unique Paths in grid with obstacle

var uniquePathsWithObstacles = function(obstacleGrid) {

    let mtrx = obstacleGrid;

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

        for (let j = 0; j < mtrx[0].length; j++) {

            if (!i && !j) mtrx[i][j] !== 1 ? mtrx[i][j] = 1 : mtrx[i][j] = 0;

            else if (!j) mtrx[i][j] !== 1 ? mtrx[i][j] = mtrx[i - 1][j] : mtrx[i][j] = 0;

            else if (!i) mtrx[i][j] !== 1 ? mtrx[i][j] = mtrx[i][j - 1] : mtrx[i][j] = 0;

            else mtrx[i][j] !== 1 ? mtrx[i][j] = mtrx[i - 1][j] + mtrx[i][j - 1] : mtrx[i][j] = 0;

        }

    }

    return mtrx[mtrx.length - 1][mtrx[0].length - 1];

};

**Remove Duplicates from Sorted List**

/\*\*

 \* Definition for singly-linked list.

 \* function ListNode(val) {

 \*     this.val = val;

 \*     this.next = null;

 \* }

 \*/

/\*\*

 \* @param {ListNode} head

 \* @return {ListNode}

 \*/

var deleteDuplicates = function(head) {

    const dummy = new ListNode();

    dummy.next = head;

    let node = dummy;                                                  // the last known distinct node

    while (node.next) {

        if (node.next.next && node.next.val === node.next.next.val) {  // if the next two nodes are equal...

            let nonValNode = node.next.next.next;

            while (nonValNode && nonValNode.val === node.next.val) {   // ...find the first one that isn't...

                nonValNode = nonValNode.next;

            }

            node.next = nonValNode;                                    // ...and glue it to the last known distinct node;...

        } else {

            node = node.next;                                          // ...otherwise the next node is distinct

        }

    }

    return dummy.next;

};

[Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree)

var isValidBST = function(root, min, max){

    return isValidBSTHelper(root, -Infinity, Infinity);

};

let isValidBSTHelper = function(root, min, max){

    if(root === null) return true;

    if(root.val <= min || root.val >= max) return false;

    return isValidBSTHelper(root.left, min, root.val) && isValidBSTHelper(root.right, root.val, max);

}

**Binary Tree Level Order Traversal**

var levelOrder = function(root) {

    let result = [];

    currentLevelNodes = [];

    if(root)

        currentLevelNodes.push(root);

    while(currentLevelNodes.length > 0) {

        current = [];

        let len = currentLevelNodes.length;

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

            let node = currentLevelNodes.shift();

            current.push(node.val);

            if(node.left) {

                currentLevelNodes.push(node.left);

            }

            if(node.right) {

                currentLevelNodes.push(node.right);

            }

        }

        result.push(current);

    }

    return result;

};

**Binary Tree Zigzag Level Order Traversal**

var zigzagLevelOrder = function(root) {

    let res = [];

    helper(root, 0, res);

    return res;

};

var helper = function(node, level, res){

    if(!node) return;

    if(!res[level]) res[level] = [];

    level % 2 ? res[level].unshift(node.val) : res[level].push(node.val);

    helper(node.left, level + 1, res);

    helper(node.right, level + 1, res);

}

**Maximum Depth of Binary Tree**

var maxDepth = function(root) {

    if(root === undefined || root===null){

        return 0;

    }

    return Math.max(maxDepth(root.left),maxDepth(root.right)) + 1;

};

[Construct Binary Tree from Preorder and Inorder Traversal](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal)

var buildTree = function(preorder, inorder) {

    function helper(p1, p2, i1, i2) {

        if (p1 > p2 || i1 > i2) return null; // sanity check

        var value = preorder[p1],           // get the root value

            index = inorder.indexOf(value), // get inorder position

            nLeft = index - i1,             // count nodes in left subtree

            root  = new TreeNode(value);    // build the root node

        // build the left and right subtrees recursively

        root.left  = helper(p1 + 1, p1 + nLeft, i1, index - 1);

        root.right = helper(p1 + nLeft + 1, p2, index + 1, i2);

        return root;

    }

    return helper(0, preorder.length - 1, 0, inorder.length - 1);

};

**Convert Sorted Array to Binary Search Tree**

/\*\*

 \* Definition for a binary tree node.

 \* function TreeNode(val) {

 \*     this.val = val;

 \*     this.left = this.right = null;

 \* }

 \*/

/\*\*

 \* @param {number[]} nums

 \* @return {TreeNode}

 \*/

var sortedArrayToBST = function(nums) {

    if (!nums.length) return null;

    const mid = Math.floor(nums.length / 2);

    const root = new TreeNode(nums[mid]);

    // subtrees are BSTs as well

    root.left = sortedArrayToBST(nums.slice(0, mid));

    root.right = sortedArrayToBST(nums.slice(mid + 1));

    return root;

};

**Minimum Depth of Binary Tree**

/\*\*

 \* Definition for a binary tree node.

 \* function TreeNode(val) {

 \*     this.val = val;

 \*     this.left = this.right = null;

 \* }

 \*/

/\*\*

 \* @param {TreeNode} root

 \* @return {number}

 \*/

var minDepth = function(root) {

    if(!root) return 0;

    var result;

    function minHeight(root, depth){

        if(!root.left && !root.right){

            result = Math.min(result || depth, depth)

        }

        if(root.left) minHeight(root.left, depth + 1);

        if(root.right) minHeight(root.right, depth + 1);

    }

    minHeight(root, 1);

    return result;

};

**Path Sum for binary tree**

var hasPathSum = function(root, sum) {

    var a = false,

        b = false;

    if(root === null)

        return false;

    sum -= root.val;

    if(sum === 0 && root.left === null && root.right === null)

        return true;

    if(root.left !== null)

        a = hasPathSum(root.left, sum);

    if(root.right !== null)

        b = hasPathSum(root.right, sum);

    return a || b;

};

**House Robber**

var rob = function(nums) {

    solution = {};

    solution[nums.length] = 0;

    solution[nums.length-1] = nums[nums.length-1];

    for (let i=nums.length-2; i>=0; i--) {

        solution[i] = Math.max(nums[i] + solution[i + 2], solution[i+1]);

    }

    return solution[0];

};

**Word Break**

const wordBreak = (s, wordDict) => {

    return verify(s, wordDict, 0, [])

};

const verify = (word, dict, start, memo) => {

    if(start === word.length) return true

    if(memo[start] !== undefined ) return memo[start]

    for (let end = start +1; end <= word.length; end++) {

        let subStr= word.substring(start, end)

        if(dict.includes(subStr) && verify(word, dict, end, memo)) {

            return memo[start] = true

        }

    }

    return memo[start] = false

}

// new - The bug fixed.

var wordBreak = function(s, wordDict) {

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

s = s.split(wordDict[i]).join('.');

}

return s.split('.').join('').length === 0;

};

**Minimum Size Subarray Sum**

function minSubArrayLen(s, nums) {

    var min = Number.MAX\_VALUE;

    // boundaries

    var l = 0;

    var r = -1;

    // current sum

    var sum = 0;

    while (r < nums.length) {

        if (sum >= s) {

            min = Math.min(min, r - l + 1);

            sum -= nums[l];

            l++;

        } else {

            r++;

            sum += nums[r];

        }

    }

    return min === Number.MAX\_VALUE ? 0 : min;

}

**Number of Islands**

var numIslands = function(grid) {

    let count = 0;

    let h = grid.length;

    let w = h && grid[0].length;

    for(let i = 0; i < h; i ++){

        for(let j = 0; j < w; j ++){

            if(grid[i][j] === '0') continue;

            count ++;

            dfs(i, j);

        }

    }

    return count;

    function dfs(n, m){

        if(n < 0 || m < 0 || n >= h || m >= w) return;

        if(grid[n][m] === '1'){

            grid[n][m] = '0';

            dfs(n + 1, m);

            dfs(n - 1, m);

            dfs(n, m + 1);

            dfs(n, m - 1);

        }

    }

};

**Move Zeroes inplace**

function moveZeroes(nums) {

    var idx = 0;

    for (var i = 0; i < nums.length; i++) {

      if (nums[i] !== 0) {

        nums[idx] = nums[i];

        nums[i] = idx === i ? nums[i] : 0;

        idx++;

      }

    }

  }

**Longest Increasing Subsequence**

function lengthOfLIS(nums) {

    var lis = [];

    for (var i = 0; i < nums.length; i++) {

        lis.push(1);

        for (var j = 0; j < i; j++) {

            if (nums[j] < nums[i]) lis[i] = Math.max(lis[i], lis[j] + 1);

        }

    }

    return nums.length ? Math.max.apply(null, lis) : 0;

}

**Coin Change**

var coinChange = function(coins, amount) {

    var dp=new Array(amount+1);

    dp.fill(Number.MAX\_VALUE-1);

    dp[0]=0;

    for(var i=1;i<=amount;i++)

    {

        for(var j=0;j<coins.length;j++)if(coins[j]<=i){

            dp[i]=Math.min(dp[i-coins[j]]+1, dp[i]);

        }

    }

       return dp[amount]==Number.MAX\_VALUE-1 ? -1:dp[amount];

};

**Top K Frequent Elements**

const topKFrequent = (nums, k) => {

    const map = {};

    for (const n of nums) {

      if (map[n] == null) map[n] = 0;

      map[n]++;

    }

    const arr = [];

    for (const n in map) {

      arr.push({ n, count: map[n] });

    }

    return arr

      .sort((a, b) => b.count - a.count)

      .slice(0, k)

      .map(a => Number(a.n));

  };

**Is Subsequence**

var isSubsequence = function(s, t) {

    let newS = s;

    for (let char of t) {

        if (newS[0] === char) {

            newS = newS.slice(1);

        }

    }

    return !newS.length;

};

**Merge Two Binary Trees**

var mergeTrees = function(t1, t2) {

    if (!t1 && !t2) {

        return null;

    }

    if (!t1 || !t2) {

        return t1 || t2;

    }

    var root = new TreeNode(t1.val + t2.val);

    root.left = mergeTrees(t1.left, t2.left);

    root.right = mergeTrees(t1.right, t2.right);

    return root;

};

Max Area of Island

const maxAreaOfIsland = (grid) => {

    const res = { count: 0 }

    for (let r = 0; r < grid.length; r++) {

        for (let c = 0; c < grid[0].length; c++) {

            dfs(grid, r, c, res)

        }

    }

    return res.count

};

const dfs = (grid, r, c, res, area = { count: 0 }) => {

    if (!grid[r] || !grid[r][c]) return

    res.count = Math.max(res.count, area.count += grid[r][c])

    grid[r][c] = 0

    dfs(grid, r, c - 1, res, area)

    dfs(grid, r, c + 1, res, area)

    dfs(grid, r - 1, c, res, area)

    dfs(grid, r + 1, c, res, area)

};

**Kth Largest Element in a Stream**

class KthLargest {

    /\*\*

     \* @param {number} k

     \* @param {number[]} nums

     \*/

    constructor(k, nums) {

        this.pq = new PriorityQueue({ initialValues: nums });

        this.k = k;

        while (this.pq.size() > this.k) {

            this.pq.poll();

        }

    }

    /\*\*

     \* @param {number} val

     \* @return {number}

     \*/

    add(val) {

        if (this.pq.size() < this.k) {

            this.pq.offer(val);

        } else if (val > this.pq.peek()) {

            this.pq.poll();

            this.pq.offer(val);

        }

        return this.pq.peek();

    }

}

**Capacity To Ship Packages Within D Days**

var shipWithinDays = function(weights, D) {

    if (!weights || weights.length === 0 || D === 0) {

        return 0;

    }

    // The weights is actually sorted sequence.

    // The minimum capacity would be the minimum weight

    // we can start with that.

    // In naive approach, start with minium weight as your capacity

    // go through the weights, when you exceed capacity, increment the days since you can only

    // deliver max weight = capacity.

    // If at the end of weights, your total days exceeded D, that means your capacity was less, and hence increase capacity.

    // Pick first item from left and add to capacity

    // Then repeat the previous process.

    // When you meet the days === D criteria, your capacity is the answer

    // What you did earlier was a linear search, since this is a sorted sequence (not literally but sequence needs to be processed in the same order)

    // We can do the same thing with binary search. Keep dividing the weights by 2 to get the capacity value instead of increasing it one element at a time.

    let totalWeight = 0;

    let maxWeight = weights[0];

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

        if (weights[i] > maxWeight) {

            maxWeight = weights[i];

        }

        totalWeight += weights[i];

    }

    let start = maxWeight;

    let end = totalWeight;

    while (start < end) {

        const mid = start + Math.floor((end - start) / 2);

        // this is capacity we are trying for.

        let numberOfDaysNeeded = 1;

        let currentDayWeightTotal = 0;

        // Now go through all the weights

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

            const weight = weights[i];

            if (weight + currentDayWeightTotal > mid) {

                // current day weight becomes more than our capacity, so increment the days,

                // reset current day sum

                numberOfDaysNeeded++;

                currentDayWeightTotal = 0;

            }

            currentDayWeightTotal += weight;

        }

        if (numberOfDaysNeeded > D) {

            // lets move right

            start = mid + 1;

        } else {

            // lets move left

            end = mid;

        }

    }

    return start;

};

Merge sorted array (m+n,n)

var merge = function (nums1, m, nums2, n) {

    var len = m + n;

    m--;

    n--;

    while (len--) {

        if (n < 0 || nums1[m] > nums2[n]) {

            nums1[len] = nums1[m--];

        } else {

            nums1[len] = nums2[n--];

        }

    }

};