# Longest Palindromic Substring.

# Medium

**Example 1:**

**Input:** s = "babad"

**Output:** "bab"

**Explanation:** "aba" is also a valid answer.

var longestPalindrome = function(s) {

let ans = ""

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

let leftIndex = i - 1

let rightIndex = i + 1

while (s[i] == s[rightIndex]) rightIndex ++ // deal with duplicate letters in the center

if (rightIndex - i > ans.length) ans = s.substring(i, rightIndex)

while (leftIndex >= 0 && rightIndex < s.length) {

if (s[leftIndex] == s[rightIndex]) {

if (rightIndex - leftIndex + 1 > ans.length) {

ans = s.substring(leftIndex, rightIndex + 1)

}

leftIndex --

rightIndex ++

}

else break

}

}

return ans

};

# Reverse Integer

# Medium

**Example 1:**

**Input:** x = 123

**Output:** 321

**Example 2:**

**Input:** x = -123

**Output:** -321

**Example 3:**

**Input:** x = 120

**Output:** 21

const reverse = function (x) {

const MAX\_INT = Math.pow(2, 31) - 1;

const MIN\_INT = -(Math.pow(2, 31));

let rev = 0;

while (x !== 0) {

const pop = x % 10;

x = x > 0 ? Math.floor(x / 10) : Math.ceil(x / 10);

rev = rev \* 10 + pop;

}

return (rev < MAX\_INT && rev > MIN\_INT) ? rev : 0 ;

};

# 3.Median of Two Sorted Arrays

# Hard

**Example 1:**

**Input:** nums1 = [1,3], nums2 = [2]

**Output:** 2.00000

**Explanation:** merged array = [1,2,3] and median is 2.

**Example 2:**

**Input:** nums1 = [1,2], nums2 = [3,4]

**Output:** 2.50000

**Explanation:** merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

var findMedianSortedArrays = function(nums1, nums2) {

let temp = [...nums1, ...nums2].sort((a, b) => a - b);

let result = 0;

if (temp.length % 2 == 0) {

let a = temp[(temp.length / 2) - 1];

let b = temp[temp.length / 2]

result = (a + b) / 2;

} else {

result = temp[Math.floor(temp.length/2)];

}

return result;

};

# 4.Zigzag Conversion

# Medium

**Example 1:**

**Input:** s = "PAYPALISHIRING", numRows = 3

**Output:** "PAHNAPLSIIGYIR"

var convert = function(s, numRows) {

if(numRows === 1) {

return s;

}

const stringLength = s.length;

const output = Array(numRows).fill('');

let addToRow = 0;

let add = 1;

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

output[addToRow] += s[i];

// If row is on top of the zigzag we will sum up +1 to the next rows

if (addToRow === 0) {

add = 1;

}

// If row is on bottom of the zigzag we will subtract -1 to the next rows

if (addToRow === (numRows -1)) {

add = -1;

}

// We just keep track of the current row to add the string e.g(numRows = 2) 0,1,2,1,0,1,2

addToRow+= add;

}

return output.join('');

};

# 5.Remove Duplicates from Sorted Array

# Easy

**Example 1:**

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2,\_]

var removeDuplicates = function(nums) {

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

if(i+1<nums.length){

if(nums[i]===nums[i+1]){

nums.splice(i,1)

i--;

}

}

}

return nums.length

};

# 6. N-Queens

# Hard

**Input:** n = 4

**Output:** [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]

**Explanation:** There exist two distinct solutions to the 4-queens puzzle as shown above

const putI = (i, n) => {

const arr = new Array(n).fill('.');

arr[i] = 'Q';

return arr.join('');

};

const findNext = (n, board) => {

const nextOptions = [];

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

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

if (

!board.some((row, rowIndex) => {

if (rowIndex === i) {

return true

}

const colIndex = row.indexOf('Q');

return (

colIndex === j ||

Math.abs(rowIndex - i) === Math.abs(colIndex - j)

)

})

) {

nextOptions.push(putI(j, n));

}

}

}

return nextOptions;

};

const solveNQueens = (n, solution = [], solutions = []) => {

if (solution.length === n) {

return [...solutions, solution];

}

const nextOptions = findNext(n, solution);

if (!nextOptions.length) {

return solutions;

}

return nextOptions.map((next) => solveNQueens(n, [...solution, next], solutions)).flat();

};

# 7.Longest Valid Parentheses

# Hard

**Example 1:**

**Input:** s = "(()"

**Output:** 2

**Explanation:** The longest valid parentheses substring is "()".

var longestValidParentheses = function(s) {

if (!s || !s.length) { return 0; }

/\* We will store the position of every invalid parenthesis.

Once we have that, the solution is simply the longest

subarray between two invalid parentheses \*/

const invalids = new Set();

/\* We stack the opening parentheses as we find them,

and pop them we we meet the corresponding closing

parenthesis. Note that a closing ) always matches the

latest opening ( one, hence the choice of a stack \*/

const stack = [];

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

if (s[i] === '(') {

stack.push(i);

} else {

// If we are closing an opening parenthesis, pop it out

if (stack.length) {

stack.pop();

} else {

/\* Otherwise there is nothing to close,

hence this parenthesis is invalid \*/

invalids.add(i);

}

}

}

/\* Any remaining opening parenthesis that has not been closed is

automatically invalid \*/

while (stack.length) {

invalids.add(stack.pop());

}

// Here we just count how many valid in between every invalid

let max = 0, count = 0;

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

if (i < s.length && !invalids.has(i)) {

count++;

} else {

max = Math.max(max, count);

count = 0;

}

}

return max;

};

# 8. 3Sum

# Medium

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

function findTriplets(nums,start,target,result) {

let end = nums.length-1;

while(start < end) {

if(nums[start]+nums[end]===target) {

result.push([-target,nums[start],nums[end]]);

while(nums[start]===nums[start+1] && start < end) {

start++;

}

while(nums[end]===nums[end-1] && start < end) {

end--;

}

start++;

end--;

} else if(nums[start]+nums[end] > target) {

end--;

} else {

start++;

}

}

}

var threeSum = function(nums) {

nums.sort((a,b)=>a-b);

let i=0,result=[];

while(i<nums.length) {

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

i++;

} else {

findTriplets(nums,i+1,0-nums[i],result);

i++;

}

}

return result;

};

# 9.Two sum(Easy)

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

var twoSum = function(nums, target) {

number=[]

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

{

for(j=i+1;j<nums.length;j++)

{

sum=nums[i]+nums[j]

if(sum==target)

{

number.push(i)

number.push(j)

return number

}

}

}

};

# 10.Longest Substring Without Repeating Characters

var lengthOfLongestSubstring = function(s) {

let leftIdx = 0;

let max = 0;

const hash = {};

if (s.length < 2) return s.length;

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

const letter = s.charAt(i);

while (hash[letter]) {

delete hash[s.charAt(leftIdx)];

leftIdx++;

}

hash[letter] = true;

max = Math.max(max, i - leftIdx + 1);

}

return max;

};

# 11.Palindrome number

var isPalindrome = function(x) {

return x.toString() === x.toString().split('').reverse().join('');

};

# 12.Longest Common Prefix

var longestCommonPrefix = function(strs) {

if (strs.length === 1) {

return strs[0]

}

let returnStr = ""

let counter = 0;

while (counter < strs[0].length) {

let filterArr = strs.filter((word) => {

return word[counter] === strs[0][counter]

})

if (filterArr.length !== strs.length) {

return returnStr;

} else {

returnStr += strs[0][counter]

counter++;

}

}

return returnStr

};

# 13.valid parenthesis

var isValid = function(s) {

let stack = []

let counter = 0

let parens = {

"}": "{",

"]": "[",

")": "("

}

while(counter < s.length) {

let current = s[counter];

if(parens[current]) {

let top = stack.pop()

if(top != parens[current]) {

return false;

}

} else {

stack.push(current)

}

counter++

}

return stack.length == 0;

};

# 14.Implement strStr()

var strStr = function(haystack, needle) {

let splitedHaystack = haystack.split(needle);

if(needle === ""){

return 0;

} else if(splitedHaystack.length > 1){

return splitedHaystack[0].length;

} else {

return -1;

}

};

# 15.Length of Last Word

var lengthOfLastWord = function(s) {

let words = s.trim().split(" ");

return words[words.length - 1].length;

};

# 16.Maximum Subarray

var maxSubArray = function(nums) {

let current\_maxSum = nums[0], maxSum = nums[0]

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

if(current\_maxSum < 0) current\_maxSum = nums[i]

else current\_maxSum += nums[i]

maxSum = Math.max(current\_maxSum, maxSum)

}

return maxSum

}

# 17.Divide Two Integers

var divide = function(dividend, divisor) {

if(divisor == -1 && dividend < 1 && dividend != -1) return (dividend/divisor)-1

return parseInt(dividend/divisor)

};

# 18.Count Primes

var countPrimes = function(n) {

let primes = new Array(n).fill(true);

for(let i=2;i\*i<primes.length;i++) {

if(primes[i]) {

for(let j=i;j\*i<primes.length;j++){

primes[i\*j] = false;

}

}

}

let count = 0;

for(let i=2;i<primes.length;i++){

if(primes[i]){

count++;

}

}

return count;

};