# 1. Longest Palindromic Substring

**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

};

# 2. Reverse Integer

**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

**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

**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

**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

**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

**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;

};