# **Disclaimer:**

COPYRIGHT NOTICE

©All content, materials, and intellectual property contained within this document, including but not limited to text, images, diagrams, graphics, code, and any other original works, are protected by copyright laws and international treaties. This intellectual property is the sole and exclusive property of SRIKANTH TEKUMUDI

Unauthorized use, reproduction, distribution, modification, or transmission of any portion of this document, in any form or by any means, electronic or mechanical, without the prior written permission of SRIKANTH TEKUMUDI, is strictly prohibited and may result in severe civil and criminal penalties.

By accessing, viewing, or using this document, you acknowledge and agree to abide by all applicable copyright laws and the terms and conditions outlined herein. This copyright notice serves as a legal warning and reminder that the unauthorized use of this intellectual property is strictly prohibited and will be vigorously enforced.

Thank you for respecting the intellectual property rights of SRIKANTH TEKUMUDI

# **DSA for frontend**

## Patterns in DSA

1. **Two Pointer**
   * **Pattern**: Utilize two pointers to iterate through an array or list from different ends or with different speeds.
   * **Common Problems**: Linked List Cycle, Two Sum II, Container with Most Water, 3Sum.
2. **Binary Tree BFS**
   * **Pattern**: Use a queue to traverse the tree level by level.
   * **Common Problems**: Binary Tree Level Order Traversal, Minimum Depth of Binary Tree, Binary Tree Zigzag Level Order Traversal.
3. **Topological Sort**
   * **Pattern**: Use DFS or Kahn’s algorithm to order tasks that depend on other tasks.
   * **Common Problems**: Course Schedule, Course Schedule II, Alien Dictionary.
4. **Binary Tree DFS**
   * **Pattern**: Use recursion or an explicit stack to traverse the tree pre-order, in-order, or post-order.
   * **Common Problems**: Binary Tree Inorder Traversal, Path Sum, Maximum Depth of Binary Tree.
5. **Top K Elements**
   * **Pattern**: Use a heap or quickselect algorithm to find the top K elements.
   * **Common Problems**: Kth Largest Element in an Array, Top K Frequent Elements, K Closest Points to Origin.
6. **Modified Binary Search**
   * **Pattern**: Adjust binary search to solve problems beyond simple searching, often in a rotated or unsorted array.
   * **Common Problems**: Search in Rotated Sorted Array, Find Peak Element, Search a 2D Matrix.
7. **Subset**
   * **Pattern**: Use backtracking or bit manipulation to generate all subsets.
   * **Common Problems**: Subsets, Subsets II, Combination Sum.
8. **Sliding Window**
   * **Pattern**: Use two pointers or a deque to maintain a window of elements that satisfies a certain condition.
   * **Common Problems**: Longest Substring Without Repeating Characters, Minimum Window Substring, Sliding Window Maximum.
9. **Monotonic Stack**
   * **Pattern**: Use a stack to maintain elements in a specific monotonic order (increasing or decreasing) to solve problems related to previous/next greater/smaller elements.
   * **Common Problems**: Largest Rectangle in Histogram, Trapping Rain Water, Daily Temperatures.
10. **Palindromic Subsequence**
    * **Pattern**: Use dynamic programming to find the longest palindromic subsequence within a string.
    * **Common Problems**: Longest Palindromic Subsequence, Count Different Palindromic Subsequences.
11. **Longest Common Substring**
    * **Pattern**: Use dynamic programming or suffix arrays to find the longest substring common to two or more strings.
    * **Common Problems**: Longest Common Substring, Longest Common Subsequence.
12. **Island Matrix Traversal**
    * **Pattern**: Use DFS or BFS to traverse a grid, typically to count the number of islands or connected components.
    * **Common Problems**: Number of Islands, Max Area of Island, Surrounded Regions.
13. **Prefix Sum**
    * **Pattern**: Use a running total to calculate cumulative sums which can be used to quickly find the sum of any subarray.
    * **Common Problems**: Subarray Sum Equals K, Continuous Subarray Sum, Range Sum Query - Immutable.

**14 . Postfix Sum**

* **Pattern**: Similar to prefix sum, but accumulate sums from the end of the array to the beginning.
* **Common Problems**: Product of Array Except Self (combined with prefix sum), Minimum Number of Operations to Move All Balls to Each Box.

**15 .Greedy Algorithm**

* **Pattern**: Make a series of choices, each of which looks best at the moment, with the hope that the global optimum will be reached.
* **Common Problems**: Interval Scheduling Maximization, Jump Game, Coin Change (minimum number of coins), Minimum Cost to Connect Sticks.

## Array based problem

### Rotate array

| function rotateArrayRight(arr, k) {  *// Ensure k is within the bounds of the array length*  k = k % arr.length;   *// If k is 0, no need to rotate*  if (k === 0) {  return arr;  }   *// Split and rearrange the array*  return arr.slice(-k).concat(arr.slice(0, -k)); }  *// Example usage:* const arr = [1, 2, 3, 4, 5]; const k = 2; const rotatedArr = rotateArrayRight(arr, k); console.log(rotatedArr); *// Output: [4, 5, 1, 2, 3]* |
| --- |

### Set Matrix Zeroes

<https://leetcode.com/problems/set-matrix-zeroes/description/>

| var setZeroes = function(matrix) {     let rowLen=matrix.length;  let colLen=matrix[0].length    let col0=1;   for(let row=0;row<rowLen;row++){  for(let col=0;col<colLen;col++){  if(matrix[row][col]==0){    matrix[row][0]=0;  if(col==0){  col0=0;    }  else{  matrix[0][col]=0;  }  }  }  }   for(let row=1;row<rowLen;row++){  for(let col=1;col<colLen;col++){  if(matrix[row][col]!=0){  if(matrix[0][col]==0 || matrix[row][0]==0){  matrix[row][col]=0  }   }  }  }    *//if 0 0 is 0 then 1st row should be zero*  if(matrix[0][0]==0){  for(let col=0;col<colLen;col++){  matrix[0][col]=0  }  }    *// it represents first coloumn*  if(col0==0){  for(let row=0;row<rowLen;row++){  matrix[row][0]=0  }  }   };  // Test Case  let matrix = [  [1, 2, 3],  [4, 0, 6],  [7, 8, 9]  ];  setZeroes(matrix);  console.log(matrix);  // Expected Output:  // [  // [1, 0, 3],  // [0, 0, 0],  // [7, 0, 9]  // ] |
| --- |

### **Pascal's Triangle**

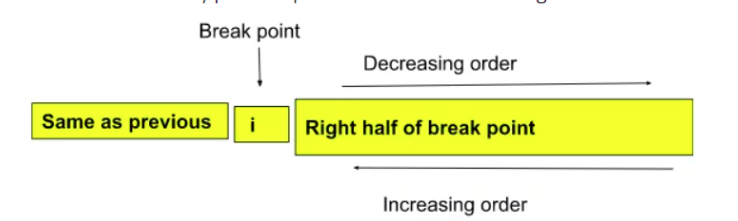
<https://leetcode.com/problems/pascals-triangle/>

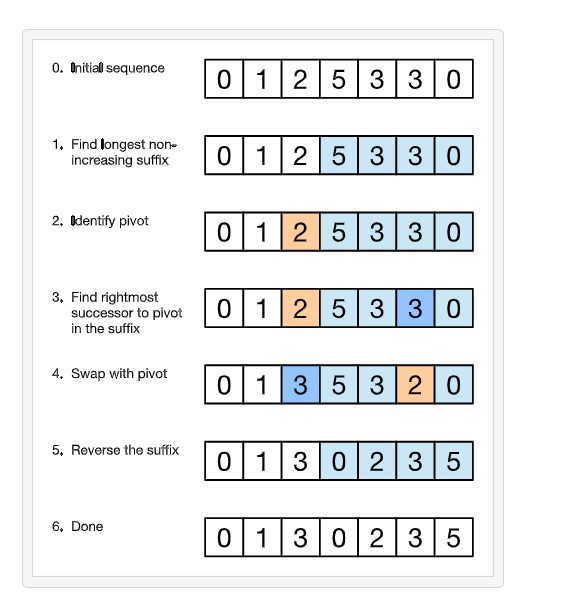
| function generateRow(n){   let ans=1  let row=[1]  for(let r=0;r<n;r++){  ans=ans\*(n-r)  ans=ans/(r+1)  row.push(ans)  }   return row }  var generate = function(numRows) {  let result=[]  let n=numRows  for(let i=0;i<numRows;i++){  let row=generateRow(i)  result.push(row)  }   return result  };  0 1 1 1c0 1c1  2 2c0 2c1 2c2  3 3c0 3c2 3c2 3c2  5 serices 1 5/1 5\*4/1\*2 5\*4\*3/1\*2\*3 5\*4\*3\*2/1\*2\*3\*4 5\*4\*3\*2\*1\*/1\*2\*3\*4\*5  let ans=1 let res=[1] for(let r=0;r<n;r++){  ans=an\*(n-r)/(r+1) } |
| --- |

### **Next Permutation**

<https://leetcode.com/problems/next-permutation/description/>

| var nextPermutation = function(nums) {  let n = nums.length - 1   let dipindex = -1  for (right = n - 1; right >= 0; right--) {  if (nums[right] < nums[right + 1]) {  dipindex = right  break  }  }   console  .log(dipindex)   if (dipindex == -1) return nums.reverse()   *// need an number just grater that dip index*   for (let right = n; right >= 0; right--) {  if (nums[right] > nums[dipindex]) {  [nums[right], nums[dipindex]] = [nums[dipindex], nums[right]]  break  }  }   *// console.log(nums,'swapped')*   *// reverse numsarray for dipIndex+1*   let left = dipindex + 1;  let rightIndex = n   while (left < rightIndex) {  [nums[left], nums[rightIndex]] = [nums[rightIndex], nums[left]]   left++;  rightIndex--  }   *// console.log(nums)* };   let's apply this algorithm to the given sequence {2, 1, 5, 4, 3, 0, 0}:  Start from the right and find the first pair of successive numbers where the left one is smaller than the right one. In this case, it's (1, 5), and index i = 1.  Now, from the right end, find the first number greater than the number at index i, which is 3. Index j = 4.  Swap the numbers at indices 1 and 4. The sequence becomes {2, 3, 5, 4, 1, 0, 0}.  Reverse the sequence from index i + 1 onwards. That means, reverse the sequence from index 2 to the end. The resulting sequence is {2, 3, 0, 0, 1, 4, 5}.  So, the next permutation of the given sequence {2, 1, 5, 4, 3, 0, 0} is {2, 3, 0, 0, 1, 4, 5}. |
| --- |





### **Maximum Subarray**

<https://leetcode.com/problems/maximum-subarray/description/>

| var maxSubArray = function(nums) {   let maxUntill=nums[0];  let max=nums[0]   for(let i=1;i<nums.length;i++){  maxUntill=Math.max(nums[i],nums[i]+maxUntill)  max=Math.max(max,maxUntill)  }   return max   };  let nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4];  let result = maxSubArray(nums);  console.log(result); // Expected Output: 6 |
| --- |

### **Sort Colors (Dutch flags)**

<https://leetcode.com/problems/sort-colors/description/>

| var sortColors = function(nums) {    let start = 0;  let mid = 0;  let end = nums.length - 1    while (mid <= end) {  if (nums[mid] == 0) {  [nums[mid], nums[start]] = [nums[start], nums[mid]]  start += 1  mid += 1  }   *//must be else if*  else if (nums[mid] == 2) {  [nums[mid], nums[end]] = [nums[end], nums[mid]]  end -= 1   } else {  mid += 1  }   } };  // Test Case  let nums = [2, 0, 2, 1, 1, 0];  sortColors(nums);  console.log(nums); // Expected Output: [0, 0, 1, 1, 2, 2] |
| --- |

### Second largest number in array

| function getSecondLargest(arr) {  let firstLargest = -Infinity;  let secondLargest = Infinity;   for (let ele of arr) {  if (ele > firstLargest) {  secondLargest = firstLargest;  firstLargest = ele;  } else if (ele > secondLargest && ele != firstLargest) {  secondLargest = ele;  }  }   return secondLargest; }  let arr = [12, 34, 34, 23, 232, -10, -38, 45, 230];  let res = getSecondLargest(arr); console.log(res); |
| --- |

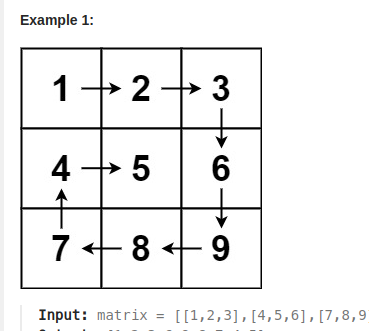
### Sort 0 and 1’s

function sortZerosOnes(arr){

| let left=0  let n=arr.length    for(let i=0;i<n;i++){  if(arr[i]==0){  [arr[i],arr[left]]=[arr[left],arr[i]];  left+=1  }    }    console.log(arr)   }  sortZerosOnes([0,1,1,0,0,1,1,0]) |
| --- |

### Spiral traversal matrix

| */\*\*  \* @param {number[][]} matrix  \* @return {number[]}  \*/* var spiralOrder = function(matrix) {   let top=0;  let bottom=matrix.length-1;  let left=0;  let right=matrix[0].length-1;   let res=[]    while(top<=bottom && left<=right){    for(let i=left;i<=right;i++){  res.push(matrix[top][i])  }   top++;   for(let i=top;i<=bottom;i++){  res.push(matrix[i][right])  }   right--;   if(top<=bottom){  for(let i=right;i>=left;i--){  res.push(matrix[bottom][i])  }  bottom--  }    if(left<=right){  for(let i=bottom;i>=top;i--){  res.push(matrix[i][left])    }   left++  }     }   return res     };  top bottom => indicates row left right =>indicate colo  let matrix1 = [  [1, 2, 3],  [4, 5, 6],  [7, 8, 9]  ];  let result1 = spiralOrder(matrix1);  console.log(result1); // Expected Output: [1, 2, 3, 6, 9, 8, 7, 4, 5] |
| --- |



### **Best Time to Buy and Sell Stock**

<https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/>

| var maxProfit = function(prices) {   let minPrice=prices[0];  let maxi=0;   for(let i=0;i<prices.length;i++){  minPrice=Math.min(minPrice,prices[i]);  maxi=Math.max(maxi,prices[i]-minPrice)  }   return maxi }  let prices1 = [1, 2, 3, 4, 5];  let result1 = maxProfit(prices1);  console.log(result1); // Expected Output: 4 |
| --- |

### **Merge Intervals**

<https://leetcode.com/problems/merge-intervals/description/>

| var merge = function(intervals) {   intervals.sort((a, b) => a[0] - b[0])   let res = [intervals[0]]  let prev = intervals[0]  let start = 0;  let end = 1  for (let current of intervals.slice(1)) {   if (current[start] <= res.at(-1)[end]) {   res.at(-1)[end] = Math.max(current[end], res.at(-1)[end])  } else {  res.push(current)  }   }   return res  };  let intervals1 = [[1, 3], [2, 6], [8, 10], [15, 18]];  let result1 = merge(intervals1);  console.log(result1); *// Expected Output: [[1, 6], [8, 10], [15, 18]]* |
| --- |

### **Count Inversions**

<https://www.codingninjas.com/studio/problems/count-inversions_615?leftPanelTabValue=PROBLEM>

| from os import \* from sys import \* from collections import \* from math import \*  count=0 def merge(arr,low,mid,high):  global count   temp=[]  left=low;  right=mid+1   while(left<=mid and right<=high):  if(arr[left]<=arr[right]):  temp.append(arr[left]);  left+=1;  else:  count+=mid-left+1  temp.append(arr[right]);  right+=1;      while(left<=mid):  temp.append(arr[left]);  left+=1;    while(right<=high):  temp.append(arr[right])  right+=1    for i in range(low,high+1):  arr[i]=temp[i-low]    # print(arr)      def mergeSort(arr,start,end):  if(start>=end):  return   mid=(start+end)*//2*  mergeSort(arr,start,mid)  mergeSort(arr,mid+1,end)  merge(arr,start,mid,end)   def getInversions(arr, n) :  # Write your code here.  global count   count=0  start=0;  end=n-1;  mergeSort(arr,start,end)  # print(arr)  return count     # Taking inpit using fast I/O. def takeInput() :  n = int(input())  arr = list(map(int, stdin.readline().strip().split(" ")))  return arr, n  # Main. arr, n = takeInput() print(getInversions(arr, n)) |
| --- |

### **Merge Sorted Array**

<https://leetcode.com/problems/merge-sorted-array/description/>

| */\*\*  \* @param {number[]} nums1  \* @param {number} m  \* @param {number[]} nums2  \* @param {number} n  \* @return {void} Do not return anything, modify nums1 in-place instead.  \*/* var merge = function (nums1, m, nums2, n) {  *// [1,2,3,0,0,0]*  *// [2,5,6]*   let l = m + n - 1;  m -= 1;  n -= 1;   while (l >= 0) {  if (n < 0 || nums1[m] > nums2[n]) {  nums1[l] = nums1[m--];  } else {  nums1[l] = nums2[n--];  }  l -= 1;  } };  let nums1 = [1, 2, 3, 0, 0, 0]; let m = 3; let nums2 = [2, 5, 6]; let n = 3;  merge(nums1, m, nums2, n);  console.log(nums1); *// Expected Output: [1, 2, 2, 3, 5, 6]* |
| --- |

### **Find the Duplicate Number**

<https://leetcode.com/problems/find-the-duplicate-number/description/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/* var findDuplicate = function (a) {  *// let sum=nums.reduce((a,b)=>a+b,0)*  *// let real\_sum=((nums.length)\*(nums.length-1))/2*  *// return Math.abs(real\_sum-sum)*   let table = {};  let n = a.length;  for (let i = 0; i < n; i++) {  if (a[i] in table) return a[i];  else table[a[i]] = "hi";  } };  let nums1 = [1, 3, 4, 2, 2]; let result1 = findDuplicate(nums1); console.log(result1); *// Expected Output: 2* |
| --- |

### **Rotate Image**

[https://leetcode.com/problems/rotate-image/description](https://leetcode.com/problems/rotate-image/description/)

| var rotate = function(matrix) {   for (let i = 0; i < matrix.length; i++) {  for (let j = 0; j < matrix[0].length; j++) {   if (i > j)  [matrix[i][j], matrix[j][i]] = [matrix[j][i], matrix[i][j]]   }  }    for (let row = 0; row < matrix.length; row++) {    let colStart = 0;  let colEnd = matrix[0].length - 1  while (colStart < colEnd) {  [matrix[row][colStart], matrix[row][colEnd]] = [matrix[row][colEnd], matrix[row][colStart]]  colStart += 1;  colEnd -= 1   }  }  *// console.log(matrix)*  };  let matrix = [  [1, 2, 3],  [4, 5, 6],  [7, 8, 9] ];  rotate(matrix);  console.log(matrix); [  [7, 4, 1],  [8, 5, 2],  [9, 6, 3] ] |
| --- |

### **Max Consecutive Ones**

<https://leetcode.com/problems/max-consecutive-ones/description/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/* var findMaxConsecutiveOnes = function(nums) {   let count=0  let max=0  for(let i=0;i<nums.length;i++){  if(nums[i]==1){  count+=1   }  else{  count=0  }  max=Math.max(max,count)   }   return max   };   *// Test Case: Typical Case* let nums1 = [1, 1, 0, 1, 1, 1]; let result1 = findMaxConsecutiveOnes(nums1); console.log(result1); *// Expected Output: 3* |
| --- |

### **Trapping Rain Water**

<https://leetcode.com/problems/trapping-rain-water/description/>

| */\*\*  \* @param {number[]} height  \* @return {number}  \*/* var trap = function (heights) {  let n = heights.length;  let rightMax = new Array(n).fill(0);  let leftMax1 = new Array(n).fill(0);   leftMax1[0] = heights[0];   for (let i = 1; i < n; i++) {  leftMax1[i] = Math.max(leftMax1[i - 1], heights[i]);  }   rightMax[n - 1] = heights[n - 1];   for (let i = n - 2; i >= 0; i--) {  rightMax[i] = Math.max(rightMax[i + 1], heights[i]);  }   let res = new Array(n);  let ans = 0;   for (let i = 0; i < n; i++) {  res[i] = Math.min(rightMax[i], leftMax1[i]) - heights[i];  ans += res[i];  }   return ans; };  *// Test Case: Simple Case* let heights1 = [0,1,0,2,1,0,1,3,2,1,2,1]; let result1 = trap(heights1); console.log(result1); *// Expected Output: 6* |
| --- |

### **Valid Triangle Number**

<https://leetcode.com/problems/valid-triangle-number/description/>

| var triangleNumber = function (nums) {  nums.sort((a, b) => a - b);  let count = 0;  for (let i = nums.length - 1; i >= 2; i--) {  let left = 0;  let right = i - 1;  while (left < right) {  if (nums[left] + nums[right] > nums[i]) {  count = count + (right - left);  right--;  } else left++;  }  }   *// console.log(count)*  return count; }; |
| --- |

## Mono stack

### **503. Next Greater Element II**

<https://leetcode.com/problems/next-greater-element-ii/description/>

f

| or next Greater element we are traversing from left to right, so in stack it always conatins it privious elements. arr[i] check if it greater than previous element in stack, if yes we just pop them out and  and for all those arr[i] is next greater element  var nextGreaterElements = function(nums) {   let originallen=nums.length    nums=[...nums,...nums]   let n=nums.length  let dp=new Array(n).fill(-1);   let stack=[]   for(let i=0;i<n;i++){   while(stack.length && nums[i]>nums[stack.at(-1)]){  let lastele = stack.pop()  dp[lastele]=nums[i]   }  stack.push(i)  }   return dp.slice(0,originallen) };  let nums2 = [1, 2, 3, 4]; let result2 = nextGreaterElements(nums2); console.log(result2); *// Expected Output: [2, 3, 4, -1]* |
| --- |

### **Daily Temperatures**

<https://leetcode.com/problems/daily-temperatures/description/>

| */\*\*  \* @param {number[]} temperatures  \* @return {number[]}  \*/* var dailyTemperatures = function (temperatures) {  let n = temperatures.length;  let res = new Array(n).fill(0);  stack = [];   for (let i = 0; i < n; i++) {  while (stack.length > 0 && temperatures[i] > temperatures[stack.at(-1)]) {  let lastele = stack.pop();   res[lastele] = i - lastele;  }   stack.push(i);  }  return res; };  let temperatures1 = [73, 74, 75, 71, 69, 72, 76, 73]; let result1 = dailyTemperatures(temperatures1); console.log(result1); *// Expected Output: [1, 1, 4, 2, 1, 1, 0, 0]* |
| --- |

### Next Smallest

| var nextSmallest = function(heights) {   let n=heights.length  let res=new Array(n).fill(-1)  let stack=[]   for(i=0;i<n;i++){  while(stack.length && heights[i]<heights[stack.at(-1)]){  let lastEle=stack.pop()   res[lastEle]=heights[i]  }  stack.push(i)  }   console.log(res)  };  input : [73,74,75,71,69,72,76,73] ans=[  71, 71, 71, 69,  -1, -1, 73, -1 ] |
| --- |

### **Largest Rectangle in Histogram**

<https://leetcode.com/problems/largest-rectangle-in-histogram/>

| */\*\*  \* @param {number[]} heights  \* @return {number}  \*/* var largestRectangleArea = function(heights) {    heights=[-1,...heights] *//to make indexing from 1*  let n=heights.length  let nextSm=new Array(n).fill(-1)  let prevSm=new Array(n).fill(-1)  let stack=[]    *//next smallest*  for(i=1;i<n;i++){  while(stack.length && heights[i]<heights[stack.at(-1)]){  let index=stack.pop()  nextSm[index]=i-1 *// we have to take i-1 element as i is smallar than current i-1 need to pic*   }  stack.push(i)  }   while(stack.length){  let index=stack.pop()  nextSm[index]=n-1   }   *//previous smallest*  stack=[]  for(i=n-1;i>0;i--){  while(stack.length && heights[i]<heights[stack.at(-1)]){  let index=stack.pop()  prevSm[index]=i+1 *// i will be smaller than current so, need to pick i+1 as its size is atleast same as current*  }  stack.push(i)   }   while(stack.length){  let index=stack.pop()  prevSm[index]=1   }   let maxArea=0  for(let i=1;i<n;i++){  let height=heights[i];  let width=nextSm[i]-prevSm[i]+1   console.log(height,width)  maxArea=Math.max(maxArea,height\*width)  }     return maxArea   };  *// Test Case: Typical Case* let heights = [2, 1, 5, 6, 2, 3]; let result = largestRectangleArea(heights); console.log(result); *// Expected Output: 10* |
| --- |

### **Remove K Digits**

<https://leetcode.com/problems/remove-k-digits/>

| need to use monotonic increasing stack  */\*\*  \* @param {string} num  \* @param {number} k  \* @return {string}  \*/* var removeKdigits = function (num, k) {  let stack = [];  let n = num.length;   for (let i = 0; i < n; i++) {  while (k > 0 && stack.length > 0 && num[i] < stack.at(-1)) {  stack.pop();  k--;  }  stack.push(num[i]);  }   while (k > 0) {  stack.pop();  k--;  }   let smalNum = stack.join("").replace(/^0+/, "");  if (smalNum == "") return "0";  return smalNum;   return stack.join(""); };  Input: num = "1432219", k = 3   Output: "1219"  need to removve 4 as threre is dip from 4 to 3 then remove 3 as dip form 3 to 2   then remove 2 as drop form 2 to 1  so finally 1219 |
| --- |

### **Remove Duplicate Letters**

<https://leetcode.com/problems/remove-duplicate-letters/description/>

| */\*var removeDuplicateLetters = function (s) {  const lastOccur = new Map();  const stack = [];   // Record the last occurrence of each character  for (let i = 0; i < s.length; i++) {  lastOccur.set(s[i], i);  }     for (let i = 0; i < s.length; i++) {  const char = s[i];  if (stack.includes(char)) continue;   while (  stack.length > 0 &&  char < stack.at(-1) &&  i < lastOccur.get(stack.at(-1))  // to check will occur later again, remove from stack  ) {  stack.pop();  }   stack.push(char);  }   return stack.join(""); };\*/*  *//bettter version* */\*\*  \* @param {string} s  \* @return {string}  \*/* var removeDuplicateLetters = function (s) {  let lastOccur = new Map();   let stack = [];  let n = s.length;  let set = new Set();   for (let i = 0; i < n; i++) {  lastOccur.set(s[i], i);  }   for (let i = 0; i < n; i++) {  let char = s[i];  if (set.has(char)) {  continue;  }   while (  stack.length > 0 &&  char < stack.at(-1) &&  i < lastOccur.get(stack.at(-1))  ) {  let lastEle = stack.pop();  set.delete(lastEle);  }   stack.push(char);  set.add(char);  }   return stack.join(""); }; et's illustrate this with an example. Consider the string s = "cbacdcbc".  Populate lastOccur:  arduino Copy code lastOccur = { 'c': 7, 'b': 6, 'a': 2, 'd': 4 } |
| --- |

## Map data structure based problems

### Decode words

| function decode(words, message) {  *// Create a map to store the original words with their encoded keys*  const wordMap = new Map();   *// Helper function to generate a key by sorting the middle characters*  function getKey(word) {  if (word.length <= 2) return word;  const middle = word.slice(1, -1).split('').sort().join('');  return word[0] + middle + word[word.length - 1];  }   *// Populate the map with original words and their keys*  for (const word of words) {  const key = getKey(word);  wordMap.set(key, word);  }   *// Split the message into words and decode each one*  const decodedWords = message.split(' ').map(scrambledWord => {  const key = getKey(scrambledWord);  return wordMap.get(key);  });   *// Join the decoded words into a single string and return*  return decodedWords.join(' '); }  *// Example usage* const words = ['orange', 'apple', 'banana', 'grape', 'pear']; const message = 'ograen apelp bnaana gerpa paer'; console.log(decode(words, message)); *// Output: "orange apple banana grape pear"* |
| --- |

### LRU cache

| */\*\*  \* @param {number} capacity  \*/* var LRUCache = function (capacity) {  this.cache = new Map();  this.size = capacity; };  */\*\*  \* @param {number} key  \* @return {number}  \*/* LRUCache.prototype.get = function (key) {  if (!this.cache.has(key)) return -1;   let val = this.cache.get(key);   this.cache.delete(key);  this.cache.set(key, val);   return val; };  */\*\*  \* @param {number} key  \* @param {number} value  \* @return {void}  \*/* LRUCache.prototype.put = function (key, value) {  this.cache.delete(key);  this.cache.set(key, value);   *// console.log(this.cache)*  if (this.cache.size > this.size) {  const firstItem = this.cache.keys().next().value;  this.cache.delete(firstItem);  } };   class LRUCache {  */\*\*  \* @param {number} capacity  \*/*  constructor(capacity) {  this.cache = new Map();  this.capacity = capacity;  }   */\*\*  \* @param {number} key  \* @return {number}  \*/*  get(key) {  if (!this.cache.has(key)) return -1;   const value = this.cache.get(key);  this.cache.delete(key);  this.cache.set(key, value);  return value;  }   */\*\*  \* @param {number} key  \* @param {number} value  \* @return {void}  \*/*  put(key, value) {  if (this.cache.has(key)) {  this.cache.delete(key);  }   this.cache.set(key, value);   if (this.cache.size > this.capacity) {  const firstKey = this.cache.keys().next().value;  this.cache.delete(firstKey);  }  } }    *// Create a new LRUCache instance with a capacity of 2* const obj = new LRUCache(2);  *// Perform operations on the cache* obj.put(1, 1); *// Cache: {1: 1}* obj.put(2, 2); *// Cache: {1: 1, 2: 2}* console.log(obj.get(1)); *// Expected Output: 1, Cache: {2: 2, 1: 1} (1 is accessed and moved to the end)* obj.put(3, 3); *// Cache: {1: 1, 3: 3} (2 is evicted due to capacity constraint)* console.log(obj.get(2)); *// Expected Output: -1 (2 is not present in the cache)* obj.put(4, 4); *// Cache: {3: 3, 4: 4} (1 is evicted due to capacity constraint)* console.log(obj.get(1)); *// Expected Output: -1 (1 is not present in the cache)* console.log(obj.get(3)); *// Expected Output: 3, Cache: {4: 4, 3: 3} (3 is accessed and moved to the end)* console.log(obj.get(4)); *// Expected Output: 4, Cache: {3: 3, 4: 4} (4 is accessed and remains in the cache)* |
| --- |

### **Two Sum**

<https://leetcode.com/problems/two-sum/submissions/1175610905/>

| var twoSum = function (nums, target) {  let map = new Map();   for (let i = 0; i < nums.length; i++) {  if (map.has(target - nums[i])) {  return [i, map.get(target - nums[i])];  } else {  map.set(nums[i], i);  }  }   return []; };  *// Test Case: Typical Case* let nums = [2, 7, 11, 15]; let target = 9; let result = twoSum(nums, target); console.log(result); *// Expected Output: [1, 0] (indices of 7 and 2 which sum up to 9)* |
| --- |

### **Longest Consecutive Sequence**

<https://leetcode.com/problems/longest-consecutive-sequence/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/* var longestConsecutive = function (nums) {  let table = new Set([...nums]);  let max = 0;   for (let num of nums) {  if (!table.has(num - 1)) {  let current = num;  let count = 0;   while (table.has(current)) {  current++;  count++;  max = Math.max(count, max);  }  }  }   return max; };   *// Test Case: Typical Case* let nums = [100, 4, 200, 1, 3, 2]; let result = longestConsecutive(nums); console.log(result); *// Expected Output: 4 (The longest consecutive sequence is [1, 2, 3, 4])* |
| --- |

### **Largest subarray with 0 sum**

<https://www.geeksforgeeks.org/problems/largest-subarray-with-0-sum/1>

| class Solution {  maxLen(arr, n) {  *//code here*   let map = new Map();  let sum = 0;  let maxi = 0;   for (let i = 0; i < n; i++) {  sum = sum + arr[i];   if (sum == 0) {  maxi = i + 1;  } else if (map.has(sum)) {  console.log(i, map.get(sum));  maxi = Math.max(maxi, i - map.get(sum));  } else {  map.set(sum, i);  }  }   return maxi;  } }  *// Test Case: Typical Case* let arr = [15, -2, 2, -8, 1, 7, 10, 23]; let n = arr.length;  const solution = new Solution(); let result = solution.maxLen(arr, n); console.log(result); *// Expected Output: 5 (The subarray with sum 0 is [-2, 2, -8, 1, 7])* |
| --- |

### **Longest Substring Without Repeating Characters**

<https://leetcode.com/problems/longest-substring-without-repeating-characters/description/>

| 1st method  */\*\*  \* @param {string} s  \* @return {number}  \*/* var lengthOfLongestSubstring = function (s) {  let left = 0;  let table = new Set();  let maxi = 0;   for (let right = 0; right < s.length; right++) {  while (table.has(s[right])) {  table.delete(s[left]);  left += 1;  }   table.add(s[right]);  maxi = Math.max(maxi, right - left + 1);  }  return maxi; };      */\*\*  \* @param {string} s  \* @return {number}  \*/* var lengthOfLongestSubstring = function (s) {  let maxi = 0;  let table = new Map();  let left = 0;  for (let i = 0; i < s.length; i++) {  if (table.has(s[i])) {  left = Math.max(left, table.get(s[i]) + 1);  }   table.set(s[i], i);  maxi = Math.max(maxi, i - left + 1);  }   return maxi; };   Input: "abcabcbb" Expected Output: 3  Test Case 3: Input: "pwwkew" Expected Output: 3 |
| --- |

## 

### **Roman to Integer and Intget to Roam**

<https://leetcode.com/problems/roman-to-integer/description/>

| */\*\*  \* @param {string} s  \* @return {number}  \*/* var romanToInt = function(s) {   let hash=new Map([["I",1],["V",5],["X",10],["L",50],["C",100],["D",500],["M",1000]])  let res=0   for(let i=0;i<s.length;i++){  if(i+1<s.length && hash.get(s[i])<hash.get(s[i+1])){  res=res-hash.get(s[i])   }   else{  res=res+hash.get(s[i])  }   console.log(res)  }   return res;    };  var romanToInt = function(s) {  *// Define an object to store the numerical values of each Roman numeral*  let numeralValues = {  "I": 1,  "V": 5,  "X": 10,  "L": 50,  "C": 100,  "D": 500,  "M": 1000  };    let res = 0; *// Initialize the result variable to store the integer value*    *// Loop through each character in the Roman numeral string*  for (let i = 0; i < s.length; i++) {  *// Check if the current numeral is smaller than the next one*  if (i + 1 < s.length && numeralValues[s[i]] < numeralValues[s[i + 1]]) {  *// If it is, subtract its value from the result*  res = res - numeralValues[s[i]];  } else {  *// If not, add its value to the result*  res = res + numeralValues[s[i]];  }  console.log(res); *// Log the current result value during each iteration*  }    return res; *// Return the final result* };  romanToInt("MMXXIV"); *// Call the function with the Roman numeral "MMXXIV"*  The main logic of the code lies in iterating through the Roman numeral string and determining whether to add or subtract the value of each numeral based on its position relative to the next numeral.  Here's how it works:  1. It loops through each character in the Roman numeral string. 2. For each character:  - If the current numeral is smaller than the next numeral, it subtracts its value from the result.  - If the current numeral is not smaller than the next numeral, it adds its value to the result.  This logic ensures that it correctly handles cases where a smaller numeral appears before a larger one, indicating subtraction, and cases where no subtraction is needed. |
| --- |

## 

### **Integer to Roman**

<https://leetcode.com/problems/integer-to-roman/description/>

| var intToRoman = function (num) {  let map = {  M: 1000,  CM: 900,  D: 500,  CD: 400,  C: 100,  XC: 90,  L: 50,  XL: 40,  X: 10,  IX: 9,  V: 5,  IV: 4,  I: 1,  }; *// total 13 mappings*  let res = "";   for (let key in map) {  while (num >= map[key]) {  res = res + key;  num = num - map[key];  }  }  return res; };  console.log(intToRoman(9)); // Expected: "IX"  console.log(intToRoman(58)); // Expected: "LVIII"  console.log(intToRoman(1994)); // Expected: "MCMXCIV" |
| --- |

## Recurrsion

### **Letter Combinations of a Phone Number**

<https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/?envType=list&envId=xlere2g3>

| */\*\*  \* @param {string} digits  \* @return {string[]}  \*/* var letterCombinations = function (digits) {  if (digits == null || digits.length === 0) return [];   let map = new Map();  map.set("2", "abc");  map.set("3", "def");  map.set("4", "ghi");  map.set("5", "jkl");  map.set("6", "mno");  map.set("7", "pqrs");  map.set("8", "tuv");  map.set("9", "wxyz");  let res = [];   let helper = (index, currSt) => {  if (currSt.length == digits.length) {  res.push(currSt);  return;  }   for (let c of map.get(digits[index])) {  helper(index + 1, currSt + c);  }  };   helper(0, "");   return res; };    *//another way* */\*\*  \* @param {string} digits  \* @return {string[]}  \*/* var letterCombinations = function (digits) {  if (!digits) return [];   let res = [];  let temp = [];   let map = {};  map[2] = "abc";  map[3] = "def";  map[4] = "ghi";  map[5] = "jkl";  map[6] = "mno";  map[7] = "pqrs";  map[8] = "tuv";  map[9] = "wxyz";   function helper(index) {  if (temp.length == digits.length) {  res.push(temp.join(""));  return;  }  for (let ch of map[digits[index]]) {  temp.push(ch);  helper(index + 1);  temp.pop();  }  }   helper(0);  return res; };      "abc" "def" *//Certainly! Let's represent the function calls and their respective arguments in a tree-like structure:*   letterCombinations("23")  |  └── go(0, '')  ├── go(1, 'a')  | ├── go(2, 'ad') => "ad"  | ├── go(2, 'ae') => "ae"  | └── go(2, 'af') => "af"  |  ├── go(1, 'b')  | ├── go(2, 'bd') => "bd"  | ├── go(2, 'be') => "be"  | └── go(2, 'bf') => "bf"  |  └── go(1, 'c')  ├── go(2, 'cd') => "cd"  ├── go(2, 'ce') => "ce"  └── go(2, 'cf') => "cf"        letterCombinations("234")  |  └── helper(0, '')  ├── helper(1, 'a')  | ├── helper(2, 'ad')  | | ├── helper(3, 'adg') => "adg"  | | ├── helper(3, 'adh') => "adh"  | | └── helper(3, 'adi') => "adi"  | |  | ├── helper(2, 'ae')  | | ├── helper(3, 'aeg') => "aeg"  | | ├── helper(3, 'aeh') => "aeh"  | | └── helper(3, 'aei') => "aei"  | |  | └── helper(2, 'af')  | ├── helper(3, 'afg') => "afg"  | ├── helper(3, 'afh') => "afh"  | └── helper(3, 'afi') => "afi"  |  ├── helper(1, 'b')  | ├── helper(2, 'bd')  | | ├── helper(3, 'bdg') => "bdg"  | | ├── helper(3, 'bdh') => "bdh"  | | └── helper(3, 'bdi') => "bdi"  | |  | ├── helper(2, 'be')  | | ├── helper(3, 'beg') => "beg"  | | ├── helper(3, 'beh') => "beh"  | | └── helper(3, 'bei') => "bei"  | |  | └── helper(2, 'bf')  | ├── helper(3, 'bfg') => "bfg"  | ├── helper(3, 'bfh') => "bfh"  | └── helper(3, 'bfi') => "bfi"  |  └── helper(1, 'c')  ├── helper(2, 'cd')  | ├── helper(3, 'cdg') => "cdg"  | ├── helper(3, 'cdh') => "cdh"  | └── helper(3, 'cdi') => "cdi"  |  ├── helper(2, 'ce')  | ├── helper(3, 'ceg') => "ceg"  | ├── helper(3, 'ceh') => "ceh"  | └── helper(3, 'cei') => "cei"  |  └── helper(2, 'cf')  ├── helper(3, 'cfg') => "cfg"  ├── helper(3, 'cfh') => "cfh"  └── helper(3, 'cfi') => "cfi" |
| --- |

### **Combinations**

<https://leetcode.com/problems/combinations/description/?envType=list&envId=xlere2g3>

| */\*\*  \* @param {number} n  \* @param {number} k  \* @return {number[][]}  \*/* var combine = function (n, k) {  let arr = new Array(n).fill(0).map((item, index) => index + 1);  *//console.log(arr);*  let res = [];  let ds = [];   helper(0);  return res;   function helper(index) {  if (ds.length == k) {  *// console.log("base",ds)*  res.push([...ds]);  return;  }   if (index == n) return;   *// console.log([...ds],index,'inside')*  ds.push(arr[index]);  helper(index + 1);  ds.pop();   *// console.log([...ds],index,'inside2')*   helper(index + 1);  } };  *// Test Case: Typical Case* let n1 = 4, k1 = 2; let result1 = combine(n1, k1); console.log(result1);  *// Expected Output: [* *// [1, 2],* *// [1, 3],* *// [1, 4],* *// [2, 3],* *// [2, 4],* *// [3, 4]* *// ]* |
| --- |

### **Subset Sums**

<https://www.geeksforgeeks.org/problems/subset-sums2234/1>

| class Solution {  subsetSums(arr, n) {  *//code here*   let result = [];   this.helper( n - 1, 0);   return result.sort((a, b) => a - b);  }   helper( index, sum) {  *// console.log(index)*   if (index < 0) {  *// console.log(result)*  result.push(sum);  return;  }   this.helper(index - 1, sum + arr[index);   this.helper( index - 1, sum);  } }  let arr1 = [1, 2, 3]; let n1 = arr1.length; const solution1 = new Solution(); let result1 = solution1.subsetSums(arr1, n1); console.log(result1);  *// Expected Output: [0, 1, 2, 3, 3, 4, 5, 6]* |
| --- |

### **Subsets**

<https://leetcode.com/problems/subsets/description/?envType=list&envId=xlere2g3>

| */\*\*  \* @param {number[]} nums  \* @return {number[][]}  \*/* var subsets = function (nums) {  let res = [];  let ds = [];   let n = nums.length;   var combination = (index, ds, k) => {  if (ds.length == k) {  *// console.log(ds,'in if')*  res.push([...ds]);  return;  }   if (index < 0) return;   *// console.log(index,ds)*  ds.push(nums[index]);  combination(index - 1, ds, k);  ds.pop();  combination(index - 1, ds, k);  };  *// it is just extension of combination of size k wheren k is from 0 ton n*  for (let i = 0; i <= n; i++) {  combination(n - 1, ds, i);  }   return res; };    *// altrenate solution*  */\*\*  \* @param {number[]} nums  \* @return {number[][]}  \*/* var subsets = function (nums) {  let res = [];  let ds = [];   let n = nums.length;   var combination = (index) => {  if (index < 0) {  res.push([...ds]);  return;  }   ds.push(nums[index]);  combination(index - 1);  ds.pop();  combination(index - 1);  };   combination(n - 1);   return res; };  *// Test Case: Typical Case* let nums1 = [1, 2, 3]; let result1 = subsets(nums1); console.log(result1);  *// Expected Output: [* *// [],* *// [1],* *// [2],* *// [1, 2],* *// [3],* *// [1, 3],* *// [2, 3],* *// [1, 2, 3]* *// ]* |
| --- |

### **Subsets II**

<https://leetcode.com/problems/subsets-ii/>

| */\*\*  \* @param {number[]} nums  \* @return {number[][]}  \*/* var subsetsWithDup = function (nums) {  nums.sort((a, b) => a - b);   let res = [];  let n = nums.length;  let ds = [];   helper(n - 1);  return res;   function helper(index) {  *// console.log(index)*   if (index < 0) {  res.push([...ds]);  return;  }   ds.push(nums[index]);  helper(index - 1);  ds.pop();   while (index > 0 && nums[index] == nums[index - 1]) index--;   helper(index - 1);  } };  let nums1 = [1, 2, 2]; let result1 = subsetsWithDup(nums1); console.log(result1);  *// Expected Output: [* *// [],* *// [1],* *// [1, 2],* *// [1, 2, 2],* *// [2],* *// [2, 2]* *// ]* |
| --- |

### **Combination Sum**

<https://leetcode.com/problems/combination-sum/>

| */\*\*  \* @param {number[]} candidates  \* @param {number} target  \* @return {number[][]}  \*/* var combinationSum = function (candidates, target) {  let res = [];  let ds = [];   let n = candidates.length;  helper(n - 1, target, ds);  *// console.log(res)*  return res;   function helper(index, target, ds) {  if (target == 0) {  res.push([...ds]);  return;  }   if (target < 0) return;   if (index < 0) return;   *// console.log(ds,target)*   ds.push(candidates[index]);   *//here we are sending index because we can use same number any number of times*  helper(index, target - candidates[index], ds);  ds.pop();   helper(index - 1, target, ds);  } };   let candidates1 = [2, 3, 6, 7]; let target1 = 7; let result1 = combinationSum(candidates1, target1); console.log(result1); *// Expected Output: [* *// [2, 2, 3],* *// [7]* *// ]* |
| --- |

## **Combination Sum II**

<https://leetcode.com/problems/combination-sum-ii/description/>

| */\*\*  \* @param {number[]} candidates  \* @param {number} target  \* @return {number[][]}  \*/* var combinationSum2 = function (candidates, target) {  let arr = candidates;  arr.sort((a, b) => a - b);   let res = [];  let ds = [];  let n = arr.length;  helper(n - 1, target);  return res;   function helper(index, target) {  *//below condition has to on top*  *//because may be for index<0 also gives answer*  if (target == 0) {  res.push([...ds]);  return;  }   if (target < 0) return;   if (index < 0) return;   ds.push(arr[index]);  helper(index - 1, target - arr[index]); *//pick*   ds.pop();   while (index > 0 && arr[index] == arr[index - 1]) index--;   helper(index - 1, target); *//not pick*  } };  let candidates1 = [10, 1, 2, 7, 6, 1, 5];  let target1 = 8;  let result1 = combinationSum2(candidates1, target1);  console.log(result1);  // Expected Output: [  // [1, 1, 6],  // [1, 2, 5],  // [1, 7],  // [2, 6]  // ] |
| --- |

### **Palindrome Partitioning**

<https://leetcode.com/problems/palindrome-partitioning/>

| */\*\*  \* @param {string} s  \* @return {string[][]}  \*/* var partition = function (s) {  let result = [];  let ds = [];   helper(0);  return result;   function helper(start) {  if (start === s.length) {  result.push([...ds]);  return;  }   for (let end = start; end < s.length; end++) {  let subStr = s.slice(start, end + 1);  if (isPalin(subStr)) {  ds.push(subStr);  helper(end + 1); *// its end + i (variable inside loop)*  ds.pop();  }  }  }   function isPalin(s) {  let reverse = s.split("").reverse().join("");  return s === reverse;  } };   let s1 = "aab"; let result1 = partition(s1); console.log(result1); *// Expected Output:* *// [* *// ["a", "a", "b"],* *// ["aa", "b"]* *// ]* |
| --- |

### **Permutations**

<https://leetcode.com/problems/permutations/>

| *//solution 2* */\*\*  \* @param {number[]} nums  \* @return {number[][]}  \*/* var permute = function(nums) {   nums.sort((a, b) => a - b);   let res = [];  let ds = [];  let n=nums.length  let used=new Array(nums.length).fill(false)   let helper = (index) => {  if (index<0) {  res.push([...ds]);  return;  }   for (let i = 0; i < nums.length; i++) {  if (used[i] ) {  continue;   *// Skip if the number is already used or if it's a duplicate and the previous one is not used*  }  ds.push(nums[i]);  used[i] = true;  helper(index - 1);  ds.pop();  used[i] = false;  }  };   helper(n-1 );   return res;     }  let nums1 = [1, 2, 3]; let result1 = permute(nums1); console.log(result1); *// Expected Output:* *// [* *// [1, 2, 3],* *// [1, 3, 2],* *// [2, 1, 3],* *// [2, 3, 1],* *// [3, 1, 2],* *// [3, 2, 1]* *// ]* |
| --- |

### **Permutations II**

<https://leetcode.com/problems/permutations-ii/?envType=list&envId=xlere2g3>

| */\*\*  \* @param {number[]} nums  \* @return {number[][]}  \*/* var permuteUnique = function (nums) {  nums.sort((a, b) => a - b);   let res = [];  let ds = [];  let n = nums.length;  let used = new Array(nums.length).fill(false);   let helper = (index) => {  if (index < 0) {  res.push([...ds]);  return;  }   for (let i = 0; i < nums.length; i++) {  if (used[i] || (i > 0 && nums[i] === nums[i - 1] && !used[i - 1])) {  continue; *// Skip if the number is already used or if it's a duplicate and the previous one is not used*  }   ds.push(nums[i]);  used[i] = true;  helper(index - 1);  ds.pop();  used[i] = false;  }  };   helper(n - 1);   return res; };  let nums1 = [1, 2, 3]; let result1 = permuteUnique(nums1); console.log(result1); *// Expected Output:* *// [* *// [1, 2, 3],* *// [1, 3, 2],* *// [2, 1, 3],* *// [2, 3, 1],* *// [3, 1, 2],* *// [3, 2, 1]* *// ]* |
| --- |

### **N-Queens**

<https://leetcode.com/problems/n-queens/>

### **Rat in a Maze Problem - I**

<https://www.geeksforgeeks.org/problems/rat-in-a-maze-problem/1>

| class Solution {  findPath(m, n) {  *//code here*   let visited = new Array(n).fill(0).map(() => new Array(n).fill(false));   let path = [];  let result = [];   this.helper(m, 0, 0, path, visited, n, result);   return result.sort();  }   helper(grid, row, col, path, visited, n, result) {  if (row == n - 1 && col == n - 1) {  result.push(path.slice().join("")); *// Push a copy of the path array*  *// console.log(path)*  return;  }   visited[row][col] = true;   let rowMove = [-1, 1, 0, 0];  let colMove = [0, 0, -1, 1];  let dir = ["U", "D", "L", "R"];   for (let i = 0; i < rowMove.length; i++) {  let nextRow = row + rowMove[i];  let nexCol = col + colMove[i];  let newPath = [...path, dir[i]];   if (this.isSafe(nextRow, nexCol, grid, visited, n)) {  this.helper(grid, nextRow, nexCol, newPath, visited, n, result);  }  }   visited[row][col] = false;  }   isSafe(row, col, grid, visited, n) {  return (  row >= 0 &&  row < n &&  col >= 0 &&  col < n &&  visited[row][col] == false &&  grid[row][col] == 1  );  } }  let grid1 = [  [1, 0, 0],  [1, 1, 0],  [0, 1, 1] ];  let solution1 = new Solution(); let result1 = solution1.findPath(grid1, 3); console.log(result1); *// Expected Output:* *// ["DDR", "DDRR", "DRDD", "RDD", "RRDD"]* *// These are the valid paths from (0,0) to (2,2) in the grid.* |
| --- |

## Binary Search

### **Single Element in a Sorted Array**

<https://leetcode.com/problems/single-element-in-a-sorted-array/description/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/* var singleNonDuplicate = function (nums) {  let start = 0;  let end = nums.length - 1;   if (end == 0) return nums[0];   if (nums[start] != nums[start + 1]) return nums[start];  if (nums[end] != nums[end - 1]) return nums[end];   while (start <= end) {  let mid = Math.floor((start + end) / 2);  *// console.log(mid)*  if (nums[mid] != nums[mid + 1] && nums[mid] != nums[mid - 1])  return nums[mid];  if (  (mid % 2 == 0 && nums[mid] == nums[mid + 1]) ||  (mid % 2 == 1 && nums[mid] == nums[mid - 1])  )  start = mid + 1;  else end = mid - 1;  }   return -1; };   [1,1,2,2,3,3,4,4,6,6] if two elements are present then even index and even index+1 should be same  so we have to serch from mid+1 to end else  we have to search in left side 0. left   let nums1 = [1, 1, 2, 2, 3, 3, 4, 4, 6, 6]; let result1 = singleNonDuplicate(nums1); console.log(result1); |
| --- |

### **Search in Rotated Sorted Array**

<https://leetcode.com/problems/search-in-rotated-sorted-array/>

| var search = function (nums, target) {  let low = 0;  let high = nums.length - 1;  let arr = nums;   while (low <= high) {  let mid = Math.floor((low + high) / 2);   if (target == arr[mid]) return mid;   if (arr[low] <= arr[mid]) {  if (target >= arr[low] && target <= arr[mid]) {  high = mid - 1;  } else {  low = mid + 1;  }  } else {  if (target >= arr[mid] && target <= arr[high]) {  low = mid + 1;  } else {  high = mid - 1;  }  }  }   return -1; };  let nums1 = [4, 5, 6, 7, 0, 1, 2]; let target1 = 0; let result1 = search(nums1, target1); console.log(result1); *// Expected Output: 4* *// The target 0 is found at index 4 in the array [4, 5, 6, 7, 0, 1, 2].* |
| --- |

### **Median of Two Sorted Arrays**

<https://leetcode.com/problems/median-of-two-sorted-arrays/description/>

| var findMedianSortedArrays = function (nums1, nums2) {  *// num1 should be small*  if (nums1.length > nums2.length) {  let temp = nums1;  nums1 = nums2;  nums2 = temp;  }   let m = nums1.length;  let n = nums2.length;   let low = 0;  let high = m;   while (low <= high) {  let cut1 = (low + high) >> 1;  let cut2 = Math.floor((m + n + 1) / 2) - cut1;   let l1 = cut1 == 0 ? -Infinity : nums1[cut1 - 1];  let l2 = cut2 == 0 ? -Infinity : nums2[cut2 - 1];   let r1 = cut1 == m ? Infinity : nums1[cut1];  let r2 = cut2 == n ? Infinity : nums2[cut2];   if (l1 <= r2 && l2 <= r1) {  console.log(l1, l2, r1, r2);   if ((m + n) % 2 == 1) {  return Math.max(l1, l2);  } else {  return (Math.max(l1, l2) + Math.min(r1, r2)) / 2;  }  } else if (l1 > r2) {  high = cut1 - 1;  } else {  low = cut1 + 1;  }  }   return 0.0; };  let nums1 = [1, 3]; let nums2 = [2]; let result1 = findMedianSortedArrays(nums1, nums2); console.log(result1); *// Expected Output: 2.0* *// Median of [1, 2, 3] is 2.0.* |
| --- |

### **K-th element of two Arrays**

<https://www.geeksforgeeks.org/problems/k-th-element-of-two-sorted-array1317/1>

| class Solution {  kthElement(A, B, n, m, k) {  *//code here*   if (A.length > B.length) {  let temp = A;  A = B;  B = temp;  n = A.length;  m = B.length;  }   *// if A has 6 element and B has 5 element and k is 7 we have to pick atlease 7{k) -5(m) => 2 from A . because B has only 5 element*  let low = Math.max(0, k - m);   *// if A has 6 elemetn and k=2 then we have to pick max 2 from A . So we have to pick min of k,n*  let high = Math.min(k, n);   while (low <= high) {  let cut1 = Math.floor((low + high) / 2);   let cut2 = k - cut1; *// we have divide in such a way that k element in one group, remainng in other group*   let l1 = cut1 == 0 ? -Infinity : A[cut1 - 1];  let l2 = cut2 == 0 ? -Infinity : B[cut2 - 1];   let r1 = cut1 == n ? Infinity : A[cut1];  let r2 = cut2 == m ? Infinity : B[cut2];   if (l1 <= r2 && l2 <= r1) {  return Math.max(l1, l2);  } else if (l1 > r2) {  high = cut1 - 1;  } else {  low = cut1 + 1;  }  }   return 1;  } }  let solution = new Solution(); let A1 = [1, 3, 5, 7, 9]; let B1 = [2, 4, 6, 8]; let n1 = A1.length; let m1 = B1.length; let k1 = 5; let result1 = solution.kthElement(A1, B1, n1, m1, k1); console.log(result1); *// Expected Output: 5* *// The 5th smallest element in [1, 2, 3, 4, 5, 6, 7, 8, 9] is 5.* |
| --- |

### **areSquaresSumExist**

| function areSquaresSumExist(c) {  if (c < 0) return false; *// Negative numbers cannot be expressed as sum of two squares*    let start = 0;  let end = Math.floor(Math.sqrt(c));    while (start <= end) {  let sum = start \* start + end \* end;    if (sum === c) {  return true;  } else if (sum < c) {  start++;  } else {  end--;  }  }    return false; }  *// Example usage:* console.log(areSquaresSumExist(5)); *// Output: true (2^2 + 1^2 = 5)* console.log(areSquaresSumExist(13)); *// Output: true (3^2 + 2^2 = 13)* console.log(areSquaresSumExist(25)); *// Output: true (5^2 + 0^2 = 25)* console.log(areSquaresSumExist(7)); *// Output: false* |
| --- |

### **Find Smallest Letter Greater Than Target**

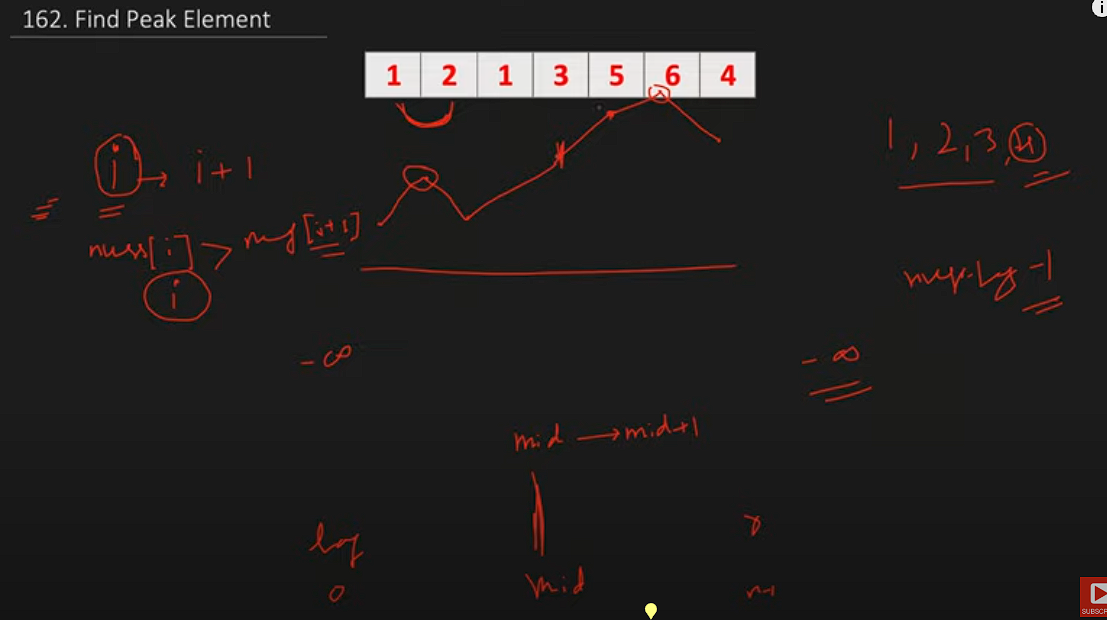
<https://leetcode.com/problems/find-smallest-letter-greater-than-target/>

| var nextGreatestLetter = function(letters, target) {   let low = 0;  let high = letters.length - 1;  let ans = ""   while (low <= high) {   let mid = Math.floor((low + high) / 2)   if (letters[mid] > target) {  ans = letters[mid]  high = mid - 1  } else {  low = mid + 1  }   }   return ans == "" ? letters[0] : ans  };  Input: letters = ['a', 'c', 'f', 'h'], target = 'd' Output: 'f' Explanation: The smallest letter greater than 'd' is 'f'. |
| --- |

### **Find Peak Element**

<https://leetcode.com/problems/find-peak-element/description/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/* var findPeakElement = function (nums) {  let low = 0;  let high = nums.length - 1;   while (low <= high) {  let mid = Math.floor((low + high) / 2);   *//ex: [3,5,7][mid mid+7] here 3 <5 so 3 cann't be peak element, so low=3*  if (nums[mid] < nums[mid + 1]) {  low = mid + 1;  } else {  high = mid - 1;  }  }   return low; *//need to return low* }; |
| --- |



### **Peak Index in a Mountain Array**

<https://leetcode.com/problems/peak-index-in-a-mountain-array/?envType=list&envId=xleplgq3>

| */\*\*  \* @param {number[]} arr  \* @return {number}  \*/* var peakIndexInMountainArray = function (arr) {  let start = 0;  let end = arr.length - 1;   while (start <= end) {  let mid = Math.floor((start + end) / 2);   if (arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1]) return mid;   *// if mid is in increasing side , we just need to skil ex: [1,2,3,4,5,4,3,1] and*  *// mid is 3 mid+1 is 4 then skip left element*  if (arr[mid] < arr[mid + 1]) {  start = mid + 1;  } else {  end = mid - 1;  }  } };  console.log(peakIndexInMountainArray([1, 3, 5, 4, 2]));  // Expected: 2 |
| --- |

### **Search in Rotated Sorted Array**

<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/?envType=list&envId=xleplgq3>

| */\*\*  \* @param {number[]} nums  \* @param {number} target  \* @return {number}  \*/* var search = function(nums, target) {   let low=0;  let high=nums.length-1  let arr=nums   while(low<=high){   let mid=Math.floor((low+high)/2)   if(target==arr[mid]) return mid   if(arr[low]<=arr[mid]){    if(target>=arr[low] && target<=arr[mid]){  high=mid-1  }  else{  low=mid+1  }   }  else{   if(target>=arr[mid] && target<=arr[high]){  low=mid+1  }  else{  high=mid-1  }   }   }   return -1      };  console.log(search([4, 5, 6, 7, 0, 1, 2], 0)); *// Expected output: 4* console.log(search([4, 5, 6, 7, 0, 1, 2], 3)); *// Expected output: -1* |
| --- |

### **Find Minimum in Rotated Sorted Array**

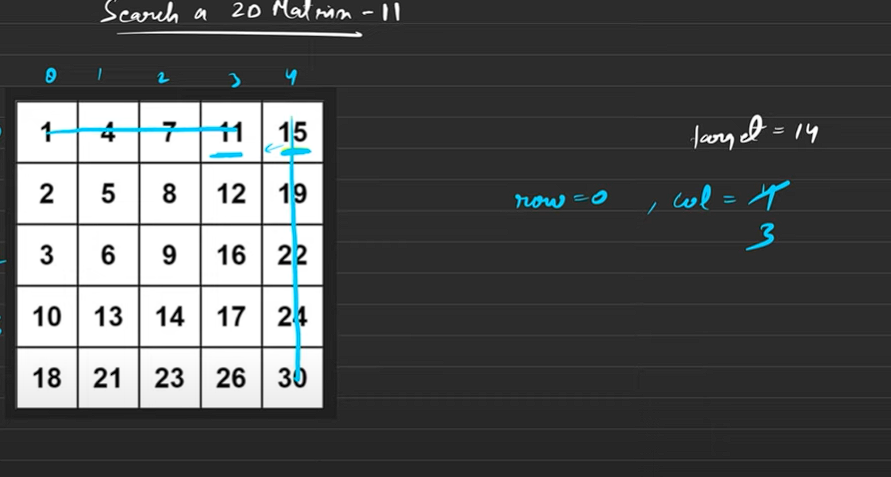
<https://leetcode.com/problems/find-smallest-letter-greater-than-target/description/>

| */\*\*  \* @param {number[]} nums  \* @return {number}  \*/*  <https:*//www.youtube.com/watch?v=nhEMDKMB44g&t=789s>* var findMin = function(nums) {   let left=0; right=nums.length-1   let res=Number.MAX\_SAFE\_INTEGER   while(left<=right){   let mid=Math.floor((left+ right)/2)   console.log(nums[mid],'mid',left,right)    *// pick min in sorted and elementate that portion as we alread picked smalles in sorted.*  if(nums[left]<=nums[mid]){  res=Math.min(nums[left],res)  left=mid+1  }  else{  res=Math.min(nums[mid],res)  right=mid-1  }  }   return res;      };  console.log(findMin([3, 4, 5, 1, 2])); *// Expected output: 1* console.log(findMin([4, 5, 6, 7, 0, 1, 2])); *// Expected output: 0* console.log(findMin([11, 13, 15, 17])); *// Expected output: 11* |
| --- |

### **Search a 2D Matrix**

<https://leetcode.com/problems/search-a-2d-matrix/?envType=list&envId=xleplgq3>

| var searchMatrix = function(matrix, target) {  let right = matrix[0].length - 1;  let top = 0;  let n = matrix.length;   while (top < n && right >= 0) {  if (matrix[top][right] === target) {  return true; *// Return true if target is found*  } else if (matrix[top][right] < target) {  top++; *// Move downwards if current element is less than target*  } else {  right--; *// Move leftwards if current element is greater than target*  }  }   return false; *// Return false if target is not found* };  var searchMatrix = function(matrix, target) {    let right=matrix[0].length-1  let top=0  let n=matrix.length      why top right posiitio? , because from top right array continuously decrease backwards,   increases donwwards    so if taget < current we need to back backwards    while(top<n && right>=0){  if(matrix[top][right]==target)  return 1  if(target<matrix[top][right]){  right-=1  }  else{  top+=1  }  }    return false     }; |
| --- |



## Heap based problems (solution in python)

| from heqpq import \*  bydefault it is minheap, but if we want max haap multipy no with -1 heappush(arr,num) => time complexity O(logn) heappop(arr) =>return min element in array in O(log n) time heappushpop(arr,num)  heapify(arr) => heapify arr in O(n) |
| --- |

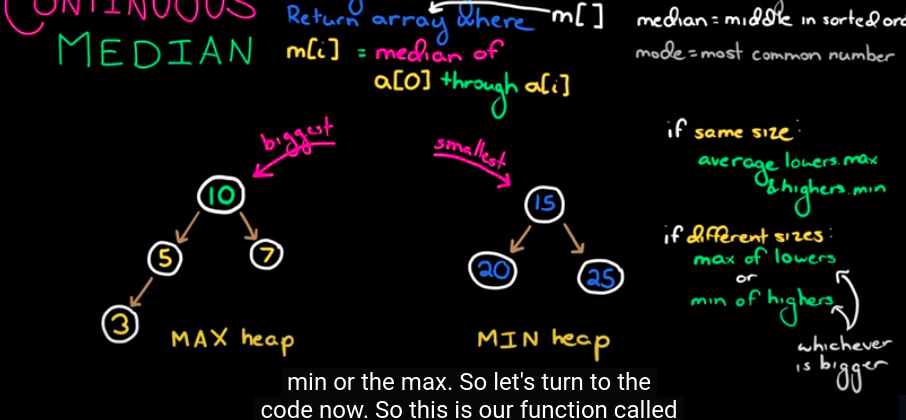
### **Kth Largest Element in an Array**

<https://leetcode.com/problems/kth-largest-element-in-an-array/description/>

| import heapq class Solution:  def findKthLargest(self, nums: List[int], k: int) -> int:   for i in range(len(nums)):  nums[i]=-nums[i]   heapq.heapify(nums)      while k>1:  heapq.heappop(nums)  k-=1      return -heapq.heappop(nums)  The time complexity of finding the kth largest element in an array using a heap in Python is O(n + k log n), where n is the number of elements in the array.  solution = Solution() nums1 = [3, 2, 1, 5, 6, 4] k1 = 2 result1 = solution.findKthLargest(nums1, k1) print(result1) # Expected Output: 5 # Explanation: The 2nd largest element in [3, 2, 1, 5, 6, 4] is 5. |
| --- |

### Median finder

| from heapq import \*  class MedianFinder:  def \_\_init\_\_(self):  self.lo = []   self.hi = []    # lo => lowers, hi=>higher   def addNum(self, num):  # heappush(self.lo, -num) # lo is maxheap, so -1 \* num  # heappush(self.hi, -self.lo[0]) # hi is minheap  # heappop(self.lo)  heappush(self.hi, -heappushpop(self.lo, -num))     if len(self.lo) < len(self.hi):  heappush(self.lo, -self.hi[0])  heappop(self.hi)    def findMedian(self):  if len(self.lo) > len(self.hi):  return -self.lo[0]   else:  return (self.hi[0] - self.lo[0]) / 2 # - as low has -ve values         # Your MedianFinder object will be instantiated and called as such: # obj = MedianFinder() # obj.addNum(num) # param\_2 = obj.findMedian() |
| --- |



### **Sliding Window Maximum**

<https://leetcode.com/problems/sliding-window-maximum/description/>

| from heapq import \* class Solution:  def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:  result=[]  heapArr=[]  for i in range(k):  heappush(heapArr, (-nums[i],i))    result.append(-heapArr[0][0])   # till now we have [0,1,2]  for i in range(k,len(nums)):  #we are adding 4th here so  # 4-3=1 . so untill first index we have to remove. so index [2,3,4]. that way we have to remove <=i-k  heappush(heapArr, (-nums[i],i))   while(heapArr[0][1]<=i-k):   heappop(heapArr)    result.append(-heapArr[0][0])    return result   # we have to create head of size k, we should always maintain same k size. # whatever indexs out of window, we need to reomve  nums = [1, 3, -1, -3, 5, 3, 6, 7] k = 3 # Expected output: [3, 3, 5, 5, 6, 7] # Explanation:  # Sliding window positions: # [1, 3, -1], -3, 5, 3, 6, 7 => max = 3 # 1, [3, -1, -3], 5, 3, 6, 7 => max = 3 # 1, 3, [-1, -3, 5], 3, 6, 7 => max = 5 # 1, 3, -1, [-3, 5, 3], 6, 7 => max = 5 # 1, 3, -1, -3, [5, 3, 6], 7 => max = 6 # 1, 3, -1, -3, 5, [3, 6, 7] => max = 7  # Run the function and print the result sol = Solution() output = sol.maxSlidingWindow(nums, k) print(output) # This will print: [3, 3, 5, 5, 6, 7] |
| --- |

## 

## Stack and Queue

### **Top K Frequent Elements**

<https://leetcode.com/problems/top-k-frequent-elements/>

| */\*\*  \* @param {number[]} nums  \* @param {number} k  \* @return {number[]}  \*/* var topKFrequent = function (nums, k) {  let count = new Map();  let n = nums.length;  for (let i = 0; i < n; i++) {  count.set(nums[i], (count.get(nums[i]) ?? 0) + 1);  }   *// console.log(count)*   let freq = new Array(n + 1).fill(0).map(() => []);  *// console.log(freq)*   for ([key, count] of count.entries()) {  freq[count].push(key);  }   *// console.log(freq)*   *// let kcount=0*  let result = [];  for (let i = freq.length - 1; i >= 0; i--) {  for (n of freq[i]) {  result.push(n);  }  if (result.length == k) {  return result;  }  }   *// console.log(result)*   *// return result* }; |
| --- |

### **Valid Parentheses**

<https://leetcode.com/problems/valid-parentheses/>

| var isValid = function (s) {  let stack = [];  let store = new Map();  store.set("}", "{");  store.set(")", "(");  store.set("]", "[");   for (let char of s) {  if (store.has(char)) {  if(stack.length>0 && stack.at(-1)==store.get(char))  stack.pop()  else  return false    } else {  stack.push(char);  }  }   if (stack.length == 0) return true;  else return false; };  console.log(isValid("{[()]}")); *// Expected Output: true* *// Explanation: The string "{[()]}" contains balanced parentheses, brackets, and braces.* |
| --- |

### **Minimum Add to Make Parentheses Valid**

<https://leetcode.com/problems/minimum-add-to-make-parentheses-valid/>

| var minAddToMakeValid = function(s) {  const matching = new Map([  [')', '('],  ['}', '{'],  [']', '[']  ]);   let stack = [];   for (let ch of s) {  if (matching.has(ch) && stack.length > 0 && stack.at(-1) === matching.get(ch)) {  stack.pop();  } else {  stack.push(ch);  }  }   return stack.length; };   *//solution 2* */\*\*  \* @param {string} s  \* @return {number}  \*/* var minAddToMakeValid = function (s) {  let stack = [];   for (let ch of s) {  if (ch == ")" && (stack.length > 0) && (stack.at(-1) == "(")) {  stack.pop();  } else stack.push(ch);   *// console.log("char",ch)*  *// console.log(stack)*  }   return stack.length; };   console.log(minAddToMakeValid("{[()()]}")); *// Expected Output: 0* *// Explanation: The string "{[()()]}" is already valid.* |
| --- |

### **Largest Rectangle in Histogram**

<https://leetcode.com/problems/largest-rectangle-in-histogram/description/>

| */\*\*  \* @param {number[]} heights  \* @return {number}  \*/* var largestRectangleArea = function(heights) {    heights=[-1,...heights] *//to make indexing from 1*  let n=heights.length  let nextSm=new Array(n).fill(-1)  let prevSm=new Array(n).fill(-1)  let stack=[]    *//next smallest*  for(i=1;i<n;i++){  while(stack.length && heights[i]<heights[stack.at(-1)]){  let index=stack.pop()  nextSm[index]=i-1 *// we have to take i-1 element as i is smallar than current i-1 need to pic*   }  stack.push(i)  }   while(stack.length){  let index=stack.pop()  nextSm[index]=n-1   }   *//previous smallest*  stack=[]  for(i=n-1;i>0;i--){  while(stack.length && heights[i]<heights[stack.at(-1)]){  let index=stack.pop()  prevSm[index]=i+1 *// i will be smaller than current so, need to pick i+1*  }  stack.push(i)   }   while(stack.length){  let index=stack.pop()  prevSm[index]=1   }   let maxArea=0  for(let i=1;i<n;i++){  let height=heights[i];  let width=nextSm[i]-prevSm[i]+1   console.log(height,width)  maxArea=Math.max(maxArea,height\*width)  }     return maxArea   };  console.log(largestRectangleArea([2, 1, 5, 6, 2, 3]));  10 |
| --- |

### **Min Stack**

<https://leetcode.com/problems/min-stack/>

| var MinStack = function() {  this.stack=[]  this.min=[]   };  */\*\*   \* @param {number} val  \* @return {void}  \*/* MinStack.prototype.push = function(val) {   *// console.log(this)*   this.stack.push(val)   let lastMinele=this.min.at(-1)  let lastele=this.stack.at(-1) *// it is nothing but val(newly added value)*    if(lastele<=lastMinele || this.min.length==0)  this.min.push(val)    };  */\*\*  \* @return {void}  \*/* MinStack.prototype.pop = function() {    if(this.stack.length){   let lastele=this.stack.at(-1)  let lastMinele=this.min.at(-1)   if(lastele==lastMinele){  this.min.pop()  }  this.stack.pop()  return lastele   }               };  */\*\*  \* @return {number}  \*/* MinStack.prototype.top = function() {  if(this.stack.length)  return this.stack.at(-1)  return null   };  */\*\*  \* @return {number}  \*/* MinStack.prototype.getMin = function() {   if(this.min.length){  return this.min.at(-1)   }  return null  };  */\*\*   \* Your MinStack object will be instantiated and called as such:  \* var obj = new MinStack()  \* obj.push(val)  \* obj.pop()  \* var param\_3 = obj.top()  \* var param\_4 = obj.getMin()  \*/*    *// Test case 1: Pushing values*  var obj = new MinStack();  obj.push(3); obj.push(5); obj.push(2); obj.push(7);  console.log(obj.stack); *// Expected: [3, 5, 2, 7]* console.log(obj.min); *// Expected: [3, 2]*  *// Test case 2: Pop operation* console.log(obj.pop()); *// Expected: 7* console.log(obj.stack); *// Expected: [3, 5, 2]* console.log(obj.min); *// Expected: [3, 2]*  *// Test case 3: Top operation* console.log(obj.top()); *// Expected: 2*  *// Test case 4: GetMin operation* console.log(obj.getMin()); *// Expected: 2* |
| --- |

## String based Problems

### **Valid Anagram**

<https://leetcode.com/problems/valid-anagram/>

| var isAnagram = function (s, t) {  let table = new Map();  for (let c of s) {  table.set(c, (table.get(c) ?? 0) + 1);  }   for (let c of t) {  table.set(c, (table.get(c) ?? 0) - 1);  }   *// console.log(table)*  let values = [...table.values()];  console.log(values);  let res = values.every((item) => item == 0);  return res;   *// console.log(table)* };  console.log(isAnagram("anagram", "nagaram")); *// Expected: true* console.log(isAnagram("rat", "car")); *// Expected: false* console.log(isAnagram("", "")); *// Expected: true* console.log(isAnagram("a", "a")); *// Expected: true* |
| --- |

### **Longest Common Prefix**

<https://leetcode.com/problems/longest-common-prefix/>

| var longestCommonPrefix = function(strs) {    let firstStr=strs[0]  let firstStrlen=strs[0].length  let res=""  for(let i=0;i<firstStrlen;i++){  if(strs.every(str=>str[i]==firstStr[i])){  res+=firstStr[i]  }  else{  break  }  }  return res; };  console.log(longestCommonPrefix(["flower", "flow", "flight"])); *// Expected: "fl"* console.log(longestCommonPrefix(["dog", "racecar", "car"])); *// Expected: ""* console.log(longestCommonPrefix(["car", "car", "car"])); *// Expected: "car"* console.log(longestCommonPrefix(["apple", "apricot", "april"]));*// Expected: "ap"* |
| --- |

## Sliding Window

### **Maximum Average Subarray I**

<https://leetcode.com/problems/maximum-average-subarray-i/>

| */\*\*  \* @param {number[]} nums  \* @param {number} k  \* @return {number}  \*/*    var findMaxAverage = function (nums, k) {  let n = nums.length;   let sum = 0;  let max = -Infinity;   for (let i = 0; i < k; i++) {  sum = sum + nums[i];  }   max = Math.max(max, sum);   let left = 0;   for (let i = k; i < n; i++) {  sum = sum - nums[left] + nums[i];  left++;  max = Math.max(max, sum);  }   return max / k; };     *//solution 2* var findMaxAverage = function (nums, k) {  let n = nums.length;   let sum = 0;  let max = -Infinity;   for (let i = 0; i < k; i++) {  sum = sum + nums[i];  }   max = Math.max(max, sum);   for (let i = k; i < n; i++) {  sum = sum - nums[i - k] + nums[i];  max = Math.max(max, sum);  }   return max / k; };  Example 1:  Input: nums = [1,12,-5,-6,50,3], k = 4 Output: 12.75000 Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75 Example 2:  Input: nums = [5], k = 1 Output: 5.00000 |
| --- |

### **Permutation in String**

<https://leetcode.com/problems/permutation-in-string/?envType=list&envId=xlep8di5>

| */\*\*  \* @param {string} s1  \* @param {string} s2  \* @return {boolean}  \*/* var checkInclusion = function(s1, s2) {   let k=s1.length;  let s1map=new Map();  let s2map=new Map();   for(let i=0;i<k;i++){  s1map.set(s1[i], (s1map.get(s1[i]) ?? 0)+1 )  s2map.set(s2[i], (s2map.get(s2[i]) ?? 0)+1 )   }   let charcount=0  console.log(s1map.entries())  let match=[...s1map.entries()].every(([key,count])=>s2map.get(key)==count)   if(match)  return true   for(let i=k;i<s2.length;i++){  match=false  s2map.set(s2[i-k], (s2map.get(s2[i-k]) ?? 0)-1 )  s2map.set(s2[i], (s2map.get(s2[i]) ?? 0)+1 )   match=[...s1map.entries()].every(([key,count])=>s2map.get(key)==count)  if(match)  return true   }   return false     };  */\*  var checkInclusion = function (s1, s2) {  let s1map = new Map();  let s2map = new Map();   let k = s1.length; // window size   for (let i = 0; i < k; i++) {  s1map.set(s1[i], (s1map.get(s1[i]) ?? 0) + 1);  s2map.set(s2[i], (s2map.get(s2[i]) ?? 0) + 1);  }   let match = 0;   s1map.forEach((count, key, item) => {  if (s2map.get(key) == count) match += count;  });  if (match === k) return true;   for (let i = k; i < s2.length; i++) {  match = 0;   // remove i-k key  s2map.set(s2[i - k], s2map.get(s2[i - k]) - 1);   // add next key  s2map.set(s2[i], (s2map.get(s2[i]) ?? 0) + 1);   s1map.forEach((count, key, item) => {  if (s2map.get(key) == count) match += count;  });  if (match === k) return true;  }   return false; };  \*/*  Example 1:  Input: s1 = "ab", s2 = "eidbaooo" Output: true Explanation: s2 contains one permutation of s1 ("ba"). Example 2:  Input: s1 = "ab", s2 = "eidboaoo" Output: false |
| --- |

### **Find All Anagrams in a String**

<https://leetcode.com/problems/find-all-anagrams-in-a-string/description/>

| */\*\*  \* @param {string} s  \* @param {string} p  \* @return {number[]}  \*/* var findAnagrams = function (s, p) {  let k = p.length;  let pmap = new Map();  let smap = new Map();   for (let i = 0; i < k; i++) {  pmap.set(p[i], (pmap.get(p[i]) ?? 0) + 1);  smap.set(s[i], (smap.get(s[i]) ?? 0) + 1);  }   let res = [];  let matching = [...pmap.entries()].every(  ([key, count]) => smap.get(key) == count,  );  if (matching) res.push(0);   for (let i = k; i < s.length; i++) {  smap.set(s[i - k], (smap.get(s[i - k]) ?? 0) - 1);  smap.set(s[i], (smap.get(s[i]) ?? 0) + 1);   matching = [...pmap.entries()].every(  ([key, count]) => smap.get(key) == count,  );  if (matching) {  res.push(i - k + 1);  }  }   return res; };  console.log(findAnagrams("cbaebabacd", "abc")); // Expected: [0, 6] console.log(findAnagrams("abab", "ab")); // Expected: [0, 1, 2] console.log(findAnagrams("aaaaaaaaaa", "aa")); // Expected: [0, 1, 2, 3, 4, 5, 6, 7]  */\* var findAnagrams = function(s, p) {  let k = p.length;   let smap = new Map();  let pmap = new Map();   let res = [];   for (let i = 0; i < k; i++) {  smap.set(s[i], (smap.get(s[i]) ?? 0) + 1);  pmap.set(p[i], (pmap.get(p[i]) ?? 0) + 1);  }   let match = 0;   pmap.forEach((count, key) => {  if (smap.get(key) == count) match += count;  });   if (match == k) res.push(0);   for (let i = k; i < s.length; i++) {  match = 0;  smap.set(s[i - k], (smap.get(s[i - k]) ?? 0) - 1);  smap.set(s[i], (smap.get(s[i]) ?? 0) + 1);   // console.log(smap)  // console.log(pmap)   pmap.forEach((count, key) => {  if (smap.get(key) == count) match += count;  });   if (match == k) res.push(i - k + 1);  }   return res; }; \*/* |
| --- |

### **Substrings of Size Three with Distinct Characters**

<https://leetcode.com/problems/substrings-of-size-three-with-distinct-characters/description/>

| */\*\*  \* @param {string} s  \* @return {number}  \*/* var countGoodSubstrings = function(s) {   let k=3;  let map=new Map()   for(let i=0;i<k;i++){  map.set(s[i],(map.get(s[i])??0) +1 )  }    let count=0  if([...map.values()].filter(item=>item==1).length==3)  count++   *// console.log(count)*   for(let i=k;i<s.length;i++){  map.set(s[i-k],(map.get(s[i-k])??0) -1 )  map.set(s[i],(map.get(s[i])??0) +1 )   *// console.log(map)*  if([...map.values()].filter(item=>item==1).length==k)  count++    }   return count  };  console.log(countGoodSubstrings("xyzzaz")); // Expected: 1 console.log(countGoodSubstrings("aababcabc")); // Expected: 4 console.log(countGoodSubstrings("abcabc")); // Expected: 3 |
| --- |

**Minimum Size Subarray Sum**

<https://leetcode.com/problems/minimum-size-subarray-sum/description/?envType=list&envId=xlep8di5>

| */\*\*  \* @param {number} target  \* @param {number[]} nums  \* @return {number}  \*/* var minSubArrayLen = function(target, nums) {   let sum=0;  let minLen=Infinity;  let n=nums.length   let j=0;  let i=0;   for(let j=0;j<n;j++){  sum=sum+nums[j];   while(sum>=target){    minLen=Math.min(minLen,j-i+1);  sum=sum-nums[i];  i++;   }    }      return minLen==Infinity?0:minLen   };  console.log(minSubArrayLen(4, [1, 4, 4])); *// Expected: 1* console.log(minSubArrayLen(11, [1, 2, 3, 4, 5])); *// Expected: 3* console.log(minSubArrayLen(15, [1, 2, 3, 4, 5])); *// Expected: 5* console.log(minSubArrayLen(100, [1, 1, 1, 1, 1, 1, 1, 1])); *// Expected* |
| --- |

### **Longest Repeating Character Replacement**

<https://leetcode.com/problems/longest-repeating-character-replacement/?envType=list&envId=xlep8di5>

| function characterReplacement(s: string, k: number): number {  let map = new Map < string,  number > ();   let left = 0;  let right = 0;  let maxFreq = 0;   let longest = 0;   for (let right = 0; right < s.length; right++) {   map.set(s[right], (map.get(s[right]) ?? 0) + 1);  *// map.set(s[right], (map.get(s[right]) ?? 0) + 1);*   maxFreq = Math.max(maxFreq, map.get(s[right]) ?? 0);   *// let W=right-left+1;*   while (right - left + 1 - maxFreq > k) {  map.set(s[left], (map.get(s[left]) ?? 0) - 1)  left++  }   longest = Math.max(longest, right - left + 1)   }   return longest };  nsole.log(characterReplacement("ABAB", 2)); *// Expected: 4* console.log(characterReplacement("AABABBA", 1)); *// Expected: 4* console.log(characterReplacement("ABCDE", 2)); *// Expected: 5* console.log(characterReplacement("AAAA", 0)); *// Expected: 4* |
| --- |

## Array 2 pointers

### **Backspace String Compare**

<https://leetcode.com/problems/backspace-string-compare/description/?envType=list&envId=xlem03mm>

| var backspaceCompare = function(s, t) {   function getOrgText(str){   let arr=[]   for(let char of str){   if(char=="#" && arr.length){  arr.pop()  }  // we need to add only if it is not hash  if(char !="#"){  arr.push(char)  }  }   return arr   }   let t1=getOrgText(s)  let t2=getOrgText(t) // console.log(t1) // console.log(t2)    return t1.join("")==t2.join("")    };  Example 1:  Input: s = "ab#c", t = "ad#c" Output: true Explanation: Both s and t become "ac". Example 2:  Input: s = "ab##", t = "c#d#" Output: true Explanation: Both s and t become "". Example 3:  Input: s = "a#c", t = "b" Output: false Explanation: s becomes "c" while t becomes "b". |
| --- |

### **Squares of a Sorted Array**

<https://leetcode.com/problems/squares-of-a-sorted-array/description/?envType=list&envId=xlem03mm>

| */\*\*  \* @param {number[]} nums  \* @return {number[]}  \*/* var sortedSquares = function (nums) {  let start = 0;  let end = nums.length - 1;   let res = [];  while (start <= end) {  if (Math.abs(nums[start]) > Math.abs(nums[end])) {  res.push(nums[start] \* nums[start]);  start++;  } else {  res.push(nums[end] \* nums[end]);  end--;  }  }  return res.reverse(); };  Example 1:  Input: nums = [-4,-1,0,3,10] Output: [0,1,9,16,100] Explanation: After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100]. Example 2:  Input: nums = [-7,-3,2,3,11] Output: [4,9,9,49,121] |
| --- |

**Subarray Product Less Than K**

<https://leetcode.com/problems/subarray-product-less-than-k/description/>

| */\*\*  \* @param {number[]} nums  \* @param {number} k  \* @return {number}  \*/* var numSubarrayProductLessThanK = function (nums, k) {  if (k <= 1) return 0;   let left = 0;  let right = 0;   let n = nums.length;   let product = 1;  let ans = 0;   while (right < n) {  product = product \* nums[right];   while (product >= k) {  product = product / nums[left];  left++  }  ans = ans + (right - left + 1);  right++;  }   return ans; };  Example 1:  Input: nums = [10,5,2,6], k = 100 Output: 8 Explanation: The 8 subarrays that have product less than 100 are: [10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6] Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k. Example 2:  Input: nums = [1,2,3], k = 0 Output: 0 |
| --- |

### **3Sum Closest**

| */\*\*  \* @param {number[]} nums  \* @param {number} target  \* @return {number}  \*/* var threeSumClosest = function (nums, target) {  let n = nums.length;   nums.sort((a, b) => a - b);   let closest = Infinity;  let result = 0;   for (let i = 0; i < n; i++) {  let a = nums[i];   let j = i + 1;  let k = n - 1;   while (j < k) {  let b = nums[j];  let c = nums[k];   let sum = a + b + c;   if (sum == target) {  j++;  k--;  }   if (Math.abs(sum - target) < closest) {  closest = Math.abs(sum - target);  result = sum;  }   if (sum < target) {  j++;  }   if (sum > target) {  k--;  }  }  }   return result; }; |
| --- |

### **132 Pattern**

<https://leetcode.com/problems/132-pattern/description/>

| */\*\*  \* @param {number[]} nums  \* @return {boolean}  \*/* var find132pattern = function (nums) {  let stack = [];  let n = nums.length;  let third = -Infinity;   for (let i = n - 1; i >= 0; i--) {  if (nums[i] < nums[third]) return true;   *// console.log(stack)*   while (stack.length > 0 && nums[i] > nums[stack.at(-1)]) {  third = stack.pop();  }  stack.push(i);  }   return false; }; |
| --- |

# **Trees - BFS**

### **Minimum Depth of Binary Tree**

| var minDepth = function (root) {  if (!root) return [];   let queue = [root];  let res = [];  let depth = 1;   while (queue.length > 0) {  let size = queue.length;  *// console.log(size)*   while (size--) {  let curr = queue.shift();  *// console.log(curr.left, curr.right)*   if (!curr.left && !curr.right) return depth;   if (curr.left) queue.push(curr.left);  if (curr.right) queue.push(curr.right);  }   depth += 1;  *// console.log(depth,'depth')*  } }; |
| --- |

### **Binary Tree Zigzag Level Order Traversal**

<https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/?envType=list&envId=xlepfebm>

| var zigzagLevelOrder = function (root) {  if (!root) return [];   let queue = [root];  let res = [];  *// res.push([root.val])*  let level = 1;   while (queue.length > 0) {  let size = queue.length;  let temp = [];  while (size--) {  let curr = queue.shift();   if (curr.left) queue.push(curr.left);  if (curr.right) queue.push(curr.right);  temp.push(curr.val);  }  level += 1;   res.push(level % 2 == 0 ? temp : temp.reverse());  }   return res; }; |
| --- |

### **Binary Tree Level Order Traversal II**

<https://leetcode.com/problems/binary-tree-level-order-traversal-ii/?envType=list&envId=xlepfebm>

| */\*\*  \* Definition for a binary tree node.  \* function TreeNode(val, left, right) {  \* this.val = (val===undefined ? 0 : val)  \* this.left = (left===undefined ? null : left)  \* this.right = (right===undefined ? null : right)  \* }  \*/* */\*\*  \* @param {TreeNode} root  \* @return {number[][]}  \*/* var levelOrderBottom = function(root) {   if(!root) return []   let queue=[root]  let res=[]   while(queue.length>0){   let size=queue.length;  let temp=[]   while(size--){   let curr=queue.shift();   temp.push(curr.val)   if(curr.left) queue.push(curr.left)  if(curr.right) queue.push(curr.right)  }  res.push(temp)   }   return res.reverse()   }; |
| --- |

### **Average of Levels in Binary Tree**

<https://leetcode.com/problems/average-of-levels-in-binary-tree/description/?envType=list&envId=xlepfebm>

| var averageOfLevels = function (root) {  if (!root) return [];  let res = [];   let queue = [root];   while (queue.length) {  let size = queue.length;  let temp = [];   while (size--) {  let curr = queue.shift();  temp.push(curr.val);  if (curr.left) queue.push(curr.left);  if (curr.right) queue.push(curr.right);  }   let sum = temp.reduce((a, b) => a + b, 0);  sum = sum / temp.length;  res.push(sum);  }   return res; }; |
| --- |

### **Binary Tree Right Side View**

<https://leetcode.com/problems/binary-tree-right-side-view/?envType=list&envId=xlepfebm>

| var rightSideView = function(root) {    let maxLevelReached = -1;  let rightView = [];    function collectRightView(node, currentLevel, rightView) {  if (!node) return;    if (maxLevelReached < currentLevel) {  rightView.push(node.val);  maxLevelReached = currentLevel;  }    *// Traverse the right subtree first*  collectRightView(node.right, currentLevel + 1, rightView);  *// Then traverse the left subtree*  collectRightView(node.left, currentLevel + 1, rightView);  }    collectRightView(root, 0, rightView);  return rightView; };6 |
| --- |

# **Trees - DFS**

### **Same Tree**

<https://leetcode.com/problems/same-tree/description/>

| */\*\*  \* @param {TreeNode} p  \* @param {TreeNode} q  \* @return {boolean}  \*/* var isSameTree = function(p, q) {    if(!p || !q) return p==q;   if(p.val!=q.val) return false   let left=isSameTree(p.left,q.left)  let right=isSameTree(p.right,q.right)   return left && right   }; |
| --- |

### **Merge Two Binary Trees**

<https://leetcode.com/problems/merge-two-binary-trees/>

| */\*\*  \* Definition for a binary tree node.  \* function TreeNode(val, left, right) {  \* this.val = (val===undefined ? 0 : val)  \* this.left = (left===undefined ? null : left)  \* this.right = (right===undefined ? null : right)  \* }  \*/* */\*\*  \* @param {TreeNode} root1  \* @param {TreeNode} root2  \* @return {TreeNode}  \*/* var mergeTrees = function(root1, root2) {   if(!root1) return root2;   if(!root2) return root1   root1.val=root1.val+root2.val;   root1.left=mergeTrees(root1.left,root2.left)  root1.right=mergeTrees(root1.right,root2.right)   return root1    }; |
| --- |

### **Path Sum**

<https://leetcode.com/problems/path-sum/>

| var hasPathSum = function(root, targetSum) {    if(!root) return false  *// check root node is value is same as targetsum*  if(!root.left && !root.right && root.val==targetSum) return true   let left=hasPathSum(root.left,targetSum-root.val);  let right=hasPathSum(root.right,targetSum-root.val)   return left || right  }; |
| --- |

### **Diameter of Binary Tree**

<https://leetcode.com/problems/diameter-of-binary-tree/>

| */\*\*  \* Definition for a binary tree node.  \* function TreeNode(val, left, right) {  \* this.val = (val===undefined ? 0 : val)  \* this.left = (left===undefined ? null : left)  \* this.right = (right===undefined ? null : right)  \* }  \*/* */\*\*  \* @param {TreeNode} root  \* @return {number}  \*/* var diameterOfBinaryTree = function(root) {   let ans=[-Infinity]   let helper=(root)=>{   if(!root) return 0   let left=helper(root.left);  let right=helper(root.right);   ans[0]=Math.max(ans[0],1+left+right)  return 1+Math.max(left,right)   }      helper(root)  *//neet to return edges so need to subtract 1*  return ans[0]-1  }; |
| --- |

Untitled

Untitled

### **Lowest Common Ancestor of a Binary Tree**

<https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/>

| */\*\*  \* @param {TreeNode} root  \* @param {TreeNode} p  \* @param {TreeNode} q  \* @return {TreeNode}  \*/* var lowestCommonAncestor = function (root, n1, n2) {  if (!root) return null;   console.log(root.val);   if (root == n1 || root == n2) return root;   let left = lowestCommonAncestor(root.left, n1, n2);  let right = lowestCommonAncestor(root.right, n1, n2);   if (left && right) return root;  return left ? left : right; }; |
| --- |

### **Invert Binary Tree**

<https://leetcode.com/problems/invert-binary-tree/>

| */\*\*  \* @param {TreeNode} root  \* @return {TreeNode}  \*/* var invertTree = function (root) {  if (!root) return null;   invertTree(root.left);  invertTree(root.right);   let temp = root.left;  root.left = root.right;  root.right = temp;   return root; }; |
| --- |

### **Path Sum II**

<https://leetcode.com/problems/path-sum-ii/description/>

| */\*\*  \* @param {TreeNode} root  \* @param {number} targetSum  \* @return {number[][]}  \*/* var pathSum = function (root, targetSum) {  let paths = [];  let ds = [];   let hepler = (root, target) => {  if (!root) return null;   ds.push(root.val);   if (!root.left && !root.right && target == root.val) {  paths.push([...ds]);  }  hepler(root.left, target - root.val);  hepler(root.right, target - root.val);   ds.pop();  };   hepler(root, targetSum);  return paths; }; |
| --- |

Untitled

### **Binary Tree Maximum Path Sum**

<https://leetcode.com/problems/binary-tree-maximum-path-sum/description/>

| */\*\*  \* @param {TreeNode} root  \* @return {number}  \*/* var maxPathSum = function (root) {  let ans = [-Infinity];   let helper = (root) => {  if (!root) return 0;   let left = Math.max(0, helper(root.left));  let right = Math.max(0, helper(root.right));   ans[0] = Math.max(ans[0], root.val + left + right);   return root.val + Math.max(left, right);  };  helper(root);   return ans[0]; }; |
| --- |

Untitled

### **Sum Root to Leaf Numbers**

<https://leetcode.com/problems/sum-root-to-leaf-numbers/>

| var sumNumbers = function (root) {  let paths = [];  let temp = [];   let helper = (root) => {  if (!root) return;   temp.push(root.val);   if (root.left == null && root.right == null) {  paths.push(Number(temp.join("")));  }   helper(root.left);   helper(root.right);  temp.pop();  };   helper(root);  *// console.log(paths)*   return paths.reduce((a, b) => a + b, 0); }; |
| --- |

### **Kth Smallest Element in a BST**

| https:*//leetcode.com/problems/kth-smallest-element-in-a-bst/description/* var kthSmallest = function (root, k) {  let count = 0;  let res = [];  let helper = (root) => {  if (!root) return;   helper(root.left);   count += 1;   if (count == k) res = root.val;  helper(root.right);  };  helper(root);   return res; }; |
| --- |

### **All Nodes Distance K in Binary Tree**

<https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/description/?envType=list&envId=xlepfebm>

| */\*\*  \* Definition for a binary tree node.  \* function TreeNode(val) {  \* this.val = val;  \* this.left = this.right = null;  \* }  \*/* */\*\*  \* @param {TreeNode} root  \* @param {TreeNode} target  \* @param {number} k  \* @return {number[]}  \*/* var distanceK = function (root, target, k) {  let map = new Map();  let visited = new Map();  let res = [];   function dfs(root) {  if (root.left) {  map.set(root.left, root);  dfs(root.left);  }   if (root.right) {  map.set(root.right, root);  dfs(root.right);  }  }   function findNodes(root, distance = 0) {  if (!root || visited.get(root)) return;   if (distance == k) {  res.push(root.val);  return;  }   visited.set(root, true);   if (map.has(root)) {  findNodes(map.get(root), distance + 1);  }   findNodes(root.left, distance + 1);  findNodes(root.right, distance + 1);  }   dfs(root);  findNodes(target);   return res; }; |
| --- |

### **Validate Binary Search Tree**

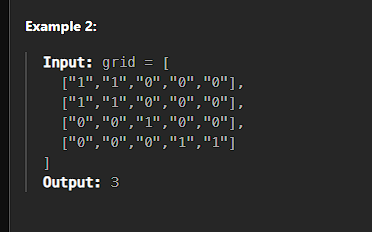
<https://leetcode.com/problems/validate-binary-search-tree/description/>

| var isValidBST = function (root, minvalue = null, maxvalue = null) {  if (!root) return true;   if (minvalue && root.val <= minvalue.val) return false;  if (maxvalue && root.val >= maxvalue.val) return false;   let left = isValidBST(root.left, minvalue, root);  let right = isValidBST(root.right, root, maxvalue);   *// console.log(left,right)*   return left && right; }; |
| --- |

## Graphs & Matrices - BFS & DFS

### **Number of Islands**

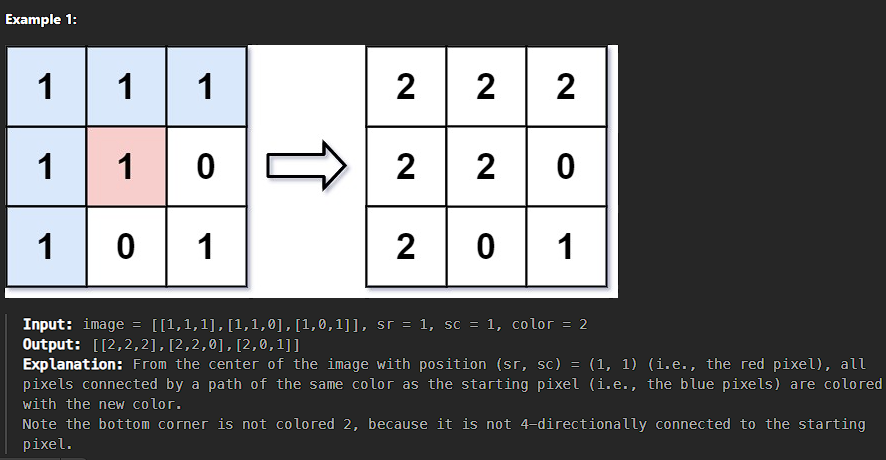
<https://leetcode.com/problems/number-of-islands/>



| var numIslands = function (grid) {  let row = grid.length;  let col = grid[0].length;   let dx = [1, -1, 0, 0];  let dy = [0, 0, 1, -1];  let count = 0;   for (let i = 0; i < row; i++) {  for (let j = 0; j < col; j++) {  if (grid[i][j] == "1") {  count++;  DFS(grid, i, j);  }  }  }   function DFS(grid, i, j) {  if (i >= 0 && j >= 0 && i < row && j < col && grid[i][j] == "1") {  console.log(i, j);  grid[i][j] = "0";  for (let k = 0; k < 4; k++) {  let x = i + dx[k];  let y = j + dy[k];  DFS(grid, x, y);  }  }  }   return count; }; |
| --- |

### **Flood Fill**

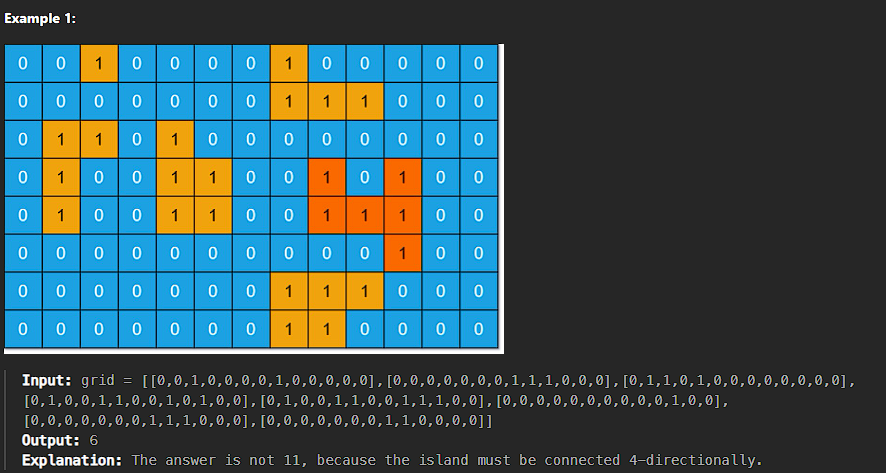
<https://leetcode.com/problems/flood-fill/description/>



| */\*\*  \* @param {number[][]} image  \* @param {number} sr  \* @param {number} sc  \* @param {number} newColor  \* @return {number[][]}  \*/* var floodFill = function (image, sr, sc, newColor) {  let row = image.length;  let col = image[0].length;   let currentColor = image[sr][sc];  if (currentColor == newColor) return image;   let dx = [1, -1, 0, 0];  let dy = [0, 0, 1, -1];   DFS(image, sr, sc);  return image;   function DFS(grid, i, j) {  console.log(i, j);  if (  i >= 0 &&  j >= 0 &&  i < row &&  j < col &&  grid[i][j] == currentColor  ) {  grid[i][j] = newColor;  for (let k = 0; k < 4; k++) {  let x = i + dx[k];  let y = j + dy[k];   DFS(grid, x, y);  }  }  } }; |
| --- |

### **Max Area of Island**

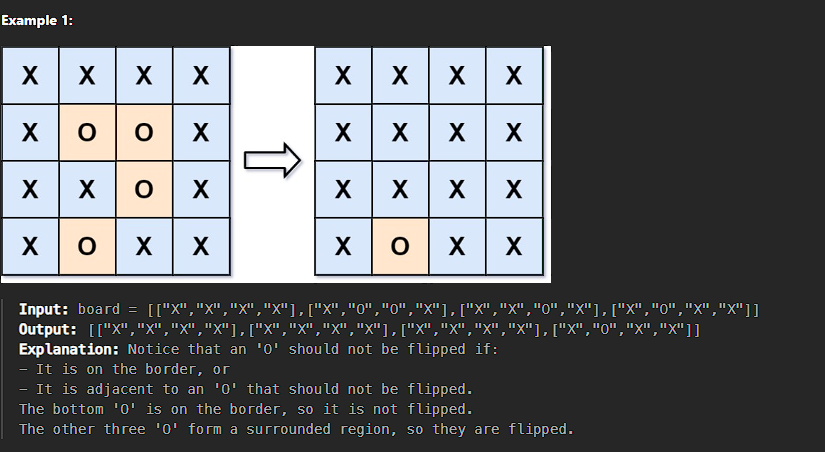
<https://leetcode.com/problems/max-area-of-island/>



| */\*\*  \* @param {number[][]} grid  \* @return {number}  \*/* var maxAreaOfIsland = function (grid) {  let row = grid.length;  let col = grid[0].length;   let dx = [1, -1, 0, 0];  let dy = [0, 0, -1, 1];  let maxCount = 0;   let count = 1;   for (let i = 0; i < row; i++) {  for (let j = 0; j < col; j++) {  if (grid[i][j] == 1) {  count = 0;  DFS(grid, i, j);  maxCount = Math.max(maxCount, count);  }  }  }   function DFS(grid, i, j) {  if (i >= 0 && j >= 0 && i < row && j < col && grid[i][j] == 1) {  count = count + 1;   grid[i][j] = "$";  for (let k = 0; k < 4; k++) {  let x = i + dx[k];  let y = j + dy[k];   DFS(grid, x, y);  }  }  }   return maxCount; }; |
| --- |

### **Surrounded Regions**

<https://leetcode.com/problems/surrounded-regions/description/>

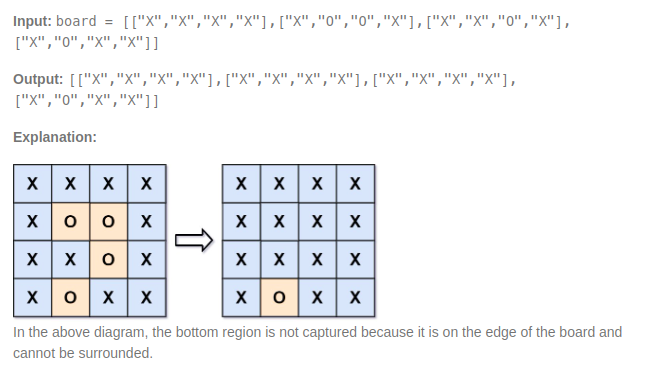


| var solve = function (board) {  let grid = board;  let row = board.length;  let col = board[0].length;   let dx = [1, -1, 0, 0];  let dy = [0, 0, 1, -1];  *//BOUNDARY dfs TRAVERSALSAL*  for (let i = 0; i < row; i++) {  for (let j = 0; j < col; j++) {  if (  (i == 0 || i == row - 1 || j == 0 || j == col - 1) &&  board[i][j] == "O"  ) {  Dfs(grid, i, j);  }  }  }   *// console.log(grid)*   for (let i = 0; i < row; i++) {  for (let j = 0; j < col; j++) {  console.log(grid[i][j]);  *//CHANGE TO ANYTHING OTHERTHAN O*  if (grid[i][j] == "V") {  grid[i][j] = "O";  *// console.log(grid[i][j],"updated")*  } else {  grid[i][j] = "X";  }  }  }  console.log(grid);   function Dfs(grid, i, j) {  if (i >= 0 && j >= 0 && i < row && j < col && grid[i][j] == "O") {  grid[i][j] = "V";   for (let k = 0; k < 4; k++) {  let x = i + dx[k];  let y = j + dy[k];   Dfs(grid, x, y);  }  }  } }; |
| --- |

### **Pacific Atlantic Water Flow**

<https://leetcode.com/problems/surrounded-regions/description/>

| */\*\*  \* @param {number[][]} heights  \* @return {number[][]}  \*/* var pacificAtlantic = function (heights) {  let pasific = new Set();  let altan = new Set();   let row = heights.length;  let col = heights[0].length;   let dx = [0, 0, -1, 1];  let dy = [1, -1, 0, 0];   for (let i = 0; i < col; i++) {  DFS(0, i, -1, pasific);  DFS(row - 1, i, -1, altan);  }   for (let i = 0; i < row; i++) {  DFS(i, 0, -1, pasific);  DFS(i, col - 1, -1, altan);  }   let res = [];   for (let i = 0; i < row; i++) {  for (let j = 0; j < col; j++) {  if (pasific.has(`${i}-${j}`) && altan.has(`${i}-${j}`)) {  res.push([i, j]);  }  }  }   return res;   function DFS(i, j, prev, ocean) {  if (!isValid(i, j, prev, ocean)) {  return;  }   ocean.add(`${i}-${j}`);   for (let k = 0; k < 4; k++) {  let x = i + dx[k];  let y = j + dy[k];   DFS(x, y, heights[i][j], ocean);  }  }   function isValid(i, j, prev, visited) {  if (  i >= 0 &&  j >= 0 &&  i < row &&  j < col &&  !visited.has(`${i}-${j}`) &&  heights[i][j] >= prev  )  return true;  return false;  } }; |
| --- |



# 

# **Trie**

### **Implement Trie (Prefix Tree)**

<https://leetcode.com/problems/implement-trie-prefix-tree/description/>

| class Trie {  constructor() {  this.root = new Map();  }   insert(word) {  let node = this.root;  for (let c of word) {  if (!node.has(c)) node.set(c, new Map());  node = node.get(c);  }  node.set("isWord", true);  }   traverse(word) {  let node = this.root;  for (let c of word) {  if (!node.has(c)) return null;  node = node.get(c);  }  return node;  }   search(word) {  const node = this.traverse(word);  return node != null && node.has("isWord");  }   startsWith(prefix) {  const node = this.traverse(word);  return node != null;  } }  *// Example usage:* const trie = new Trie(); trie.insert("apple"); console.log(trie.search("apple")); *// Output: true* console.log(trie.search("app")); *// Output: false* console.log(trie.startsWith("app")); *// Output: true* trie.insert("app"); console.log(trie.search("app")); *// Output: true* |
| --- |