|  |
| --- |
| class Solution {  fun isValid(s: String): Boolean {  if (s.length % 2 != 0) return false  val brackets = mapOf(')' to '(', '}' to '{', ']' to '[')  val opened = arrayListOf<Char> ()  for (c in s) {  if (c in brackets.values) {  opened.add(c)  } else if (opened.size == 0 || brackets.get(c) != opened.removeAt(opened.size - 1)) {  return false  }  }  return opened.isEmpty()  }  } |
| class Solution {  fun isValid(s: String): Boolean {  val stack = mutableListOf<Char> ()  for (letter in s){  when(letter){  '(' -> stack.add(')')  '[' -> stack.add(']')  '{' -> stack.add('}')  '}',']',')' -> {  if(stack.isEmpty()) return false  if(stack.removeAt(stack.lastIndex) != letter) return false  }  }  }  if(stack.isNotEmpty()) return false  return true  }  } |
| <https://leetcode.com/problems/median-of-two-sorted-arrays/>  class Solution {  fun findMedianSortedArrays(nums1: IntArray, nums2: IntArray): Double {  val sort = nums1.plus(nums2).sorted()  val mid = sort.size / 2  return if (sort.size % 2 == 0)  (sort[mid] + sort[mid-1]).toDouble() / 2  else  sort[mid].toDouble()  }  } |
| fun reverse (x: Int): Int {  if (x == 0) return 0  var sign = if (x < 0) -1 else 1  var num = 0  var origin = sign \* x  while (origin > 0) {  val tmp = origin % 10  if (num > (Int.MAX\_VALUE - tmp) / 10) {  return 0  }  num = num \* 10 + tmp  origin /= 10  }  return num \* sign  } |
| <https://leetcode.com/problems/implement-strstr/>  fun strStr(haystack: String, needle: String): Int {  var i = 0  while (true) {  var j = 0  while (true) {  if (needle.length == j) return i  if (i + j >= haystack.length) return -1  if (haystack [i + j] == needle[j]) j++ else break  }  i ++  } |
| [Longest Common Prefix - LeetCode](https://leetcode.com/problems/longest-common-prefix/)  fun longestCommonPrefix(strs: Array<String>): String {  if (strs.isEmpty()) return ""  if (strs.size == 1) return strs[0]  strs.sort()  for (i in strs[0].indices){  if (strs[0][i]! = strs[strs.size-1][i]) return strs[0].substring(0,i)  }  return strs[0]  } |
| <https://leetcode.com/problems/plus-one>  fun plusOne(digits: IntArray): IntArray {  for (i in digits.size - 1 downTo 0) {  digits[i] += 1  if (digits[i] <= 9) return digits  digits[i] = 0  }  val arr = IntArray(digits.size + 1)  arr[0] = 1  return arr  } |
| <https://leetcode.com/problems/maximum-subarray>  fun maxSubArray(nums: IntArray): Int {  var currSum = nums[0]  var prevSum:Int  var maxSum = currSum  for (i in 1 until nums.size) {  prevSum = currSum  if (prevSum < 0) {  prevSum = 0  }  currSum = nums[i] + prevSum  maxSum = Math.max(maxSum,currSum)  }  return maxSum  } |
| <https://leetcode.com/problems/symmetric-tree>  private fun areNodeMirror(left: TreeNode?, right: TreeNode?): Boolean {  if (left == null && right == null) {  return true  }  if (left! = null && right != null && left.`val` == right.`val`) {  return areNodeMirror(left.left, right.right)  && areNodeMirror(left.right, right.left)  }  return false  } |
| <https://leetcode.com/problems/search-insert-position/>  fun searchInsert(nums: IntArray, target: Int): Int {  var i = 0  while (i < nums.size) {  if (nums[i] == target) return i  if (i == 0 && target < nums[i]) return i  if (i > 0 && target > nums[i-1] && target < nums[i]) return i  i++  }  return i  } |
| <https://leetcode.com/problems/climbing-stairs/>  fun climbStairs(n: Int): Int {  val dp = IntArray(n + 1)  dp[0] = 1  dp[1] = 1  for (i in 2..n) {  dp[i] = dp[i - 1] + dp[i - 2]  }  return dp[n]  } |
| [Roman to Integer - LeetCode](https://leetcode.com/problems/roman-to-integer/)  fun romanToInt(s: String): Int {  val map = mutableMapOf(  'I' to 1, 'V' to 5, 'X' to 10, 'L' to 50, 'C' to 100, 'D' to 500, 'M' to 1000  )  var number = 0  var last = 1000  s.forEach {  val value = map[it] ?: 0  if (value > last) number -= last \* 2  number += value  last = value  }  return number  } |
| <https://leetcode.com/problems/best-time-to-buy-and-sell-stock/>  class Solution {  fun maxProfit(prices: IntArray): Int {  var max = 0  var min = Int.MAX\_VALUE  prices.forEach {  if (it< min) {  min = it  } else {  max = Math.max(max, it - min)  }  }  return max  }  } |
| [Happy Number - LeetCode](https://leetcode.com/problems/happy-number/)  class Solution {  fun isHappy(n: Int): Boolean {  var myNum = n  val set = mutableSetOf<Int> ()  while (!set.contains(myNum) && myNum != 1) {  set.add(myNum)  val digits = myNum.toDigits()  var sum = 0  digits.forEach {  sum = sum + (it \* it)  }  myNum = sum  }  return myNum == 1  }  fun Int.toDigits(): List<Int> = toString().map { it.toString().toInt() }  } |
| <https://leetcode.com/problems/add-digits/>  fun addDigits(num: Int): Int {  var result = num  while (result.toString().length > 1) {  result = result.toString().sumBy { it.toString().toInt() }  }  return result  }  -------------------  class Solution {  fun addDigits(num: Int): Int {  var n = num  while(true){  if (n<10){  return n  }  n = n.toString().split("").filter{it->it!=""}.map{it->it.toInt()}.sum()  }  }  }  -----------  class Solution {  fun addDigits(num: Int): Int {  var copy = num  while (copy >= 10){  val digits = copy.toString().map {it.toString().toInt()}  copy = digits.sum()  }  return copy  }  } |
|  |
|  |
|  |
|  |