# array

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 121. Best Time to Buy and Sell Stock | [Time Limit Exceeded] |
| 题目 | |
| Say you have an array for which the ith element is the price of a given stock on day i.  Input: [7, 1, 5, 3, 6, 4]  Output: 5  max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)  Input: [7, 6, 4, 3, 1]  Output: 0  In this case, no transaction is done, i.e. max profit = 0. | |
| 自己代码（暴力破解时间复杂度过高） | |
| public int maxProfit(int[] prices) {  int result=0;  for(int i=0;i<prices.length-1;i++){  for(int j=i;j<prices.length;j++){  int temp=prices[j]-prices[i];  if(temp<=0)  continue;  else  result=result>=temp?result:temp;  }  }  return result;  } | |
| 优质代码 | |
| public int maxProfit(int prices[]) {  int minprice = Integer.MAX\_VALUE;  int maxprofit = 0;  for (int i = 0; i < prices.length; i++) {  if (prices[i] < minprice)  minprice = prices[i];  else if (prices[i] - minprice > maxprofit)  maxprofit = prices[i] - minprice;  }  return maxprofit;  }  /\*先存个最小值，再用后面的数跟最小值比，得出差作为利润，利润比原来大就更新利润\*/ | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 485. Max Consecutive Ones |  |
| 题目 | |
| Given a binary array, find the maximum number of consecutive 1s in this array.  Example 1:  Input: [1,1,0,1,1,1]  Output: 3  Explanation: The first two digits or the last three digits are consecutive 1s.  The maximum number of consecutive 1s is 3. | |
| 自己代码 | |
| public int findMaxConsecutiveOnes(int[] nums) {  int count1=0;  int count2=0;  for(int i=0;i<nums.length;i++) {  if(nums[i]==1)  count1++;  if(nums[i]==0){  count2=count1>=count2?count1:count2; //留下较大的，这样才能更新  count1=0;  }  }  // System.out.println(count1>=count2?count1:count2);  return count1>=count2?count1:count2;  } | |
| 优质代码 | |
|  | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 53. Maximum Subarray |  |
| 题目 | |
| Find the contiguous subarray within an array (containing at least one number) which has the largest sum  For example, given the array [-2,1,-3,4,-1,2,1,-5,4],  the contiguous subarray [4,-1,2,1] has the largest sum = 6. | |
| 自己代码 | |
| public int maxSubArray(int[] nums) {  int maxSum=Integer.MIN\_VALUE;  int thisSum=0;  for(int i=0;i<nums.length;i++)  {  thisSum+=nums[i];  if(thisSum>maxSum)  maxSum=thisSum;  if(thisSum<0);  //如果当前的的和是负数的话，那么下一个数组元素是不管正负都会会导致和更小，所以应当置为0重新累加  thisSum=0;  }  System.out.println(maxSum);  return maxSum;  } | |
| 优质代码 | |
|  | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 345. Reverse Vowels of a String | Time Limit Exceeded |
| 题目 | |
| Write a function that takes a string as input and reverse only the vowels of a string.  Example 1:  Given s = "hello", return "holle".  Example 2:  Given s = "leetcode", return "leotcede". | |
| 自己代码 | |
| public String reverseVowels(String s) {  ArrayList<Character> list=new ArrayList<>();  list.add('a');  list.add('e');  list.add('i');  list.add('o');  list.add('u');  list.add('A');  list.add('E');  list.add('I');  list.add('O');  list.add('U');  char str[]=s.toCharArray();  char tem;  for(int i=0;i<str.length-1;i++)//暴力破解，遇到前后有aeiou的就换  {  for(int j=i+1;j<str.length;j++)  {  if(list.contains(s.charAt(i))&&list.contains(s.charAt(j)))  {  tem=str[i];  str[i]=str[j];  str[j]=tem;  }  }  }  String newstr="";  for(char e:str)  newstr=newstr+e;  // System.out.println(newstr);  return newstr;  } | |
| 优质代码 | |
| 迭代字符串的元素，找到第n个元音字母后，就尝试去找第len-1-n个元音字母，如果找到就交换位置，再进入下一次循环迭代；如果没找到，则退出循环。代码很像快速排序中处理第n个和第len-1-n个元素的方式。  public String reverseVowels(String s) {  String vowels = "aoeiuAOEIU";  char[] a = s.toCharArray();  int i = 0;  int j = a.length - i - 1;  while (i < j) {  while (i < j && !vowels.contains(a[i] + "")) {//String的contains方法必须是String类型，所以加个空字符串  i++;  }  while (i < j && !vowels.contains(a[j] + "")) {  j--;  }  if (i < j) {  char c = a[i];  a[i++] = a[j];  a[j--] = c;  }  }  return new String(a);  } | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 167. Two Sum II - Input array is sorted | Time Limit Exceeded |
| 题目 | |
| Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.  The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.  You may assume that each input would have exactly one solution and you may not use the same element twice.  Input: numbers={2, 7, 11, 15}, target=9  Output: index1=1, index2=2 | |
| 自己代码 | |
| public class Solution {  public int[] twoSum(int[] numbers, int target) {  int res[]=new int[2];  for(int i=0;i<numbers.length-1;i++){  for(int j=i+1;j<numbers.length;j++)  {  if(numbers[i]+numbers[j]==target)  {  res[0]=i+1;  res[1]=j+1;  break;  }  }  }  return res;  }  } | |
| 优质代码 | |
| public class Solution {  public int[] twoSum(int[] numbers, int target) {  int res[]=new int[2];  int left=0;  int right=numbers.length-1;  while(left<right)  {  if(numbers[left]+numbers[right]==target)  {  res[0]=left+1;  res[1]=right+1;  break;  }  if(numbers[left]+numbers[right]<target)  left++;  else  right--;  }  return res;  }  }  /\*题目本身并不难，但是如果用两个for循环的话时间复杂度过高，导致超时，而像例子这么做的话，用两个指针来移动指示，只用一次扫描，时间就下来了\*/ | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
|  |  |
| 题目 | |
|  | |
| 自己代码 | |
|  | |
| 优质代码 | |
|  | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
|  |  |
| 题目 | |
|  | |
| 自己代码 | |
|  | |
| 优质代码 | |
|  | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
|  |  |
| 题目 | |
|  | |
| 自己代码 | |
|  | |
| 优质代码 | |
|  | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 11. Container With Most Water | 88% |
| 题目 | |
| Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.相当于坐标的X轴上有一些竖线，求其中的两条竖线与X轴围成的面积最大 | |
| 自己代码 | |
| public class Solution {  public int maxArea(int[] height) {  int maxSize=0;  for(int i=0,j=height.length-1;i<height.length;i++)  {  if(height[i]<=height[j])  maxSize=maxSize>height[i]\*(j-i)?maxSize:height[i]\*(j-i);  else {  maxSize=maxSize>height[j]\*(j-i)?maxSize:height[j]\*(j-i);  j--;  i--;  }  }  // System.out.println(maxSize);  return maxSize;  }  }  让水自己满上来，从第一条竖线开始判断，另一个指针j从数组的最后计数，如果第i位置的那条竖线比j位置低，说明以i位置的竖线为准，否者以j位置的为准，要注意的是变换准线时，i位置要等一步j位置（即为什么要i--）,否则i位置就会少比较一次 | |
| 优质代码 | |
|  | |

# String

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 13. Roman to Integer |  |
| 题目 | |
| Given a roman numeral, convert it to an integer.  Input is guaranteed to be within the range from 1 to 3999. | |
| 自己代码 | |
| 未解答 | |
| 优质代码 | |
| public int romanToInt(String s) {  int res = 0;  for (int i = s.length() - 1; i >= 0; i--) {  char c = s.charAt(i);  if(c == 'I'){  if(res >= 5)  //如果>=5, 说明之前肯定遍历过V了，所以这个I肯定在左边，减  res += -1;  else  res += 1;  }else if(c == 'V'){//遇见V,L,D,M,统统都加5，50，500，100  res += 5;  }else if(c == 'X'){  if(res >= 50)//说明肯定之前有过L，这个X肯定在左边，减  res += -10;  else  res += 10;  }else if(c == 'L'){  res += 50;  }else if(c == 'C'){  //说明之前有D，这个C肯定在左边，减。能被减的只有I X C  if(res >= 500)  res += -100;  else  res += 100;  }else if(c == 'D'){  res += 500;  }else if(c == 'M'){  res += 1000;  }  }  return res; | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 12. Integer to Roman |  |
| 题目 | |
| Given an integer, convert it to a roman numeral.  Input is guaranteed to be within the range from 1 to 3999. | |
| 自己代码 | |
|  | |
| 优质代码 | |
| public String intToRoman(int num) {  String c[][]={  {"","I","II","III","IV","V","VI","VII","VIII","IX"},  {"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"},  {"","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"},  {"","M","MM","MMM"}  };  StringBuffer roman=new StringBuffer();  roman.append(c[3][num / 1000 % 10]);//拼凑千位  roman.append(c[2][num / 100 % 10]);  roman.append(c[1][num / 10 % 10]);  roman.append(c[0][num % 10]);  //System.out.println(roman);  return roman.toString();  } | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 205. Isomorphic Strings | [Time Limit Exceeded] |
| 题目 | |
| 同构字符串  For example,  Given "egg", "add", return true.  Given "foo", "bar", return false.  Given "paper", "title", return true. | |
| 自己代码 | |
| public class Solution {  public boolean isIsomorphic(String s, String t) {  String com1="";  String com2="";  int count1=0;  int count2=0;  if(s.length()==0||s.length()==1)  return true;  for (int i = 0; i <s.length()-1; i++) {  if(s.charAt(i)!=s.charAt(i+1))  count1++;  com1+=Integer.toString(count1);  }  for (int i = 0; i <t.length()-1; i++) {  if(t.charAt(i)!=t.charAt(i+1))  count2++;  com2+=Integer.toString(count2);  }    // System.out.println(com1);  // System.out.println(com2);  if(com1.equals(com2))  return true;  else  return false;  }//维护两个字符串，遇见不同的字母就变一个数字，最后的字符串相同就同构  } | |
| 优质代码 | |
| public class Solution {  public boolean isIsomorphic(String s, String t) {  if(s.length()==0||s.length()==1)  return true;  Map<Character, Character> map = new HashMap<Character, Character>();  Set<Character> set = new HashSet<Character>(); //set是无序的，且不允许重复    for(int i=0; i<s.length(); i++) {  char c1 = s.charAt(i);  char c2 = t.charAt(i);    if(map.containsKey(c1)) //c1有映射值  {  if(map.get(c1) != c2) //但是c1的映射值不是c2，说明不对应  return false;  }  else  {  if(set.contains(c2))  //c2已经存到set里面了，但是map中又没有映射c2，说明不对应  return false;  else  {  map.put(c1, c2); //map中有映射，set中就有元素  set.add(c2);  }  }  }  return true;  }  } | |

|  |  |
| --- | --- |
| 题号&标题 | 效率 |
| 3. Longest Substring Without Repeating Characters |  |
| 题目 | |
| Given a string, find the length of the longest substring without repeating characters.  Examples:  Given "abcabcbb", the answer is "abc", which the length is 3.  Given "bbbbb", the answer is "b", with the length of 1.  Given "pwwkew", the answer is "wke", with the length of 3. Note that the answer must be a substring, "pwke" is a subsequence and not a substring | |
| 自己代码 | |
|  | |
| 优质代码 | |
| public class Solution {  public int lengthOfLongestSubstring(String s) {  if (s.length()==0)  return 0;  HashMap<Character, Integer> map = new HashMap<Character, Integer>();  int max=0;  for (int i=0, j=0; i<s.length(); ++i)  {  if (map.containsKey(s.charAt(i)))  {  j = Math.max(j,map.get(s.charAt(i))+1);  }  map.put(s.charAt(i),i);  max = Math.max(max,i-j+1);  }  // System.out.println(max);  return max;  }  }  //维护一个map，和一个指针j，当i不断往后移动，碰到一个map中有的char就看看指针j的位置，返回差值 | |