春节7天练讲Day3：排序和二分查找

你好，我是王争。初三好！

为了帮你巩固所学，真正掌握数据结构和算法，我整理了数据结构和算法中，必知必会的30个代码实现，分7天发布出来，供你复习巩固所用。今天是第三篇。

和昨天一样，你可以花一点时间，来完成测验。测验完成后，你可以根据结果，回到相应章节，有针对性地进行复习。

前两天的内容，是关于数组和链表、排序和二分查找的。如果你错过了，点击文末的“上一篇”，即可进入测试。

关于排序和二分查找的几个必知必会的代码实现

排序

- 实现归并排序、快速排序、插入排序、冒泡排序、选择排序
- 编程实现O(n)时间复杂度内找到一组数据的第K大元素

二分查找

- 实现一个有序数组的二分查找算法
- 实现模糊二分查找算法（比如大于等于给定值的第一个元素）

对应的LeetCode练习题（@Smallfly 整理）

- Sqrt(x) （x 的平方根）

英文版：https://leetcode.com/problems/sqrtx/

中文版：https://leetcode-cn.com/problems/sqrtx/

做完题目之后，你可以点击“请朋友读”，把测试题分享给你的朋友，说不定就帮他解决了一个难题。

祝你取得好成绩！明天见！

精选留言

李皮皮皮皮皮

各种排序算法真要说起来实际中使用的最多的也就是快排了。然而各种编程语言内置的标准库都包含排序算法的实现，基本没有自己动手实现的必要。然后作为经典的算法，自己实现一遍，分析分析时间空间复杂度对自己的算法设计大有裨益。需要注意的是为了高效，在实际的实现中，多种排序算法往往是组合使用的。例如c标准库中总体上是快排，但当数据量小于一定程度，会转而使用选择或插入排序。

求平方根使用牛顿法二分逼近

2019-02-06 20:27

吴...

虽然现在有很多排序算法自己不会亲自写，但是作为算法的基础，分治，归并，冒泡等排序算法在时间复杂度，空间复杂度以及原地排序这些算法知识上的理解非常有帮助。递归分治这些算法思想在简单的算法中也能体现出来，其实更多的是思维方式的训练。

2019-02-07 23:05

虎虎

基本排序算法的关注点分为：

1. 时间复杂度。如n的平方（冒泡，选择，插入）；插入排序的优化希尔排序，则把复杂度降低到n的3/2次方；n乘以logn(快排，归并排序，堆排序）。

2. 是否为原地排序。如，归并排序需要额外的辅助空间。

3. 算法的稳定性。稳定排序（by nature）如冒泡，插入，归并。如果把次序考虑在内，可以把其他的排序（如快排，堆排序）也实现为稳定排序。

4. 算法的实现。同为时间复杂度同为n平方的算法中，插入排序的效率更高。但是如果算法实现的不好，可能会降低算法的效率，甚至让稳定的算法变得不稳定。又如，快速排序有不同的实现方式，如三路快排可以更好的应对待排序数组中有大量重复

元素的情况。堆排序可以通过自上而下的递归方式实现，也可以通过自下而上的方式实现。
5. 不同算法的特点，如对于近乎有序的数组进行排序，首选插入排序，时间复杂度近乎是n，而快速排序则退化为n平方。

二分查找，需要注意 (l+r)/2可能存在越界问题。

leetcode题，用二分查找找到x*x > n 且(x-1)的平方小于n的数，则n-1就是结果。或者 x的平方小于n且x+1的平方大于n,则返回x。

2019-02-07 12:50

abner

java实现冒泡排序
代码如下：

```
package sort;

public class BubbleSort {

public int[] bubbleSort(int[] array) {
for (int i = 0;i < array.length - 1;i++) {
for (int j = 0;j < array.length - i - 1;j++) {
if (array[j] > array[j + 1]) {
int temp = array[j + 1];
array[j + 1] = array[j];
array[j] = temp;
}
}
}
return array;
}

public static void main(String[] args) {
int[] array = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
BubbleSort bubbleSort = new BubbleSort();
int[] result = bubbleSort.bubbleSort(array);
for (int i = 0;i < result.length;i++) {
System.out.print(result[i] + " ");
}
}

}
```

2019-02-11 17:44

kai

实现模糊二分查找算法2:

```
public class BinarySearch {
// 3. 查找第一个大于等于给定值的元素
public static int bsFistGE(int[] array, int target) {
int lo = 0;
int hi = array.length - 1;

while (lo <= hi) {
int mid = lo + ((hi - lo) >> 1);
```

```
if (array[mid] >= target) {
if (mid == 0 || array[mid-1] < target) {
return mid;
} else {
hi = mid - 1;
}
} else {
lo = mid + 1;
}
}

return -1;
}

// 4. 查找最后一个小于等于给定值的元素
public static int bsLastLE(int[] array, int target) {
int lo = 0;
int hi = array.length - 1;

while (lo <= hi) {
int mid = lo + ((hi - lo) >> 1);

if (array[mid] <= target) {
if (mid == hi || array[mid+1] > target) {
return mid;
} else {
lo = mid + 1;
}
} else {
hi = mid - 1;
}
}

return -1;
}
}
```

2019-02-11 10:32

kai

实现模糊二分查找算法1:

```
public class BinarySearch {

// 1. 查找第一个值等于给定值的元素
public static int bsFirst(int[] array, int target) {
int lo = 0;
int hi = array.length - 1;

while (lo <= hi) {
int mid = lo + ((hi - lo) >> 1);
```

```
if (array[mid] > target) {
hi = mid - 1;
} else if (array[mid] < target) {
lo = mid + 1;
} else {
if (mid == lo || array[mid-1] != array[mid]) {
return mid;
} else {
hi = mid - 1;
}
}
}

return -1;
}

// 2. 查找最后一个值等于给定值的元素
public static int bsLast(int[] array, int target) {
int lo = 0;
int hi = array.length - 1;

while (lo <= hi) {
int mid = lo + ((hi - lo) >> 1);

if (array[mid] > target) {
hi = mid - 1;
} else if (array[mid] < target) {
lo = mid + 1;
} else {
if (mid == hi || array[mid] != array[mid+1]) {
return mid;
} else {
lo = mid + 1;
}
}
}

return -1;
}
}
```

2019-02-11 10:31

kai

实现一个有序数组的二分查找算法:

```
public class BinarySearch {
// 最简单的二分查找算法：针对有序无重复元素数组
// 迭代
public static int binarySearch(int[] array, int target) {
if (array == null) return -1;

int lo = 0;
```

```
int hi = array.length-1; // 始终在[lo, hi]范围内查找target

while (lo <= hi) {
int mid = lo + ((hi - lo) >> 1); // 这里若是 (lo + hi) / 2 有可能造成整型溢出

if (array[mid] > target) {
hi = mid - 1;
} else if (array[mid] < target) {
lo = mid + 1;
} else {
return mid;
}
}

return -1;
}

// 递归
public static int binarySearchRecur(int[] array, int target) {
if (array == null) return -1;
return bs(array, target, 0, array.length-1);
}

private static int bs(int[] array, int target, int lo, int hi) {
if (lo <= hi) {
int mid = lo + ((hi - lo) >> 1);
if (array[mid] > target) {
return bs(array, target, lo, mid-1);
} else if (array[mid] < target) {
return bs(array, target, mid+1, hi);
} else {
return mid;
}
}

return -1;
}
}
```

2019-02-11 10:29

黄丹

王争老师初三快乐！

这是今天两道题的解题思路和代码

1. O(n)时间内找到第K大的元素：

解题思路：利用快排中分区的思想，选择数组区间A[0...n-1]的左右一个元素A[n-1]作为pivot，对数组A[0...n-1]原地分区，这样数组就分成了三部分，A[0..p-1],A[p],A[p+1...n-1],如果p+1=k,那么A[p]就是要求解的元素，如果K>p+1,则说明第K大的元素在A[p+1...n-1]这个区间，否则在A[0...p-1]这个区间，递归的查找第K大的元素

2. Sqrt(x) （x 的平方根）

解题思路：利用二分查找的思想，从1到x查找x的近似平方根

代码：

https://github.com/yyxd/leetcode/blob/master/src/leetcode/sort/Problem69_Sqrt.java

2019-02-07 18:35

C_love

Use Binary Search

```
class Solution {
public int mySqrt(int x) {
if (x == 0 || x == 1) {
return x;
}

int start = 0;
int end = (x >> 1) + 1;

while (start + 1 < end) {
final int mid = start + ((end - start) >> 1);
final int quotient = x / mid;
if (quotient == mid) {
return mid;
} else if (quotient < mid) {
end = mid;
} else {
start = mid;
}
}

return start;
}
}
```

2019-02-07 13:00

失火的夏天

牛顿法或者二分逼近都可以解决平方根问题，leetcode上有些大神的思路真的很厉害，经常醍醐灌顶

2019-02-06 22:55

吴...

```
#include<iostream>
#include<cmath>
using namespace std;
double a = 1e-6;
double sqrt(double n){
double low = 0.0;
double high = n;

int i = 1000;

while(i--){
double mid = low + (high - low) / 2.0;
//cout<<"n:"<<n<<endl;
double square = mid * mid;
//cout<<"sq:"<<square<<endl;
//cout<<"s:"<<abs(square - n)<<endl;
if(abs(mid * mid - n) < a){
return mid;
}
else{
```

```
if(square > n){
high = mid;
}
else{
low = mid;
}
}
}
return -2.0;
}
int main(){
double t;
while(true){
cin>>t;
cout<<sqrt(t)<<endl;
}
}
```

2019-02-14 11:34

Monster

```
/**
* O(n)时间复杂度内求无序数组中第K大元素
*/
public class TopK {

public int findTopK(int[] arr, int k) {
return findTopK(arr, 0, arr.length - 1, k);
}

private int findTopK(int[] arr, int left, int right, int k) {
if (arr.length < k) {
return -1;
}
int pivot = partition(arr, left, right);
if (pivot + 1 < k) {
findTopK(arr, pivot + 1, right, k);
} else if (pivot + 1 > k) {
findTopK(arr, left, pivot - 1, k);
}
return arr[pivot];
}


private int partition(int[] array, int left, int right) {
int pivotValue = array[right];
int i = left;

///小于分区点放在左边 大于分区点放在右边
for (int j = left; j < right; j++) {
if (array[j] < pivotValue) {
int tmp = array[i];
```

```
array[i] = array[j];
array[j] = tmp;
i++;
}
}
//与分区点交换
int tmp = array[i];
array[i] = array[right];
array[right] = tmp;
return i;
}
}
```

2019-02-13 18:22

Eid

编程实现 O(n) 时间复杂度内找到一组数据的第 K 大元素。

这个的时间复杂路应该是n·logk吧?

2019-02-12 19:05

纯洁的憎恶

这道题似乎可以等价于从1到x中找到一个数y，使得y*y小于等于x，且（y+1）*（y+1）大于x。那么可以从1到x逐个尝试，提高效率可以采用二分查找方法，时间复杂度为O（logx）。

2019-02-09 17:00

molybdenum

老师新年好~这是我的作业

https://blog.csdn.net/github_38313296/article/details/86818929

2019-02-09 16:21

ALAN

```
// find the k-th biggest number
public int heapsort(int[] arr, int k) {
// build minimum heap
for (int i = 1; i <= k; i++) {
while (i / 2 > 0 && arr[i] < arr[i / 2]) { // 自下往上堆化
swap(arr, i, i / 2); // swap() 函数作用：交换下标为 i 和 i/2 的两个元素
i = i / 2;
}
}
// replace the heap top element with the new element and heapify
for (int i = k; i < arr.length; i++) {
if (arr[i] <= arr[1])
continue;
else {
arr[1] = arr[i];
heapify(arr, k, 1);
}
}
return arr[1];
}

public void swap(int[] arr, int j, int k) {
int temp = arr[j];
arr[j] = arr[k];
arr[k] = temp;
}
```

```
private void heapify(int[] a, int n, int i) { // 自上往下堆化
while (true) {
int minPos = i;
if (i * 2 <= n && a[i] > a[i * 2])
minPos = i * 2;
if (i * 2 + 1 <= n && a[minPos] > a[i * 2 + 1])
minPos = i * 2 + 1;
if (minPos == i)
break;
swap(a, i, minPos);
i = minPos;
}
}
```

2019-02-08 21:34

ALAN

sort answer:

```
// guibing sort
public int[] gbsort(int[] arr, int start, int end) {

if (start == end)
return new int[] { arr[start] };
int[] l1 = gbsort(arr, start, (start + end) / 2);
int[] r1 = gbsort(arr, (start + end) / 2 + 1, end);
return merge(l1, r1);

}

// merge
public int[] merge(int[] a, int[] b) {
int[] c = new int[a.length + b.length];
int j = 0, k = 0;
for (int i = 0; i < a.length + b.length; i++) {
if (j == a.length) {
c[i] = b[k];
k++;
continue;
} else if (k == b.length) {
c[i] = a[j];
j++;
continue;

}
if (a[j] < b[k]) {
c[i] = a[j];
j++;
} else {
c[i] = b[k];
k++;
}
```

```
}

return c;
}

// quick sort
public void qsort(int[] arr, int start, int end, int index) {
// compare and swap
if (start >= end)
return;
int j = start, k = end;
int value = (start + end) / 2;
while (j < k) {
for (; j < k; k--) {
if (k <= value)
break;
else if (arr[k] <= arr[value]) { // from small to big >=
continue;
} else {
// swap
int temp = arr[k];
arr[k] = arr[value];
arr[value] = temp;
value = k;
break;
}

}
for (; j < k; j++) {
if (j >= value)
break;
else if (arr[j] >= arr[value]) { // from small to big <=
continue;
} else {
// swap
int temp = arr[j];
arr[j] = arr[value];
arr[value] = temp;
value = j;
break;
}

}
}

qsort(arr, start, j, index);
qsort(arr, k + 1, end, index);
}
```

2019-02-08 21:34

老杨同志

```
//平方根
//递归的话会栈溢出
//迭代法，要处理好溢出的问题，开始以为溢出时结果是负数，实测并非如此。
public int sqrtLoop(int x) {
return _mySqrtLoop(x,0,x/2+1);
}
private int _mySqrtLoop(int x, int l, int r) {
while(l<r){
int m = l+(r-l)/2;
long tmp = (long)m * (long)m;
if(tmp==x||(tmp<x && (m+1)*(m+1)>x)){
return m;
}else if(tmp>x){
r = m - 1;
}else{
l = m + 1;
}
}
return l;
}
```

2019-02-08 17:00

老杨同志

```
package com.jxyang.test.geek.day3;
//数组中求第k大的元素
public class BigK {
public static void main(String[] args) {
int[] arr = {3,5,6,9,7,4,2,1,11,16};
BigK bigK = new BigK();
System.out.println(bigK.findBigK(arr,10));
}
private int findBigK(int[] arr, int k) {
if(k>arr.length||arr==null){
return -1;
}
int m = partition(arr,0,arr.length-1,k-1);
return arr[m];
}
/*
[l,r]处理数组l到r的闭区间
[l+1,j] 小于arr[l],[j+1,i] 大于arr[l]
循环结束，arr[l] 与 arr[j] 交换
返回j
*/
private int partition(int[] arr, int l, int r,int k) {
//结束条件
if(l==r)
return l;
int j =l;
```

```
for(int i=l+1;i<=r;i++){
if(arr[i]<arr[l]){
j++;
int tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
}else{
continue;
}
}
int tmp = arr[l];
arr[l] = arr[j];
arr[j] = tmp;
while(j!=k){
if(j<k){
j = partition(arr,j+1,r,k);
}else{
j = partition(arr,l,j-1,k);
}
}
return k;
}
}
```

2019-02-08 11:30

你看起来很好吃

求平方根用python实现，基于二分查找法思想：

```
class Solution:
def mySqrt(self, x: 'int') -> 'int':
if x == 0:
return 0

left, right = 1, x

while True:
mid = (left + right) // 2
if mid * mid > x:
right = mid
else:
if (mid + 1) * (mid +1) > x:
return mid
left = mid
```

2019-02-08 07:42