**算法设计与分析第三章作业**

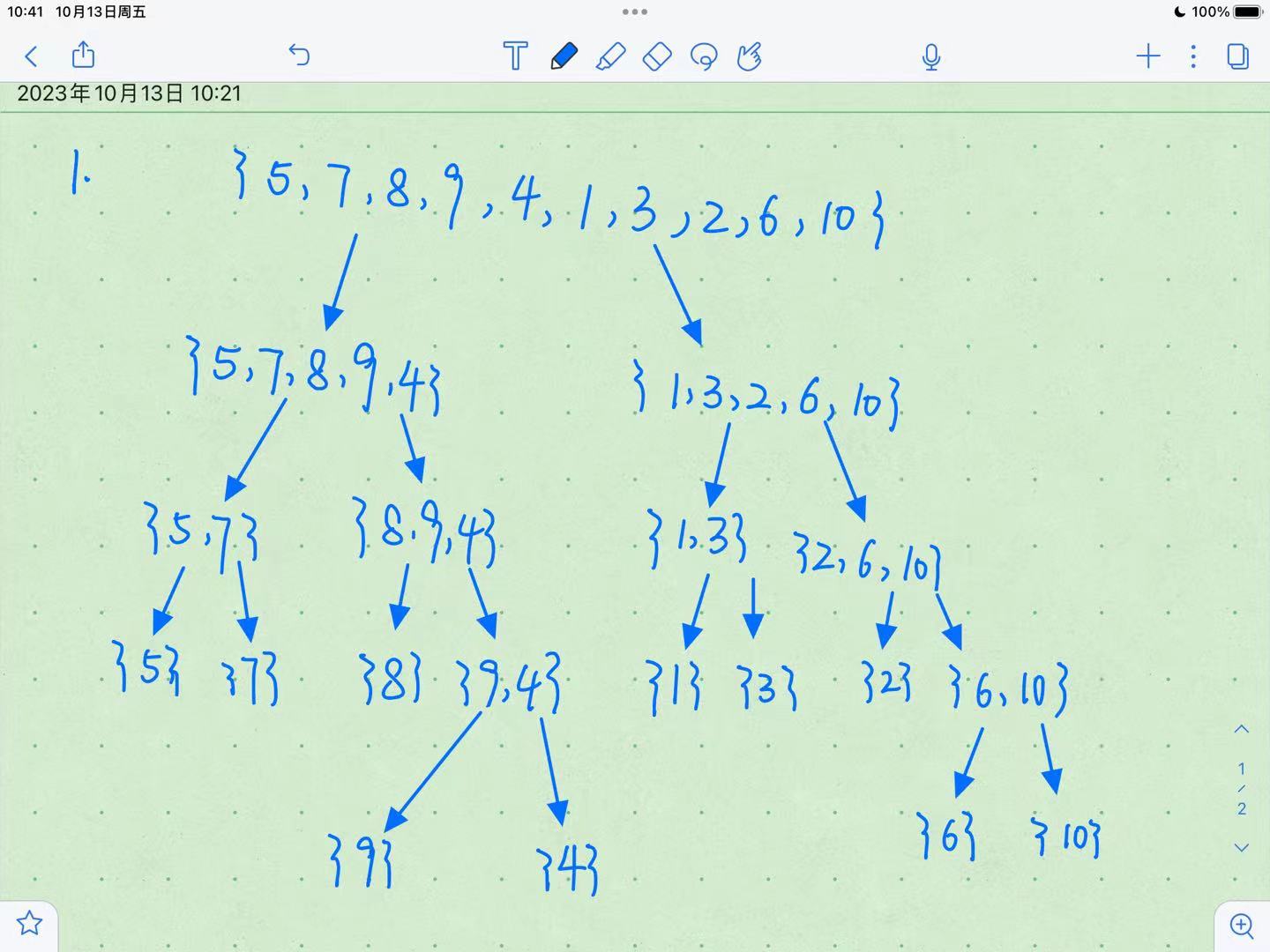
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **周健毅** | **班级** | **7** | **学号** | 220110713 |
| **第1题** |  | | | | |
| **第2题** |  | | | | |
| **第3题** |  | | | | |
| **第4题** |  | | | | |
| **第5题** |  | | | | |
| **总分** |  | | | | |
| **备注** | 作业提交截止时间：2023-10-14日24:00，超过提交截至时间的作业视为无效。作业提交邮箱：[272181768@QQ.com。](mailto:hitcsalgo2020@163.com。作业的首页要注明：学生姓名，班级，学号)作业文件名命名方式： 第x章-x班-姓名-学号（例， 第1章-1班-张三-2018054000.doc）； 邮件主题为：第x章作业, x班，姓名，学号（例， 第1章作业，1班，张三，2018054000）。缺少这些信息的作业将被酌情扣分。 | | | | |

1. 将数组进行归并排序，并通过递归树方法计算归并排序时间复杂度。**（该题考察归并排序以及时间复杂度，15分）**

给定数组 a[10]={5,7,8,9,4,1,3,2,6,10}：

* 1. 分治：
     1. 将数组分为两部分：{5,7,8,9,4} 和 {1,3,2,6,10}。
     2. 将每部分再次分为两部分：{5,7} 和 {8,9,4}；{1,3} 和 {2,6,10}
     3. 继续分割：{5}、{7}、{8}、{9,4}；{1}、{3}、{2}、{6,10}
     4. 继续分割：{9}、{4}、{6}、{10}

具体步骤如图：

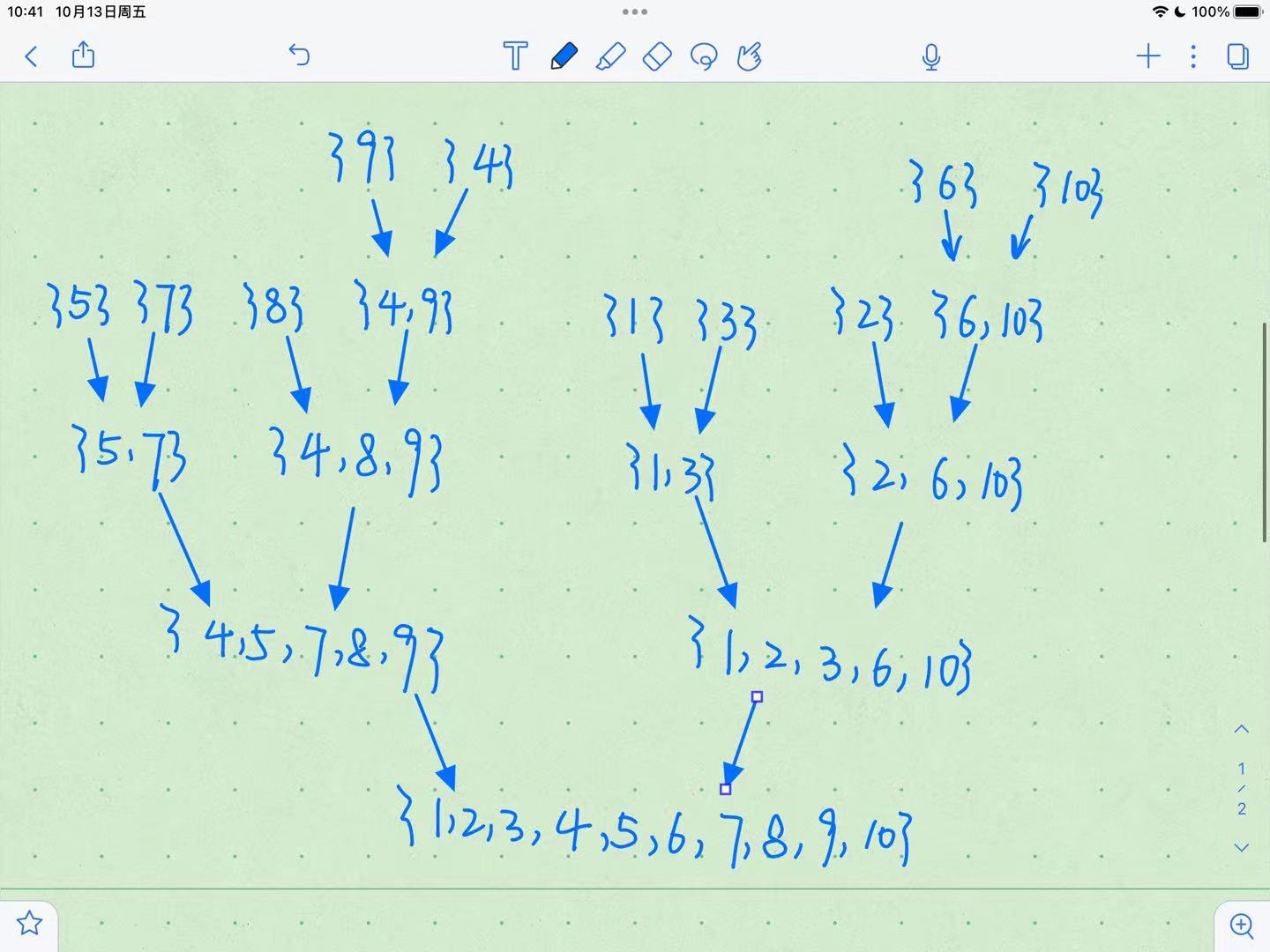


* 1. 归并：
     1. {9},{4} 归并为 {4,9}；{6},{10} 归并为 {6,10}
     2. {5},{7} 归并为 {5,7}；{8}{4,9} 归并为 {4,8,9}；

{1},{3} 归并为 {1,3}；{2},{6,10} 归并为 {2,6,10}

* + 1. {5,7}{4,8,9} 归并为 {4,5,7,8,9}；{1,3},{2,6,10} 归并为 {1,2,3,6,10}
    2. {4,5,7,8,9},{1,2,3,6,10} 归并为 {1,2,3,4,5,6,7,8,9,10}

具体归并步骤如下图：



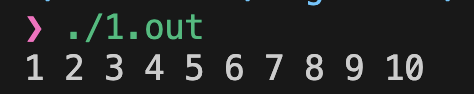
复杂度分析：

白板上的文字

描述已自动生成

递归树共有$log\_2n+1$层，每层的复杂度都是线性的，故归并排序的复杂度为O(nlog\_2n)

运行结果截屏：



1. 存储给定一个个元素有序的（升序）整型数组和一个目标值，写一个时间复杂度为的伪代码搜索中的**（需附完整Python源代码）**，如果目标值存在返回下标，否则返回。证明代码时间复杂度为。**（该题考察分治法以及时间复杂度，15分）**

**示例 1: 示例 2:**

**输入: nums = [-1,0,3,5,9,12], target = 9 输入: nums = [-1,0,3,5,9,12], target = 2**

**输出: 4 输出: -1**

**解释: 9 出现在 nums 中并且下标为 4 解释: 2 不存在 nums 中因此返回 -1**

采用二分查找法来寻找target：

伪代码：

# Pseudocode for Binary Search

# Define a constant for the maximum array size

const N = 1010

# Declare an integer array 'a' of size N

Declare an integer variable 'n'

# Function to perform binary search on an array

Function binary\_search(target)

Declare integer variables 'l', 'r'

Set 'l' to 0

Set 'r' to n - 1

While 'l' is less than 'r'

Declare an integer variable 'mid'

Set 'mid' to the average of 'l' and 'r'

If 'a[mid]' is less than 'target'

Set 'l' to 'mid' + 1

Else if 'a[mid]' is greater than 'target'

Set 'r' to 'mid' - 1

Else

Return 'mid'

# If the target is not found, return -1

Return -1

# Main program

Function main()

Declare an integer variable 'n'

Read 'n' from input

Declare an integer array 'a' of size N

For each integer 'i' from 0 to 'n - 1'

Read 'a[i]' from input

Declare an integer variable 'target'

Read 'target' from input

Declare an integer variable 'result'

Set 'result' to the result of 'binary\_search(target)'

Print "The position of the target is: " followed by 'result'

End Function

完整代码：

#include<iostream>

using namespace std;

const int N =1010;

int a[N],n;

int binary\_search(int target)

{

int l=0,r=n-1;

while(l<r)

{

int mid=(l+r)>>1;

if(a[mid]<target)

l=mid+1;

else if(a[mid]>target)

r=mid-1;

else return mid;

}

return -1;

}

int main()

{

cin>>n;

for(int i=0;i<n;i++)

cin>>a[i];

int target;

cin>>target;

cout<<"The position of the target is: "<<binary\_search(target)<<endl;

return 0;

}

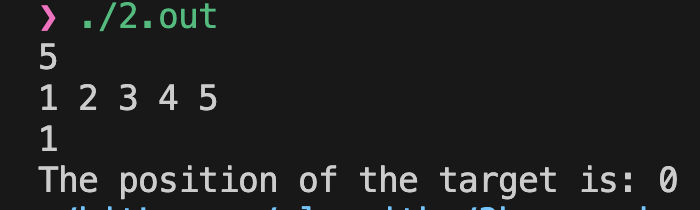
证明该算法复杂度为O(logn):

因为每次比较的时间复杂度为O(1),每次比较后会使范围减小为原范围的一半，故最多进行log\_2n次搜索，总的时间复杂度为O(log2n)。

白板上的文字

描述已自动生成

运行结果截屏：



1. 给定由个互不相同的数组成的集合以及正整数，试设计一个时间复杂度的算法，并写出伪代码，找出中最接近的中位数的个数。**（该题考察分治策略，20分）**

思路：先找到中位数，然后建立一个新数组，该数组的每个数等于原数组的数剪去中位数的绝对值，再找出新数组大小从1到k的k个数，该k个数在原数组中对应的下标即为最接近原数组中位数的k个数。

同时该题需要分n为奇数和偶数两种情况：

* + - 1. n为奇数时，求中位数即求数组中排在第（n+1）/2的元素。
      2. n为偶数时，求中位数即求数组中排在第n/2和n/2+1的两个元素的平均数。

伪代码：

# 输入 n 和 k

输入 n

输入 k

# 初始化数组 a，b

a = []

b = []

# 逐个输入 a 的元素

从 1 到 n 循环，对每个 i 执行以下操作：

输入 a[i].first

a[i].second = i

# 调用 quick\_sort 函数获取 median

median = quick\_sort(a, 1, n, (n + 1) / 2)

# 输出 median 的值和位置

输出 "median =", median.first

输出 "position =", median.second

# 逐个计算 b 的元素

从 1 到 n 循环，对每个 i 执行以下操作：

b[i].first = 绝对值(a[i].first - median.first)

b[i].second = i

# 输出最接近中位数的 k 个数

输出 "最接近中位数的", k, "个数为:"

从 1 到 k 循环，对每个 i 执行以下操作：

close = quick\_sort(b, 1, n, i)

输出 a[close.second].first

结束

完整代码：

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

const int N = 1e5 + 10;

typedef pair<double, int> PDD;

int n, m;

PDD a[N], b[N];

bool cmp(PDD a, PDD b)

{

if (a.first < b.first)

return true;

else if (a.first == b.first && a.second < b.second)

return true;

else

return false;

}

PDD quick\_find(PDD a[], int l, int r, int k)

{

if (l >= r)

return a[l];

int mid = (l + r) >> 1;

PDD x = a[mid];

int i = l - 1, j = r + 1;

while (i < j)

{

do

i++;

while (cmp(a[i], x));

do

j--;

while (cmp(x, a[j]));

if (i < j)

swap(a[i], a[j]);

}

// l~j,j+1~r;

if (j - l + 1 >= k)

{

return quick\_find(a, l, j, k);

}

else

{

return quick\_find(a, j + 1, r, k - j - 1 + l);

}

}

int main()

{

int k;

cin >> n >> k;

for (int i = 1; i <= n; i++)

{

cin >> a[i].first;

a[i].second = i;

}

PDD median;

if (n & 1)

median = quick\_find(a, 1, n, (n + 1) / 2);

else

{

PDD m1 = quick\_find(a, 1, n, n / 2);

PDD m2 = quick\_find(a, 1, n, n / 2 + 1);

median.first = (m1.first + m2.first) / 2;

median.second = (m1.second + m2.second) / 2;

}

printf("median = %lf\nposition = %d\n", median.first, median.second);

for (int i = 1; i <= n; i++)

{

b[i].first = abs(a[i].first - median.first);

b[i].second = i;

}

printf("最接近中位数的%d个数为:\n", k);

for (int i = 1; i <= k; i++)

{

PDD close = quick\_find(b, 1, n, i);

printf("%lf ", a[close.second].first);

}

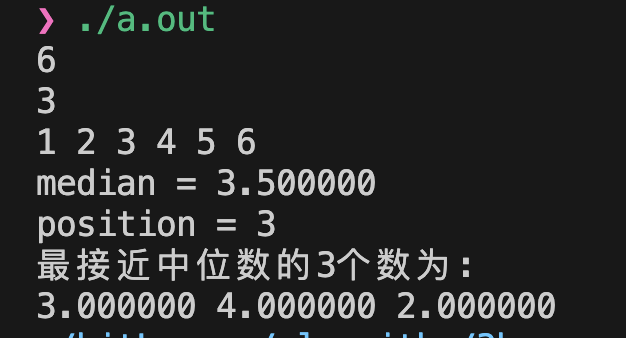
cout << endl;

return 0;

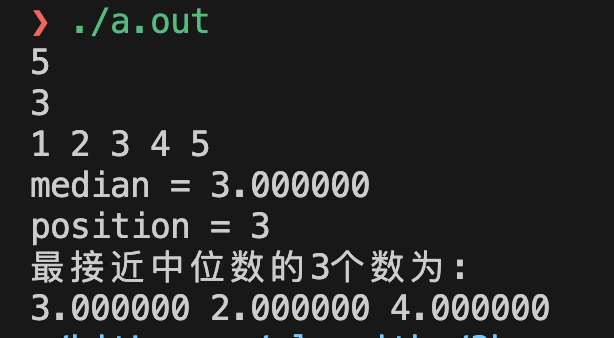
}

运行结果截屏：

偶数情况：



奇数情况：



1. 输入两个大小分别为和的正序（从小到大）数组和。设计算法，并编写时间复杂度为的伪代码**（需附完整Python源代码）**，找出并返回这两个正序数组的中位数。**（该题考察分治法以及时间复杂度，20分）**

**示例 1： 示例2：**

**输入：nums1=[1,3], nums2=[2] 输入：nums1=[1,2], nums2=[3,4]**

**输出：2.00000 输出：2.50000**

**解释：合并数组=[1,2,3]，中位数2 解释：合并数组=[1,2,3,4]，中位数 (2+3)/2=2.5**

思路：找中位数，和第三题类似，也是找数组中第k个数，不过该题有两个数组，需要特殊的比较方法。和上题一样，当(n+m)为奇数时，中位数即为第(n+m+1)/2的数，当(n+m)为偶数时，中位数即为第(n+m)/2和第(n+m)/2+1两个数的平均值。

具体步骤：

假设两个有序数组分别是A和B，要找到第k个元素我们可以先比较A[k/2-1]和B[k/2-1]。

每次比较有三种情况：

* + - 1. A[k/2-1]<B[k/2-1]，此时，比A[k/2-1]小的数最多只有A[0…k/2-2]和B[0…k/2-2]共k-2个数，因此A[k/2-1]不可能是第k大的数，同样的也可以排除A[0…k/2-2]。
      2. A[k/2-1]>B[k/2-1],和情况1类似，此时可以排除B[0…k/2-1]。
      3. A[k/2-1]==B[k/2-1],此时可以归到情况1，把A[0…k/2-1]排除，因为比A[0…k/2-1]的数最多只有k-2个。

可以看到，每次比较完A[k/2-1]和B[k/2-1]后都可以排除掉k/2个数，查找范围缩小了一半。同时，我们可以在新数组上继续寻找第k/2的数，则该数即为原两个数组的第k个数。

伪代码：  
# Pseudocode for Finding Median of Two Sorted Arrays

# Declare constants and variables

Constant N = 1010

Declare integers m, n, nums1[N], nums2[N]

# Function to get the kth element

Function getKthElement(k)

Declare integers index1 = 0, index2 = 0

# Continue until k elements have been processed

While True

# Handle boundary cases

If index1 equals m

Return nums2[index2 + k - 1]

If index2 equals n

Return nums1[index1 + k - 1]

If k equals 1

Return the minimum of nums1[index1] and nums2[index2]

# Normal case

Declare integers newIndex1 = min(index1 + k / 2 - 1, m - 1)

Declare integers newIndex2 = min(index2 + k / 2 - 1, n - 1)

Declare integers pivot1 = nums1[newIndex1]

Declare integers pivot2 = nums2[newIndex2]

If pivot1 is less than or equal to pivot2

Reduce k by newIndex1 - index1 + 1

Set index1 to newIndex1 + 1

Else

Reduce k by newIndex2 - index2 + 1

Set index2 to newIndex2 + 1

# Function to find the median of two sorted arrays

Function findMedianSortedArrays()

Declare integer totalLength = m + n

If totalLength is odd

Return getKthElement((totalLength + 1) / 2)

Else

Return (getKthElement(totalLength / 2) + getKthElement(totalLength / 2 + 1)) / 2.0

# Main program

Function main()

Read integers m and n from input

For each integer i from 0 to m - 1

Read nums1[i] from input

For each integer i from 0 to n - 1

Read nums2[i] from input

Call findMedianSortedArrays and output the result

End Function

完整代码：

#include <iostream>

#include <algorithm>

using namespace std;

const int N = 1010;

int m, n, nums1[N], nums2[N];

int getKthElement(int k)

{

int index1 = 0, index2 = 0;

while (true)

{

// 边界情况

if (index1 == m)

{

return nums2[index2 + k - 1];

}

if (index2 == n)

{

return nums1[index1 + k - 1];

}

if (k == 1)

{

return min(nums1[index1], nums2[index2]);

}

// 正常情况

int newIndex1 = min(index1 + k / 2 - 1, m - 1);

int newIndex2 = min(index2 + k / 2 - 1, n - 1);

int pivot1 = nums1[newIndex1];

int pivot2 = nums2[newIndex2];

if (pivot1 <= pivot2)

{

k -= newIndex1 - index1 + 1;

index1 = newIndex1 + 1;

}

else

{

k -= newIndex2 - index2 + 1;

index2 = newIndex2 + 1;

}

}

}

double findMedianSortedArrays()

{

int totalLength = m + n;

if (totalLength % 2 == 1)

{

return getKthElement((totalLength + 1) / 2);

}

else

{

return (getKthElement(totalLength / 2) + getKthElement(totalLength / 2 + 1)) / 2.0;

}

}

int main()

{

cin >> m >> n;

for (int i = 0; i < m; i++)

cin >> nums1[i];

for (int i = 0; i < n; i++)

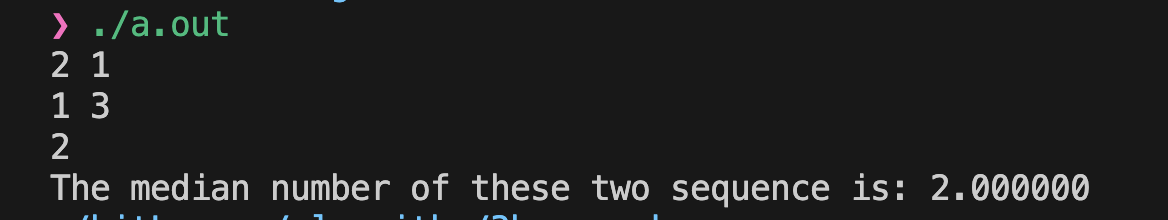
cin >> nums2[i];

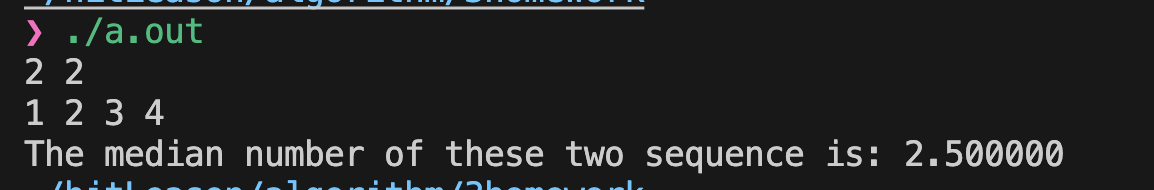
printf("The median number of these two sequence is: %lf\n",findMedianSortedArrays());

return 0;

}

代码运行结果：





1. 输入一个整数数组和一个整数，请输出其中出现频率前高的元素，设计算法并写出伪代码**（需附完整Python源代码）**。**（该题考察分治法中快速排序划分的思想，30分）**

**示例 1: 示例 2:**

**输入: nums = [1,1,1,2,2,3], k = 2 输入: nums = [1], k = 1**

**输出: [1,2] 输出: [1]**

思路：本质上还是选择第k个数的问题，统计每个数出现的频率，输出分别输出前k个频率最大的数即可。

#include<iostream>

#include<algorithm>

using namespace std;

const int N =1010;

typedef pair<int,int>PII;

PII a[N];

bool cmp(PII a, PII b)

{

if (a.first < b.first)

return true;

else if (a.first == b.first && a.second < b.second)

return true;

else

return false;

}

PII quick\_find(PII a[], int l, int r, int k)

{

if (l >= r)

return a[l];

int mid = (l + r) >> 1;

PII x = a[mid];

int i = l - 1, j = r + 1;

while (i < j)

{

do

i++;

while (cmp(a[i], x));

do

j--;

while (cmp(x, a[j]));

if (i < j)

swap(a[i], a[j]);

}

// l~j,j+1~r;

if (j - l + 1 >= k)

{

return quick\_find(a, l, j, k);

}

else

{

return quick\_find(a, j + 1, r, k - j - 1 + l);

}

}

int main()

{

int n,l=0x3f3f3f3f,r=-0x3f3f3f,k;

cin>>n>>k;

int cnt=0;

for(int i=0;i<n;i++)

{

int x;

cin>>x;

if(!a[x].first)cnt++;

a[x].first++;

a[x].second=x;

r=max(r,x);

l=min(l,x);

}

for(int i=cnt;i>=cnt-k+1;i--)

{

printf("出现频率第%d大的数为:",cnt-i+1);

cout<<quick\_find(a,l,r,i).second<<endl;

}

return 0;

}

