Skip to toolbar
Log in
Register
Search

LeetCode
Online Portal for IT Interview

Register Login
Home
About
Discuss
Members
Online Judge
LeetCode

Find the k-th Smallest Element in the Union of Two Sorted Arrays

January 27, 2011 by 1337c0d3r 122 Replies

> Given two sorted arrays A, B of size *m* and *n* respectively. Find the k-th smallest element in the union of A and B. You can assume that there are no duplicate elements.

Thanks to an anonymous reader who posted this question.

I would have to admit that this problem is pretty tricky to solve. Like most difficult problems, it requires some pretty clever observations to solve in a neat way.

The trivial way, *O*(*m+n*):
Merge both arrays and the k-th smallest element could be accessed directly. Merging would require extra space of *O*(*m*+*n*). The linear run time is pretty good, but could we improve it even further?

A better way, *O*(*k*):
There is an improvement from the above method, thanks to readers who suggested this. (See comments below by Martin for an implementation). Using two pointers, you can traverse both arrays without actually merging them, thus without the extra space. Both pointers are initialized to point to head of A and B respectively, and the pointer that has the ~~larger~~ smaller (thanks to a reader for this correction) of the two is incremented one step. The k-th smallest is obtained by traversing a total of *k* steps. This algorithm is very similar to finding intersection of two sorted arrays.

The best solution, but non-trivial, *O*(lg *m* + lg *n*):
Although the above solution is an improvement both in run time and space complexity, it only works well for small values of *k*, and thus is still in linear run time. Could we improve the run time further?

The above logarithmic complexity gives us one important hint. Binary search is a great example of achieving logarithmic complexity by halving its search space in each iteration. Therefore, to achieve the complexity of $O(\lg m + \lg n)$, we must halved the search space of A and B in each iteration.

We try to approach this tricky problem by comparing middle elements of A and B, which we identify as A_i and B_j. If A_i is between B_j and B_{j-1}, we have just found the $i+j+1$ smallest element. Why? Therefore, if we choose *i* and *j* such that $i+j = k-1$, we are able to find the k-th smallest element. This is an important invariant that we must maintain for the correctness of this algorithm.

Summarizing the above,

Maintaining the invariant
$i + j = k – 1$,

If $B_{j-1} < A_i < B_j$, then A_i must be the k-th smallest,
or else if $A_{i-1} < B_j < A_i$, then B_j must be the k-th smallest.

If one of the above conditions are satisfied, we are done. If not, we will use *i* and *j* as the pivot index to subdivide the arrays. But how? Which portion should we discard? How about A_i and B_j itself?

We make an observation that when $A_i < B_j$, then it must be true that $A_i < B_{j-1}$. On the other hand, if $B_j < A_i$, then $B_j < A_{i-1}$. Why?

Using the above relationship, it becomes clear that when $A_i < B_j$, A_i and its lower portion could never be the k-th smallest element. So do B_j and its upper portion. Therefore, we could conveniently discard A_i with its lower portion and B_j with its upper portion.

If you are still not convince why the above argument is true, try drawing blocks representing elements in A and B. Try visualize inserting blocks of A up to A_i in front of B_{j-1}. You could easily see that no elements in the inserted blocks would ever be the k-th smallest. For the latter, you might want to keep the invariant $i + j = k – 1$ in mind to reason why B_j and its upper portion could never be the k-th smallest.

On the other hand, the case for $A_i > B_j$ is just the other way around. Easy.

Below is the code and I have inserted lots of assertion (highly recommended programming style by the way) to help you understand the code. Note that the below code is an example of tail recursion, so you could technically convert it to an iterative method in a straightforward manner. However, I would leave it as it is, since this is how I derive the solution and it seemed more natural to be expressed in a recursive manner.

Another side note is regarding the choices of *i* and *j*. The below code would subdivide both arrays using its array sizes as weights. The reason is it might be able to guess the k-th element quicker (as long as the A and B is not differed in an extreme way; ie, all elements in A are smaller than B). If you are wondering, yes, you could choose *i* to be A's middle. In theory, you could choose any values for *i* and *j* as long as the invariant *i*+*j* = *k*-1 is satisfied.

```
int findKthSmallest(int A[], int m, int B[], int n, int k) {
  assert(m >= 0); assert(n >= 0); assert(k > 0); assert(k <= m+n);

  int i = (int)((double)m / (m+n) * (k-1));
  int j = (k-1) - i;

  assert(i >= 0); assert(j >= 0); assert(i <= m); assert(j <= n);
  // invariant: i + j = k-1
  // Note: A[-1] = -INF and A[m] = +INF to maintain invariant
  int Ai_1 = ((i == 0) ? INT_MIN : A[i-1]);
  int Bj_1 = ((j == 0) ? INT_MIN : B[j-1]);
  int Ai   = ((i == m) ? INT_MAX : A[i]);
  int Bj   = ((j == n) ? INT_MAX : B[j]);

  if (Bj_1 < Ai && Ai < Bj)
    return Ai;
  else if (Ai_1 < Bj && Bj < Ai)
    return Bj;

  assert((Ai > Bj && Ai_1 > Bj) ||
         (Ai < Bj && Ai < Bj_1));

  // if none of the cases above, then it is either:
  if (Ai < Bj)
    // exclude Ai and below portion
    // exclude Bj and above portion
    return findKthSmallest(A+i+1, m-i-1, B, j, k-i-1);
  else /* Bj < Ai */
    // exclude Ai and above portion
    // exclude Bj and below portion
    return findKthSmallest(A, i, B+j+1, n-j-1, k-j-1);
}
```

★★★★★

Rating: 4.7/**5** (158 votes cast)

Find the k-th Smallest Element in the Union of Two Sorted Arrays, 4.7 out of 5 based on 158 ratings

← Sliding Window Maximum   Coins in a Line →

122 thoughts on "Find the k-th Smallest Element in the Union of Two Sorted Arrays"

Martin

January 27, 2011 at 8:19 am

A much simpler algorithm that is O(k) is this:

```
int findKthSMallest(int[] A, int[] B, int k) {
int a_offset = 0, b_offset = 0;
if (A.length + B.length < k) return -1;

while (true) {
if (a_offset < A.length) {
while (b_offset == B.length ||
A[a_offset] <= B[b_offset]) {
a_offset++;
if (a_offset + b_offset == k) return A[a_offset];
}
}
if (b_offset < B.length) {
while (a_offset == A.length ||
A[a_offset] >= B[b_offset]) {
b_offset++;
}
if (a_offset + b_offset == k) return B[b_offset];
}
}
}
```

(btw, would be nice to have <pre/> support in comments).

Reply ↓

+2

freedom77

December 30, 2012 at 8:05 pm

Won't work for this case
{1,2,3} {4,5}, 3

Reply ↓

+1

QIlu

October 5, 2014 at 2:49 pm

your solution is very inspiring but there are minor bugs in your code:

1, when return the kth smallest element, we need to return A[a_offset-1] and B[b_offset-1] instead of A[a_offset] or B[b_offset]. Since when a_offset is 3 then A[3] is actually the 4th element in array A.

2, In the two sub loops, after a_offset++/b_offset++, if a_offset /b_offset are equal to A.length/B.length, we need to break this while loop. Otherwise, when this while condition is checked again, A[a_offset]/B[b_offser] will be out of range.

I fixed these two minor bugs and post my code (in c#) as follows:

```
public int FindSmallestElement(List listA, List listB, int k)
    {
        int A_offset = 0;
        int B_offset = 0;
        if (listA.Count + listB.Count < k)
            return -1;
        while (true) {
            if (A_offset < listA.Count) {
                while (B_offset == listB.Count|| listA [A_offset] <= listB [B_offset]) {
                    A_offset++;
                    if (A_offset + B_offset == k)
                        return listA [A_offset-1];
                    if (A_offset == listA.Count)
                        break;
                }
            }
            if (B_offset < listB.Count) {
                while (A_offset == listA.Count|| listB [B_offset] <= listA [A_offset]) {
                    B_offset++;
                    if (A_offset + B_offset == k)
                        return listB [B_offset-1];
                    if (B_offset == listB.Count)
                        break;
                }
            }
        }
    }
```

Reply ↓

+1 👍 👎

sammy

November 28, 2014 at 4:21 pm

This is not going to work for the cases when k == 0 and k ==1 (or k == 1 and k == 2, depending how do you count the first element)

Reply ↓

0 👍 👎

simon

January 27, 2011 at 10:08 am

Hi 1337coder,

You are a genius.

1.

Two reverse sorted arrays A and B have been given.
such that size of A is m and size of B is n
You need to find the k th largest sum (a+b) where a is taken from A and b is taken from B. such that k < m*n
It can be done in O(n*n) but i believe it can be done in a very efficient way. I heard someone say there is a paper from SODA. It's possible to solve it in O(N), however, it requires very complex algorithm. I am wondering if you can come up with an NlogN algorithm first then try O(N). These two problems have been very hot interview problems from Google, however, not many people can solve it very efficiently yet. I believe you can do it after I read many posts on your site.

2.
Given a N*N Matrix.
All rows are sorted, and all columns are sorted.
Find the Kth Largest element of the matrix.

Reply ↓

0

raman
April 6, 2011 at 6:06 am

```
#include
using std::cin;;
using std::cout;

int main()
{

int A[]={10,8,5,3,2};

int B[]={9,7,6,4,1};

bool atraverse = true;

int sumToBeatAIndex=0;

int sumToBeatBIndex=1;

int j=1;

int k=0;

int i;

int sumtobeat;

int n=5;
int tempBIndex;

int tempAIndex;

cout<<"("<<B[0]<<","<<A[0]<<") ";

sumtobeat=A[0]+B[1];

for(i=1;isumtobeat)

cout<<"("<<B[k]<<","<<A[j]<<") ";

else

{

sumtobeat=B[k]+A[j];

tempBIndex=k;

tempAIndex=j;

j=sumToBeatAIndex;

k=sumToBeatBIndex;

sumToBeatAIndex=tempAIndex;

sumToBeatBIndex=tempBIndex;

cout<<"("<<B[k]<<","<<A[j]<<") ";

atraverse=!(atraverse);

}

if(atraverse)

j++;

else

k++;

}
```

```
cout<<"\n";

return 0;
}
```

Reply ↓

+1

Raynor
January 27, 2011 at 2:06 pm

I just want to point out that the trivial way does not require that much of space. You don't need to merge two arrays. Just traverse through them is enough.

Reply ↓

-1

1337c0d3r
January 27, 2011 at 3:04 pm

@simon:
1) Using a heap-based solution you could get O(k lg min(m, n)) run time complexity. To figure out the O(n) solution seemed pretty difficult. It seemed like someone already solved it here: http://www.ocf.berkeley.edu/~wwu/cgi-bin/yabb/YaBB.cgi?board=riddles_cs;action=display;num=1132204952;start=25

I haven't read his post, so I couldn't provide the validity of his solution yet.

2) I remembered seeing this as an exercise problem in CLRS. Will look into it when I have time.

Reply ↓

0

1337c0d3r
January 27, 2011 at 3:04 pm

@Martin and Raynor:
Yup both of you are right. I have updated my post, thanks!

Reply ↓

0

Anonymous
January 28, 2011 at 6:27 am

@1337 — u are seriously a bilody genius or someone who has spent hours at algos. at work or at school and can tie all the algo concepts really well (Which is really encourage-able)

Now-a-days at most software jobs where you just code (mostly read from a trivial DB and do some business logic and display results) you lose this touch.

Thanks for all the great posts.

Reply ↓

0

1337c0d3r

January 28, 2011 at 11:20 am

@Anonymous:
Definitely the latter. I love algos and though I admit it didn't get apply at work very often, it does come into play once in awhile.

Have you read the book "Programming pearls"? It gives some really good real world examples of how efficient algos can help even in some simple business logic.

Reply ↓

0

Anonymous

January 29, 2011 at 4:52 pm

The following algorithm can achieve O(logK) complexity. The idea is to first select k items from both arrays using their proportional length, then adjust the number of items from each array using binary search. The follwoing C# code illustrates the idea:

```
private static int FindKthInUnion(int[] a, int[] b, int k)
{
if (k == 0) return Min(a[0], b[0]);
int la = Min(a.Length – 1, k);
int lb = Min(b.Length – 1, k);

int i = (k * a.Length) / (a.Length + b.Length);
int j = k – i – 1;

for (; ; )
{
if (a[i] < b[j])
{
if (i == la || a[i + 1] > b[j])
{
return b[j];
}
else
{
i += (la – i + 1) / 2;
j = k – i – 1;
}
}
else
{
if (j == lb || b[j + 1] > a[i])
{
return a[i];
}
else
{
j += (lb – j + 1) / 2;
i = k – i – 1;
}
}
}
}
```

Reply ↓

+1

Anonymous
February 7, 2011 at 12:44 am

Hi 1337coder,

Can you please tel me why "We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1 (you mentioned Bj-1 <Ai < Bj, right?). On the other hand, if Bj < Ai, then Bj < Ai-1." ?

Thank you very much

Reply ↓

+3

1337c0d3r
February 7, 2011 at 2:28 am

@Anonymous:
If Bj-1 < Ai < Bj, then we are already done, no need to continue further.

"We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1" –> This observation is due to the above condition Bj-1 < Ai < Bj NOT being satisfied.

Reply ↓

+1

Anonymous
February 9, 2011 at 11:09 am

I found this code sample to be easier to understand:
http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15451-s01/recitations/rec03/rec03.ps

Reply ↓

-1 👍 👎

anonymous
April 6, 2011 at 6:08 pm

I can see some limitation in this code. It won’t work if first array is of size 100, second is of size 10, and k = 50. It probably works only when k <= min (first array size, second array size)

Reply ↓

0 👍 👎

Ashok Koyi
October 31, 2011 at 7:20 am

I think we can augment the code to skip the steps of comparison, if we knew that the index we have calculated is out of bounds in the smaller array

Reply ↓ Report user

0 👍 👎

online.service
February 14, 2011 at 1:27 pm

Hi 1337code, "Using a heap-based solution you could get O(k lg min(m, n)) run time complexity. " for simon's question 1. I feel we need a O(k lg max(m, n)), since after pop a max in the heap, we need to fill two neighbour items to the heap, we can get a max(m,n) heap for some m*n matrix like this one

3*n matrix

3*(n+n 2*n n
3*(2n-1) 2*(n-1) n-1
.
.
.
3*(n) 2 1

so the largest will be in the first column, after we pop the max in the heap , we need to push the next one in the first column and the one in the same row but in second column. So we get to the end of the first column we have a O(n) space heap

Reply ↓ 0

ripper234
February 14, 2011 at 11:40 pm

Regarding simon's problems. I went over the discussion at the Berkly forum and didn't find any solution that I was convinced both works and is O(n). I posted this to Stack Overflow, hoping the discussion there will turn out to be more fruitful:

http://stackoverflow.com/questions/5000512/find-the-top-k-values-in-a-sorted-n-x-n-matrix

Reply ↓ 0

orchid
April 25, 2011 at 8:05 am

```
int findKthSmallestInoneset(int s1[], int s2[], int m, int n, int k){
int mid, low, high, pos;

if(k > (m + n))
cout << "There does not exist the k-th element" < s2[n – 1])
return s1[m – 1];
else
return s2[n – 1];
}

if(k == 1){
if(s1[0] > s2[0])
return s2[0];
else
return s1[0];
}

if(k > m)
high = m – 1;
else
high = k – 1;
low = 0;

while(low (n – 1)){
low = mid + 1;
continue;
```

```
}

if(pos == (n – 1)){
if(s1[mid] >= s2[pos])
return s1[mid];
low = mid + 1;
}
else{
if(s1[mid] >= s2[pos] && s1[mid] = s2[pos + 1])
high = mid – 1;

if (s1[mid] < s2[pos])
low = mid + 1;
}
}//while
return 0;
}
```

Reply ↓ 0

orchid

April 25, 2011 at 8:06 am

a non-recrusive method

Reply ↓ 0

tanliboy

May 1, 2011 at 8:53 am

There is an O(lg(min{m, n, k}) method. 🙂 FYI.
http://tanliboy.wordpress.com/2011/05/01/some-interesting-google-interview-problems/

Reply ↓ 0

David

May 16, 2011 at 4:22 am

I was looking for a solution to a very similar problem, the k-th largest element of 2 vectors sorted in nondecreasing order. I tried to adapt the solution to use with vectors, but it just did't work:

```
int findKthSmallest(int A[], int m, int B[], int n, int k) {
    assert(m >= 0); assert(n >= 0); assert(k > 0); assert(k <= m+n);
    int i = (int)((double)m / (m+n) * (k-1));
    int j = (k-1) - i;

    assert(i >= 0); assert(j >= 0); assert(i <= m); assert(j <= n);
    // invariant: i + j = k-1
    // Note: A[-1] = -INF and A[m] = +INF to maintain invariant
    int Ai_1 = ((i == 0) ? INT_MIN : A[i-1]);
    int Bj_1 = ((j == 0) ? INT_MIN : B[j-1]);
    int Ai   = ((i == m) ? INT_MAX : A[i]);
    int Bj   = ((j == n) ? INT_MAX : B[j]);

    if (Bj_1 < Ai && Ai < Bj)
        return Ai;
    else if (Ai_1 < Bj && Bj < Ai)
        return Bj;

    assert((Ai > Bj && Ai_1 > Bj) || (Ai < Bj && Ai < Bj_1));

    // if none of the cases above, then it is either:
    if (Ai < Bj)
        // exclude Ai and below portion
        // exclude Bj and above portion
        return findKthSmallest(A+i+1, m-i-1, B, j, k-i-1);
    else // Bj < Ai
        // exclude Ai and above portion
        // exclude Bj and below portion
        return findKthSmallest(A, i, B+j+1, n-j-1, k-j-1);
}

int select(int k, const vector& v1, const vector& v2) {
    int A[v1.size()], B[v2.size()];
    for (int i = 0; i < v1.size(); ++i) A[i] = v1[i];
    for (int i = 0; i < v2.size(); ++i) B[i] = v2[i];
    return findKthSmallest(A, v1.size(), B, v2.size(), k);
}
```

I just call the “select” function with the 2 vectors and fit them in 2 arrays. I think the algorithm is incorrect.

I did the same with the solution above, from talinboy:

```
int findKthSmallest2(const vector& A, int m, const vector& B, int n, int k) {
    if(m > n){
        return findKthSmallest2(B, n, A, m, k);
    }

    m = min(k, m), n = min(k, n);
    int s = 0, f = m;
    while (s < f) {
        int mid = s + (f - s) / 2,  bIdx = k - 1 - mid;
        if (bIdx >= n || A[mid] < B[bIdx]) {
            s = mid + 1;
        } else {
            f = mid;
        }
    }
    return max(s ? A[s - 1] : INT_MIN, k - 1 - s >= 0 ? B[k - 1 - s] : INT_MIN);
}
```

```
int select(int k, const vector& v1, const vector& v2) {
    return findKthSmallest2(v1, v1.size(), v2, v2.size(), k);
}
```

It works perfectly and with O(lg(min{m, n, k}) !!

I don’t know where is the error in your algorithm. It might be i’m also using vectors with duplicate elements.

I think the algorithm from tanliboy is the quickest and most general implementation i’ve seen in internet that works fine.

P.S.: two examples that doesn’t work with 1337 code:

```
v1 = {0}
v2 = {0, 1}
k = 2
```

```
v1 = {0, 1, 2}
v2 = {0, 1, 2, 3}
k = 5
```

Reply ↓

0

1337c0d3r Post author
May 16, 2011 at 10:26 am

David, please read my post carefully:
“You can assume that there are no duplicate elements.”
My solution assumes no duplicate elements. If you really understand how my solution works, you should be able to adapt it to work with duplicate elements. tanliboy’s solution is another possible efficient solution, which uses the idea of binary search.

Reply ↓

0

Anantha Krishnan
June 16, 2011 at 10:43 am

#include

int findKthsmallest(int a[],int m,int b[],int n,int k)
{
int i=0,j=0,ti=0,tj=0,I=0,J=0,M=m,N=n;
while(1)
{
ti = (int)((double)m/(m+n) * (k-1));

```
tj = (k-1)-ti;
i = I+ti;
j= J+tj;
//printf(" i=%d j=%d\n",i,j);
if(j>0 && j<N && ib[j-1] && a[i]0 && i<M && ja[i-1] && b[j]<a[i])
return b[j];
if(j==0 && i<M && a[i]<b[j])
return a[i];
if(i==0 && j<N && b[j]b[j-1])
return a[i];
if(i==M && b[j]>a[i-1])
return b[j];
if(i<M && j<N)
{
if(a[i]=M)
{
k=k-tj-1;
n=n-tj-1;
J=j+1;
}
else
{
k=k-ti-1;
m=m-ti-1;
I=i+1;
}
}
}

int main()
{
int a[]={1,2,3};
int b[]={4};
int m=3,n=1,k=3;
printf("%d",findKthsmallest(a,m,b,n,k));
return 0;
}
```

Reply ↓

0

daxingqiao

June 19, 2011 at 2:36 am

Hi 1337, there is one bug for your solution. How is it going if k = m+n? I think we need special handling for this edge case.

Reply ↓

0

daxingqiao

June 19, 2011 at 3:01 am

There is another bug when m == 1 or n == 1, in this case, we need special handling to adjust i or j, here is the code snippet I have modified:

```
int findKthElement(int A[], int m, int B[], int n, int k)
{
if ( m == 0 && n == 0 || k == 0 || k > m + n)
{
return -1;
}
else if( m == 0)
{
return B[k – 1];
}
else if (n == 0)
{
return A[k – 1];
}
else if(k == m + n)
{
return std::max(A[m – 1], B[n -1]);
}
else if( k == 1)
{
return std::min(A[0], B[0]);
}

int i = (int)(double(m)/(m+n)*(k – 1) + 0.5);
int j = k-1-i;
if (i == m)
{
i–;
j++;
}
else if (j == n)
{
i++;
j–;
}

// A[i-1], B[j-1]
int A_i_1 = ( i == 0)? INT_MIN : A[i – 1];
int B_j_1 = ( j == 0)? INT_MIN : B[j – 1];

if (A[i] >= B_j_1 && A[i] <= B[j])
{
return A[i];
}
else if (A[i] = A_i_1 && B[j] <= A[i])
{
return B[j];
}
else if (B[j] <= A[i])
{
return findKthElement(A, i, B + j + 1, n – j – 1, k – j – 1);
}
return -1;
}

double findMedianSortedArrays(int A[], int m, int B[], int n) {

if (m + n & 1)
{
return findKthElement(A, m, B, n, (m + n + 1)/2);
}
else
{
return (findKthElement(A, m, B, n, (m + n)/2) + findKthElement(A, m, B, n, (m + n)/2 + 1))*1.0/2;
}
return -1;
}
```

Reply ↓

0

siva

June 26, 2011 at 12:54 am

Can we not do in log k?

have 2 pointers.. i and j for array 1 and array 2..

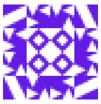
```
while (i+j != k) {
```

```
if (array1[i] > array2[j]) {
i = i + [(k-(i+j))/2]
} else {
j = j + [(k-(i+j))/2]
}
}

return min of (array[i] , array [j])
```

validate this one..

Reply ↓

0



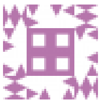
Nitish

June 29, 2011 at 8:08 am

Siva, initially n=100000, m=10;
suppose k=1000;
i=0;j=0;
arrray1[]={ 1,2,3,4….}
array2[]={100901,1212}'
Now, as array1[i=0] j is set to 10000. Seg fault

Reply ↓

0



siva

July 2, 2011 at 10:51 pm

you are correct.. I missed to have bound check on both i and j.. if we do that.. we can do this solve this in log k rite?

Reply ↓

0

Anurag Atri

July 4, 2011 at 8:52 am

Again , fantastic work , a tried to get a solution to this problem on a lot of places on the net but all of them were either wrong or lacked explanation , your code works fine and the explanation is great !
just one thing , you can include these tests :
m = min (k , m) ;
n = min (k , n) ;
a slight improvement .
and if you use
if (ai >= bj_1 && ai <= bj) //greater than equal to , in place of greater than

the code will work for duplicate values as well (provided values of i and j are checked)

Thank You sooo much 😃 😃

Reply ↓

0 👍 👎

new

July 19, 2011 at 5:44 pm

was wondering why you initialized i this way, why not i from 0?
”

int i = (int)((double)m / (m+n) * (k-1));
int j = (k-1) – i;”

Reply ↓

0 👍 👎

maxq

July 20, 2011 at 8:08 pm

I just wrote another version with the same log complexity,
and verified with a few inputs via your coding panel:)

Any comments?

```
// Type your C++ code and click the “Run Code” button!
// Your code output will be shown on the left.
// Click on the “Show input” button to start entering input data.
#include
using namespace std;

int find_k_small(int a[], int m, int b[], int n, int k, int &kmin)
{
if (k m+n)
return -1;

int i = (m * (k-1))/(m+n);
int j = k – 2 – i;

while (1) {
if ((j == n-1 || a[i] < b[j+1]) &&
(i == m-1 || b[j] b[j]) ? a[i]: b[j];
return 1;
} else if (j = b[j+1]) {
i /= 2;
j = k -2 -i;
} else {
j /= 2;
i = k -2 -j;
}
}
return 0;
}

int main() {
// Start typing your code here…
int kmin = 0;
int a[] = {1,3, 9, 11, 13, 15, 23};
int b[] = {2, 4, 6, 16, 18, 20};
int ret = find_k_small(a, sizeof(a)/sizeof(int),
b, sizeof(b)/sizeof(int),
9, kmin);
cout <<"ret="<<ret<<",kmin="<<kmin<<endl;
return 0;
}
```

Reply ↓

0

Pingback: Find the k-th Smallest Element in the Union of Two Sorted Arrays | ㅁㅏㅅㅅㅣ의 Blog

Coder

July 30, 2011 at 9:11 am

Are both arrays sorted in increasing order or decreasing order….
if they are sorted in increasing order ,How is this claim true??

We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj < Ai-1.

i may sound stupid but i dont get this claim

Reply ↓

0

lipingwu

August 7, 2011 at 9:10 am

The i and j are not the middle of the array, then how can you prove that the algorithm will be O(lgm + lgn) or less? Thanks.

Reply ↓ 0

siv

September 28, 2011 at 3:23 am

Hi,
Can some one explain me why 'i' value is calculated as below

int i = (int)((double)m / (m+n) * (k-1));

Thanks,
siv

Reply ↓ 0

ibrahim

February 1, 2013 at 3:50 pm

that's for to stay inside of the array. if m < (k-1) /2 i will be out of array bounds

Reply ↓ 0

Pingback: BLOG.MASTER_SLAVE — Find the k-th Smallest Element in the Union of Two Sorted Arrays

japanbest

October 13, 2011 at 2:22 am

why set the ratio i/j= m/n , I prefer i=j, in other words, i=j= (k-1)/2
Since the next recursion k-> k-i or k->k-j if i=j then we can optimize for the worst case. And yes I know both ratios work the same in average cases.

Does i/j=m/n has a special objective?

Reply ↓ 0

ibrahim

February 1, 2013 at 3:49 pm

that's for to stay inside of the array. if m < (k-1) /2 i will be out of array bounds

Reply ↓ 0

Jeffrey

December 30, 2011 at 6:32 pm

The post says

```
We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other ha
```

I don’t agree. If A is [1,2,3], B is [4,5,6], then this statement is not true.

Reply ↓

0

idank

January 20, 2012 at 7:50 am

I solved this using a similar approach in O(lgn), see here for an explanation.

Reply ↓

0

reader

March 4, 2012 at 9:07 am

Hi 1337c0d3r.

I am having trouble understanding your code.

1) You mention that the k is found if Bj-1 < Ai < Bj **AND** the invariant i+j+1==k holds true.

But I don’t see how you check for the invariant in your code.

You just return if for the current i,j the condition Bj-1 < Ai < Bj (or its reverse for B) holds true.How do you check for the invariant?

2) Also your code does not do what you describe in the code.If Ai<Bj you do a recursive call and you use j as the size of the array.So you just don't neglect a portion as you say.

3)For A={1,2,6} and B={3,4,5} and k = 2, the code ends up with division by 0 when calculating i since in the first entry of the method i=0 and j=1 and so you dont' find k and you hit the Ai<Bj and you call the method recursevily with k-i-1 which is 2-0-1=1 so in the calculation of i you divide by zero.

This is a very interesting problem and perhaps I didn't get your solution, so if you please took the time to read my comments and help me understand this I would be greatful. Thank you

Reply ↓

+1

Maja Grubic

September 9, 2013 at 9:20 am

1) Invariant is checked when initializing j.

j = (k-1) -i

so it is obvious that i+j = k-1

2) Make a trace of the code. Size of the array B with its upper part discarded is exactly j

3) No, no it doesn't

Reply ↓ Report user

0

upi

March 26, 2012 at 12:39 am

Narrow bounding method.
Assume we have two arrays A (A.size() == n) and B (B.size() == m) and k – target value.
In case of n<=0 or m= 1.
Get median element of A (A[n/2]) and find position in B, where we can insert such element, let position be i, such that:
– i==0 and B[i] > A[n/2];
– i==m and B[i-1] < A[n/2];
– 0 < i < m and B[i-1] < A[n/2] **k, then we start procedure with A[0..n/2-1], B[0..i-1]**
3. i+n/2+1 < k, then we start procedure with A[n/2+1..n-1], B[i+1..m-1], k -= i+n/2+1

try to code:

```
int upperbound (int *arr, int n, int t)
{
    int   l=0, r=n;
    while (l> 1;
        if (arr[m] <= t) l = m + 1;
        else r = m;
    }
    return l;
}

/// Assume k start from 1
int findKth (int *A, int n, int *B, int m, int k)
{
    /// trivial border case (don't forget k started from 1)
    if (n <= 0) return B[k-1];
    if (m  A[n/2]), m (in case of all B[j] <A> k)
    {
        return findKth (A, n/2, B, i, k);
    }
    return findKth (&A[n/2+1], n - n/2 -1, &B[i], m - i, k - (i + n/2 + 1));
}
```

Is it correct?

Reply ↓

0

upi

March 26, 2012 at 2:35 am

Strange code formatting:

```
/// Assume k start from 1
int findKth (int *A, int n, int *B, int m, int k)
{
/// trivial border case (don’t forget k started from 1)
if (n <= 0) return B[k-1];
if (m k)
{
return findKth (A, n/2, B, i, k);
}
return findKth (&A[n/2+1], n – n/2 -1, &B[i], m – i, k – (i + n/2 + 1));
}
```

Reply ↓

0

veeru

March 31, 2012 at 7:14 pm

```
public int KSmallest(int[] s1, int[] s2, int k)
{
int i = 0;
int j = 0;

if (k >= s1.Length + s2.Length)

throw new ArgumentOutOfRangeException(“k”);

while(i s2.Length)
{
break;
}

if (i + j + 1 == k)
break;

if (s1[i] < s2[j])
i++;
else
{
```

```
j++;
}
}

if(i == s1.Length)
{
while (i + j + 1 != k)
j++;

return s2[j];
}

if(j == s2.Length)
{
while (i + j + 1 != k)
i++;

return s1[i];
}

if(i + j + 1 == k)
{
return s1[i] <= s2[j]?s1[i]:s2[j];
}

return int.MaxValue;
}
```

Reply ↓ 0 👍 👎

jastination
April 23, 2012 at 6:41 pm

Hi Genius!

What if arrays are unsorted? Do you have a good solution for that?

Thanks!

Reply ↓ Report user 0 👍 👎

vk
May 12, 2012 at 2:41 am

i found martin's explanation slightly off …what if there are duplicates
thus i implemented the same concept(using 2 ptrs)which actually worked for me and it handles duplication as well…
cheers

```
int kthsmallest(int *A,int *B,int k)
{
int i=0,j=0;
k=k-1;
while(1)
{
if(A[i]B[j])
{ if(i+j==k)return B[j];
j++;
}
else //(A[i]==B[j])
{if(i+j==k)return A[i];
i++;j++;k++;
}
}
}
```

Reply ↓ 0

flynewdream
May 16, 2012 at 5:29 pm

Why "We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj < Ai-1" ?

Ai can be between Bj and Bj-1, right?

Reply ↓ Report user 0

Hai
July 3, 2012 at 10:50 am

then you found the solution.

Reply ↓ +2

Fred
May 22, 2012 at 12:41 pm

actually, we can initially bound m and n to be smaller than k.

```
m=min(m,k);
n= min(n,k);
```

then the time complexity is O(log(k))

Reply ↓ Report user 0

Hai
July 3, 2012 at 10:52 am

Seems original code can not handle the case if A and B doesn’t overlap, e.g:

A= {2 , 4} B={ 7, 9, 11, 13}, will has assertion fail.

Reply ↓ 0

golden
August 14, 2012 at 11:03 pm

```
int first=0,second=0,k,min;
scanf(“%d”,&k);
while((k–) > 0)
{
if (b[second]<a[first])
{ min=b[second];second++;}
else{
min=a[first];
first++;}}
```

this is working code

Reply ↓ Report user 0

Prateek Caire
August 19, 2012 at 2:00 am

Thanks 🙂

Median of 2 sorted array is special case of this one where k = (m+n)/2

Reply ↓ 0

rajat

September 4, 2012 at 8:26 pm

by comparing middle elements of A and B, which we identify as Ai and Bj. If Ai is between Bj and Bj-1, we have just found the i+j+1 smallest element.

I think instead of Bj-1 it should be Bj+1

Reply ↓ Report user

0

Pingback: 二分查找法的实现和应用(进阶篇) | 编程·早晨

imposter

September 27, 2012 at 2:41 am

can any one please tell me why pivot i is choosen as m/(m+n)*(k-1) … i didnt get the logic for that…

Reply ↓

+2

sankalp

October 6, 2012 at 9:02 am

hey I don’t understand why the element is i+j+1 th smallest.

Ai has i-1 elements ahead of it in the original array and j-1 elements in the second array . Thus , it is i+j-1 th element I think.

Reply ↓

0

YangOu

February 21, 2013 at 10:19 pm

I actually have the same question. Have you figured out?

Reply ↓ Report user

0

phantom11

March 31, 2013 at 11:17 am

Here indexing is done 0 based. So Ai has i elements before it and Bj_1 has j elements before it (+1 including it ,so j+1 elemtns less than equal to it). So it is i+j+1

Reply ↓

+1

Pingback: Must read problems of LeetCode | What I learn, I blog !

Pingback: 二分查找法的实现和应用(进阶篇) « 成人免费资源三级分享网站

Pingback: Ider – 二分查找法的实现和应用(进阶篇)

Pingback: read.guoruEi » Blog Archive » 二分查找法的实现和应用(进阶篇)

Juanissa

March 1, 2013 at 11:17 am

@1337c0d3r, what if the arrays are constant arrays of the same value? I think

```
if (Bj_1 < Ai && Ai < Bj)
return Ai;
else if (Ai_1 < Bj && Bj < Ai)
return Bj;
```

```
can be
```

```
if (Bj_1 <= Ai && Ai <= Bj)
return Ai;
else if (Ai_1 <= Bj && Bj <= Ai)
return Bj;
```

Reply ↓ Report user

0

Anonymous

March 4, 2013 at 8:12 am

You really should provide the reference where you get these solutions. This article is almost identical to the solution provided in Algorithms for Interviews book, exercise 1.18. It lays the three solutions in the same sequence and with similar explanations.

For the sake of fairness, don’t pretend these are your bright ideas.

However I congratulate you on the quality of the narrative and code examples.

Reply ↓

0

Terry Li

April 11, 2013 at 9:45 am

log(m+n) solution without recursive:

```
    public int findKthLargest(int A[], int B[], int k) {
        int n = A.length;
        int m = B.length;

        if (k  m + n)
            return -1;
        int al = 0, ar = n - 1;
        int bl = 0, br = m - 1;

        while (true) {
            n = ar - al + 1;
            m = br - bl + 1;
            int i = Math.max(0, (int) ((1.0 * n * (k - 1)) / (m + n)));
            int j = k - 1 - i;
            i += al;
            j += bl;
            if (al > ar)
                return B[j];

            if (bl > br)
                return A[i];

            int ai = (i > ar) ? Integer.MAX_VALUE : A[i];
            int ai_1 = (i  br) ? Integer.MAX_VALUE : B[j];
            int bj_1 = (j = bj_1 && ai = ai_1 && bj <= ai) {
                return bj;
            }

            if (ai < bj) {
                k -= i - al + 1;
                al = i + 1;
                br = j;
            } else {
                k -= j - bl + 1;
                ar = i;
                bl = j + 1;
            }
        }
    }
```

Reply ↓ Report user

0

Terry Li

April 11, 2013 at 9:47 am

here it is:

```
public int findKthLargest(int A[], int B[], int k) {
int n = A.length;
int m = B.length;

if (k m + n)
return -1;
int al = 0, ar = n – 1;
int bl = 0, br = m – 1;

while (true) {
n = ar – al + 1;
m = br – bl + 1;
int i = Math.max(0, (int) ((1.0 * n * (k – 1)) / (m + n)));
int j = k – 1 – i;
i += al;
j += bl;
if (al > ar)
return B[j];

if (bl > br)
return A[i];

int ai = (i > ar) ? Integer.MAX_VALUE : A[i];
int ai_1 = (i br) ? Integer.MAX_VALUE : B[j];
int bj_1 = (j = bj_1 && ai = ai_1 && bj <= ai) {
return bj;
}

if (ai < bj) {
k -= i – al + 1;
al = i + 1;
br = j;
} else {
k -= j – bl + 1;
ar = i;
bl = j + 1;
}
}
}
```

Reply ↓ Report user

0

frank

May 14, 2013 at 11:08 am

The author gave a very good point in providing the relationship between i,j and k: i+j = k-1. And I believe the best algorithm should have the time complexity of O(log(min(m,n,k)). Simple proof:
Suppose m>=n,
case 1: k=m, since m+n>=k, we can flip to find the m+n-k'th largest element. and m+n-k<=n. So same as case 1.
case 3: n<k<m,
case 3a: if n = O(k), no problem.
case 3b: if n = o(k), we can use the bisection method for the small array. And it's not hard to find the k's smallest element in O(n).

Reply ↓ Report user

0 👍 👎

frank
May 15, 2013 at 12:17 pm

This is the C# code with time complexity O(min(m, n, k)). All possible cases were covered.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SortApp
{
    class Program
    {
        static int sizeA;
        static int sizeB;
        static int kth;
        static List listA;
        static List listB;

        static void Main(string[] args)
        {
            Console.WriteLine("init...");
            Init();
            //Init1();
            Console.WriteLine("First Method...");
            Solution();
            Console.WriteLine("Second Method...");
            Solution2();
        }

        static void Init()
        {
            listA = new List();
            listB = new List();
            Random r = new Random();

            sizeA = r.Next(100000, 100000000);
            int div = r.Next(0, sizeA-100);
            sizeB = sizeA - div;    //sizeB always <= sizeA
            kth = r.Next(sizeB/2, sizeA + sizeB);

            //case 1
            //kth = sizeB / 2;
            //case 2
            //kth = (sizeA + sizeB) / 2;
            //case 3
            //kth = sizeA + sizeB / 2;

            //A even
            //B odd
            int val = r.Next(0, 100000);
            int inc = 0;
            for (int i = 0; i < sizeA; i++)
            {
                val += inc;
                if (val % 2 != 0)
                {
                    val--;
                }
                if (val < 0)
                {
                    throw new Exception("Error: number too big");
                }
                inc = r.Next(2, 10);
                listA.Add(val);
            }

            val = r.Next(0, 100000);
            inc = 0;
            for (int i = 0; i < sizeB; i++)
            {
                val += inc;
                if (val % 2 == 0)
                {
                    val--;
                }
                if (val < 0)
                {
                    throw new Exception("Error: number too big");
                }
                inc = r.Next(2, 100);
                listB.Add(val);
            }
        }

        static void Init1()
        {
            sizeA = 1000;
            sizeB = 500;
            //case 1
            //kth = 100;
            //case 2
            //kth = 800;
            //case 3
            kth = 1200;
            int A0 = 0;
            int B0 = 111110;
            listA = new List();
            listB = new List();
            for (int i = 0; i < sizeA; i++)
            {
                listA.Add(A0+2 * i);
            }
            for (int i = 0; i < sizeB; i++)
            {
                listB.Add(B0+2 * i + 1);
            }
        }

        static void Solution()
        {
            DateTime dt1 = DateTime.Now;

            if (kth  sizeA)
            {
                Console.WriteLine("Case 3:");
                int kkth = sizeA + sizeB + 1 - kth;
                //kkth will be <= sizeB
                //find the kkth largest means the kkth+1 smallest
                FindSmallest(kkth+1, sizeA - kkth, sizeA - 1, sizeB - kkth, sizeB - 1);
            }
            else //sizeB < kth  listB[rightB])
            {
                Console.WriteLine("A>B");
                //B < A
                if (ord  listA[leftA - 1])
                    {
                        Console.WriteLine("result: {0}", listB[rightB]);
                    }
                    else
                    {
                        Console.WriteLine("result: {0}", listA[leftA - 1]);
                    }
                }
                else
                {
                    Console.WriteLine("result: {0}", listA[leftA + (ord - size - 1)]);
                }
                return;
            }
            else if (listA[rightA] < listB[leftB])
            {
                Console.WriteLine("A<B");
                //A < B
                if (ord  listB[leftB - 1])
                    {
                        Console.WriteLine("result: {0}", listA[rightA]);
                    }
                    else
                    {
                        Console.WriteLine("result: {0}", listB[leftB - 1]);
                    }
                }
                else
                {
                    Console.WriteLine("result: {0}", listB[leftB + (ord - size - 1)]);
                }
                return;
            }

            while (true)
            {
                int midA = (leftA + rightA)/2;
                int midB = offsetB + (ord - 1 - (midA-offsetA));
                int ret = CheckRelation(midA, midB);
                if (ret == 0) {
                    return;
                }
                else if (ret > 0)
                {
                    leftA = midA;
                }
                else
                {
                    rightA = midA;
                }
            }

        }

        static int CheckRelation(int midA, int midB)
        {
            if (listA[midA]  listB[midB - 1])
                {
                    Console.WriteLine("result: {0}", listA[midA]);
                    return 0;
                }
                else
                {
                    // smallest should be above midA
                    // or largest should be below midA
                    return 1;
                }
            }
            else // listA[midA] > listB[midB]
            {
                if (listA[midA - 1] < listB[midB])
                {
                    Console.WriteLine("result: {0}", listB[midB]);
                    return 0;
                }
                else
                {
                    //smallest should be below midA
                    // or largest should be above midA
                    return -1;
                }
            }
        }

        static void Solution2()
        {
            DateTime dt1 = DateTime.Now;
            int iA = 0, iB = 0;
            int i = 0;
            bool isA;
            do
            {
                if (iA < sizeA && iB < sizeB)
                {
                    if(listA[iA] < listB[iB])
                    {
                        iA++;
                        i++;
                        isA = true;
                    }
                    else
                    {
                        iB++;
                        i++;
                        isA = false;
                    }
                }
                else if (iA == sizeA)
                {
                    iB++;
                    i++;
                    isA = false;
                }
                else
                {
                    iA++;
                    i++;
                    isA = true;
                }
            } while (i < kth);
            if (isA)
            {
                Console.WriteLine("Verify: {0}", listA[--iA]);
            }
            else
            {
                Console.WriteLine("Verify: {0}", listB[--iB]);
            }
            DateTime dt2 = DateTime.Now;
            TimeSpan ts = dt2 - dt1;
            Console.WriteLine("duration={0}", ts);
        }
    }
}
```

Reply ↓ Report user

0

cpthk

May 15, 2013 at 3:53 pm

Really nice post. But I do have 1 questions.

I am not convienced that 3rd solution O(lg m + lg n) is *always* better than 2nd solution O(k).

I do agree that O(log m) is better than O(m), but I do not agree that O(log m) is *always* better than O(k).

If k = 3, m = 100000, n = 100000

Using the 2nd solution, you would only need to traverse 3 steps to get the answer

Using the 3rd solution, in the worst case, you would need to traverse log(100000) + log(100000).

It is obvious that 3 < log(100000) + log(100000).

In this case, it is obvious that 2nd solution would run faster.

Am I correct?

Reply ↓ Report user

0

frank

May 17, 2013 at 9:14 am

My algorithm always ensures the time complexity is O(lg(min(m,n,k))).

Assuming m>=n, so there are 3 possibilities:

1. k<=n,

we only consider A[0..k-1] and B[0..k-1]. So it is O(lg(k))

2. n<km

since k smallest is the same as m+n+1-k largest, and value of m+n+1-k is always <= n, Set m+n+1-k = p. We only need to consider the interval A[m-p, m] and B[n-p, n]. And the time complexity is O(lg(p)), i.e., O(lg(n)).

Reply ↓ Report user 0 👍 👎

frank

May 17, 2013 at 8:38 am

c++ code. It works fine in visual studio, but failed on leetcode because leetcode doesn't support time class.

```
#include
#include
#include

using namespace std;

static int sizeA;
static int sizeB;
static int kth;
static int* listA;
static int* listB;

int genRand(int, int);
int init();
void solution();
void solution2();
void findSmallest(int, int, int, int, int);
int checkRelation(int, int);

int genRand(int min, int max)
{
    int range = max-min+1;
    return (rand() % range + min);
}

int init()
{
    srand(time(NULL));
    sizeA = genRand(10000000, 100000000);
    //sizeA = genRand(1000, 10000);
    listA = new int[sizeA];
    int div = genRand(0, sizeA-100);
    sizeB = sizeA - div;
    listB = new int[sizeB];
    kth = genRand(sizeB/2, sizeA + sizeB);

    cout << "sizeA=" << sizeA << ", sizeB="<< sizeB << ", k=" << kth << "\n";
    int val = genRand(0, 100000);
    int inc = 0;
    for(int i=0; i<sizeA; i++)
    {
        val += inc;
        if (val % 2 != 0)
        {
            val--;
        }
        if (val < 0)
        {
            cerr << "Error: number too big";
            return -1;
        }
        inc = genRand(2, 10);
        listA[i] = val;
    }
    val = genRand(0, 100000);
    inc = 0;
    for (int i = 0; i < sizeB; i++)
    {
        val += inc;
        if (val % 2 == 0)
        {
            val--;
        }
        if (val < 0)
        {
            cerr << "Error: number too big";
            return -1;
        }
        inc = genRand(2, 100);
        listB[i] = val;
    }
    return 0;
}

int main()
{
    if(init() != 0)
    {
        return 0;
    }
    solution();
    solution2();
    return 0;
}

void solution()
{
    clock_t start, end;
    start = clock();
    if(kth <= sizeB)
    {
        cout <sizeA)
    {
        cout << "Case 3:\n";
        int kkth = sizeA + sizeB + 1 - kth;
        //find the kkth largest means the kkth+1 smallest
        findSmallest(kkth+1, sizeA - kkth, sizeA - 1, sizeB - kkth, sizeB - 1);
    }
    else //sizeB < kth <= sizeA
    {
        cout << "Case 2:\n";
        findSmallest(sizeB, kth - sizeB, kth - 1, 0, sizeB-1);
    }
    end = clock();
    cout << "duration(clicks)=" << end-start < listB[rightB])
    {
        cout <B\n";
        //B < A
        if (ord < size)
        {
            cout << "result: " << listB[leftB + ord - 1]< listA[leftA - 1])
            {
                cout << "result: " << listB[rightB]<< "\n";
            }
            else
            {
                cout << "result: " << listA[leftA - 1]<< "\n";
            }
        }
        else
        {
            cout << "result: " << listA[leftA + (ord - size - 1)] << "\n";
        }
        return;
    }
    else if (listA[rightA] < listB[leftB])
    {
        cout << "A<B\n";
        //A < B
        if (ord < size)
        {
            cout << "result: " << listB[leftA + ord - 1]< listB[leftB - 1])
            {
                cout << "result: " << listA[rightA]<< "\n";
            }
            else
            {
                cout << "result: " << listB[leftB - 1]<< "\n";
            }
        }
        else
        {
            cout << "result: " << listB[leftB + (ord - size - 1)]< 0)
        {
            leftA = midA;
        }
        else
        {
            rightA = midA;
        }
    }

}

int checkRelation(int midA, int midB)
{
    if (listA[midA]  listB[midB - 1])
        {
            cout << "result: "<< listA[midA]< listB[midB]
    {
        if (listA[midA - 1] < listB[midB])
        {
            cout << "result: " << listB[midB]<< "\n";
            return 0;
        }
        else
        {
            //smallest should be below midA
            // or largest should be above midA
            return -1;
        }
    }
}

void solution2()
{
    clock_t start, end;
    int iA = 0, iB = 0;
    int i = 0;
    bool isA;

    start = clock();
    do
    {
        if (iA < sizeA && iB < sizeB)
        {
            if(listA[iA] < listB[iB])
            {
                iA++;
                i++;
                isA = true;
            }
            else
            {
                iB++;
                i++;
                isA = false;
            }
        }
        else if (iA == sizeA)
        {
            iB++;
            i++;
            isA = false;
        }
        else
        {
            iA++;
            i++;
            isA = true;
        }
    } while (i < kth);
    if (isA)
    {
        cout << "Verify: " << listA[--iA]<< "\n";
    }
    else
    {
        cout <<"Verify: " << listB[--iB]<< "\n";
    }

    end = clock();
    cout << "duration2(clicks)=" << end-start<< "\n";
}
```

Reply ↓ Report user

0

codermojo

June 2, 2013 at 12:57 pm

Can we extend this to find the median of two sorted Arrays(with unique numbers) ?

Reply ↓ Report user

0

Pingback: Find the k-th Smallest Element in the Union of Two Sorted Arrays | This Dongfeng Han's Blog. Welcome!

Pingback: Find kth small/largest element in two sorted array | cloris1000

ZhigangZhao

July 29, 2013 at 2:18 am

i implemented the O(log(min(m,n,k)) algorithm,here is the code below , go to tanliboy ‘s post for detail.

```
int findK(int A[],int alen,int B[],int blen,int K)
{
    assert(K<=alen+blen && K);
    if(alen == 0)return B[K-1];
    if(blen == 0) return A[K-1];
    if(K == 1)return min(A[0],B[0]);
    int subK = (K/2)*2==K?K/2:(K-1)/2;
    int asub,bsub;
    asub = min(subK,alen);
    bsub = min(subK,blen);
    if(A[asub-1] < B[bsub-1])
        return findK(A+asub,alen-asub,B,blen,K-asub);
    else return findK(A,alen,B+bsub,blen-bsub,K-bsub);
}
```

Reply ↓

+2

Nigel

August 20, 2013 at 10:24 pm

Hi 1337c0d3r,

I have a question on the recursive call you make at the end.

if (Ai < Bj)
// exclude Ai and below portion
// exclude Bj and above portion
return findKthSmallest(A+i+1, m-i-1, B, j, k-i-1);

The first arguement we pass for this function is a array (int[] A). Then, How do you pass A+i+1 is the recursive call. ? I understand, you only trying to pass the part of array A from a position beyond i. But A+i+1 refers to an element and not the array.

Thank You in advance.

Reply ↓ 0 👍 👎

fentoyal
September 2, 2013 at 12:01 pm

The best solution is O(lgK), not O(lgM+lgN). Usually lgM + lgN may be much bigger than lgK.
And O(lgk) solution is not hard. but a good and robust implementation is non-trivial.

Reply ↓ 0 👍 👎

Maja Grubic
September 8, 2013 at 10:01 am

Can you please clarify how you came up with the formula for i?
int i = (int)((double)m / (m+n) * (k-1));

Reply ↓ Report user 0 👍 👎

Aadi
September 9, 2013 at 7:30 pm

very nicely explain, thank you.

Reply ↓ 0 👍 👎

HengZhang
September 22, 2013 at 9:03 pm

```
def kthelementintwounionsorteda(a,b,k):
    la=len(a)
    lb=len(b)
    left=0
    right=la-1
    middle=-1

    leftb=rightb=-1

    while leftk-1:
            if pbleftb:
                leftb=pb
            left=middle+1

    if leftb==0 and rightb==-1:
        rightb = lb-1

    print('leftb'+str(leftb)+' rightb'+str(rightb))



    while leftbk-1:
            rightb=middle-1
        else:
            leftb=middle+1

    return None
```

```
#ascending
def findIndexInSorted(sorted,a):
    slen=len(sorted)
    if a >= sorted[slen-1]:
        return slen

    if a<= sorted[0]:
        return 0

    left=0
    right=slen-1
    middle=int((left+right)/2)
    while lefta:
            if sorted[middle-1]<a> a:
                return middle
            left=middle+1
            middle=int((left+right)/2)

    return left
```

Reply ↓ Report user 0 👍 👎

Xu Zhang

October 2, 2013 at 3:59 am

Here is a simpler O(log(k)) solution:

```
public double findKthInSortedArrays(int A[], int B[], int k) {
    if (A == null)  A = new int[0];
    if (B == null)  B = new int[0];
    int nA = A.length;
    int nB = B.length;
    if (nA == 0 && nB == 0) return Integer.MIN_VALUE;

    int l = k - Math.min(k, nB);
    int r = Math.min(k, nA);

    while(l <= r) {
        int i = l + (r - l) / 2;
        int j = k - i;

        int a_i = (i  0) ? A[i-1] : Integer.MIN_VALUE;
        int b_j = (j  0) ? B[j-1] : Integer.MIN_VALUE;

        if (a_i >= b_j_prev && b_j >= a_i_prev) {
            return Math.max(a_i_prev, b_j_prev);
        }

        if (a_i < b_j_prev) {
            l = i + 1;
        } else {
            r = i - 1;
        }
    }

    return Integer.MIN_VALUE;
}
```

Reply ↓ Report user

0 👍 👎

Xu Zhang

October 2, 2013 at 4:02 am

Don’t know why two lines are missing. Working code:

```
public double findKthInSortedArrays(int A[], int B[], int k) {
    if (A == null)  A = new int[0];
    if (B == null)  B = new int[0];
    int nA = A.length;
    int nB = B.length;
    if (nA == 0 && nB == 0) return Integer.MIN_VALUE;

    int l = k - Math.min(k, nB);
    int r = Math.min(k, nA);

    while(l <= r) {
        int i = l + (r - l) / 2;
        int j = k - i;

        int a_i = (i  0) ? A[i-1] : Integer.MIN_VALUE;
        int b_j = (j  0) ? B[j-1] : Integer.MIN_VALUE;

        if (a_i >= b_j_prev && b_j >= a_i_prev) {
            return Math.max(a_i_prev, b_j_prev);
        }

        if (a_i < b_j_prev) {
            l = i + 1;
        } else {
            r = i - 1;
        }
    }

    return Integer.MIN_VALUE;
}
```

Reply ↓ Report user 0

Xu Zhang
October 2, 2013 at 4:03 am

Still missing two lines. Plain text as below:

```
public double findKthInSortedArrays(int A[], int B[], int k) {
if (A == null) A = new int[0];
if (B == null) B = new int[0];
int nA = A.length;
int nB = B.length;
if (nA == 0 && nB == 0) return Integer.MIN_VALUE;

int l = k – Math.min(k, nB);
int r = Math.min(k, nA);

while(l <= r) {
int i = l + (r – l) / 2;
int j = k – i;

int a_i = (i 0) ? A[i-1] : Integer.MIN_VALUE;
int b_j = (j 0) ? B[j-1] : Integer.MIN_VALUE;

if (a_i >= b_j_prev && b_j >= a_i_prev) {
return Math.max(a_i_prev, b_j_prev);
}

if (a_i < b_j_prev) {
l = i + 1;
} else {
r = i – 1;
}
}
```

return Integer.MIN_VALUE;

}

Reply ↓ Report user

0

Xu Zhang

October 2, 2013 at 4:11 am

So sorry for the messed up code. Seems that content between ” are omitted automatically.

```
public double findKthInSortedArrays(int A[], int B[], int k) {
    if (A == null)  A = new int[0];
    if (B == null)  B = new int[0];
    int nA = A.length;
    int nB = B.length;
    if (nA == 0 && nB == 0) return Integer.MIN_VALUE;

    int l = k - Math.min(k, nB);
    int r = Math.min(k, nA);

    while(l = b_j_prev && b_j >= a_i_prev) {
            return Math.max(a_i_prev, b_j_prev);
        }

        if (a_i < b_j_prev) {
            l = i + 1;
        } else {
            r = i - 1;
        }
    }

    return Integer.MIN_VALUE;
}
```

Reply ↓ Report user

0

Xu Zhang

October 2, 2013 at 4:23 am

So sorry:(

Check http://discuss.leetcode.com/questions/142/median-of-two-sorted-arrays?page=1&focusedAnswerId=2695#2695 to see definition lines for a_i, b_j, a_i_prev and b_j_prev.

Reply ↓ Report user

0

Pingback: CopyQuery | Question & Answer Tool for your Technical Queries

Lea

November 6, 2013 at 2:54 am

Hi! Firstly, it’s a great algorithm, well done. And now, I could use some help. I need to modify this algorithm to work with duplicate elements with keeping complexity O(log(m)+log(n)). For example if A={2,4,5,8} and B={1,2,5,9,10} and I want to find 4 smallest element, my answer should be 5 (union should be {1,2,4,5,8,9,10}). I would be grateful for all help.

Reply ↓

0

Ishtiaque
December 5, 2013 at 7:50 pm

“We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj < Ai-1. Why?"

Shouldn't it be '<=' ?

"We make an observation that when Ai <= Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj <= Ai-1. Why?"

Thank you!

Reply ↓

0

Pingback: Find Median of Two Sorted Arrays | Little House

gli00001
December 11, 2013 at 9:19 pm

Martin’s idea is great, but the algo has bugS,
first,
if (a_offset + b_offset == k) return A[a_offset]; should be
if (a_offset + b_offset == k) return A[a_offset -1];

second,
after a_offset++, need check if outofboundary,
try {1},{2}, k=1 and see 1st bug,
try {1},{2}, k=2 and see 2nd bug,

Reply ↓

0

_chills

January 13, 2014 at 10:22 pm

This iterative code seems to be working for me with O (log k) complexity. Can someone verify?

```
public int getK(int[] A, int[] B, int k) {
        if (k > (A.length + B.length)) {
            throw new IllegalArgumentException("K is bigger than arrays combined");
        }
        int count = 0; int startA = 0; int startB = 0, retVal = -1;
        while (k > 0) {
            int a = k/2;
            int b = k - a;
            if (a > 0) {    --a;    }
            if (b > 0) {    --b;    }

            //debug("k = " + k + ", a = " + a + ", b = " + b + ", startA = " + startA + ", st

            if ((A.length-1) < (startA+a)) {
                return B[startB+k-1];
            }
            if ((B.length-1)  B[startB+b]) {
                retVal = B[startB+b];
                k -= (b+1);
                startB += (b+1);
            } else {
                retVal = A[startA+a];
                k -= (a+1);
                startA += (a+1);
            }
        }
        return retVal;
    }
```

Reply ↓

0

brainless

January 28, 2014 at 4:27 am

Have you tried this test case?

A[] = {1,2,3,4,5};

B[] = {6,7};

and k = 7;

Reply ↓ Report user

0

Ashish

February 2, 2014 at 5:36 pm

We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj <

Ai-1. W

this doesnt work for this case:

array A {1,2,3,7,12,15}

array B {0,4,5,6,9}

in this case middle of A is 7 and B is 5. and the condition that Bj < Ai then Bj< Ai-1 should be true doesn't work.

Am I right or wrong?

Reply ↓

0

Armand

February 8, 2014 at 4:14 pm

I do not even know how I ended up here, but I thought this post was great.
I do not know who you are but certainly you’re going to a famous blogger
if you aren’t already 😉 Cheers!

Reply ↓

0

Max

February 16, 2014 at 10:46 pm

not so neat

```
int findOpponentIndex(int a[], int s, int e, int key)
{
    while(s key)
            e = m;
        else
            s = m+1;
    }
    return s;
}

int recFindKthElementInTwoSortArray(int arrA[], int arrB[], int startA, int endA, int startB,
{
    if(startA == endA){
        //find arrA[startA]'s position in arrB through binary search method o(log n)
        int p = findOpponentIndex(arrB, startB, endB, arrA[startA])-startB;
        if(k == p)
            return arrA[startA];
        else
            return k > p ? arrB[k-1+startB] : arrB[k+startB];
    }else if(startB == endB){
        //find arrB[startB] 's position in arrA through binary search method o(log n)
        int p = findOpponentIndex(arrA, startA, endA, arrB[startB])-startA;
            if(k == p)
                return arrB[startB];
```

```
      else
        return k > p ? arrA[k-1+startA] : arrA[k+startA];
  }else
  {
    int midA = (startA+endA)/2;
    int midB = (startB+endB)/2;
    int l, r=0;
    if(arrA[midA] < arrB[midB]){
      l = findOpponentIndex(arrB, startB, midB, arrA[midA]);
      r = findOpponentIndex(arrA, midA, endA, arrB[midB]);
      //split two array into left middle and right part
      int leftParts = (midA-startA+1 + l-startB+1);
      int rightParts = r-startA+midB-startA+1;
      if(k+1 <= leftParts)
        return recFindKthElementInTwoSortArray(arrA, arrB, startA, midA, startB, l, k
      if(rightParts <= k)
        return recFindKthElementInTwoSortArray(arrA, arrB, r, endA, midB+1, endB, k-r

      return recFindKthElementInTwoSortArray(arrA, arrB, midA+1, r-1, l+1, midB-1, k-le

    }else if(arrB[midB] < arrA[midA]){

      l = findOpponentIndex(arrA, startA, midA, arrB[midB]);
      r = findOpponentIndex(arrB, midB, endB, arrA[midA]);

      //split two array into left middle and right part
      int leftParts = (midB-startB+1 + l-startA+1);
      int rightParts = r-startB+midA-startA+1;
      if(k+1 <= leftParts)
          return recFindKthElementInTwoSortArray(arrA, arrB, startA, l, startB, midB
      if(rightParts < k)
          return recFindKthElementInTwoSortArray(arrA, arrB, midA+1, endA, r, endB,

      return recFindKthElementInTwoSortArray(arrA, arrB, l+1, midA-1, midB+1, r-1, k-le
    }else{
      if(k == (midA-startA+midB-startB))
        return arrA[midA];
      return k  lenA+lenB || k < 0)
    return -1;

  if(arrA[lenA-1] < arrB[0]){
    return k <= lenA ? arrA[k-1] : arrB[k-lenA-1];
  }else if(arrB[lenB-1] < arrA[0]){
    return k <= lenB ? arrB[k-1] : arrA[k-lenB-1];
  }else{
    //indices from 0 to len-1, so k will be k-1
    return recFindKthElementInTwoSortArray(arrA, arrB, 0, lenA-1, 0, lenB-1, k-1);
  }
}
```

Reply ↓ Report user

0

MarwanSdeek
February 23, 2014 at 7:19 am

i can’t find 3rd smallest element using code with this test case findKthSmallest([1, 3, 4, 7, 9], 5, [2, 3, 4, 7], 4, 3)

because there is no condestion about if Ai==Bj

Thanks

Reply ↓

0

Dun Liu

March 18, 2014 at 2:42 pm

I have a better solution. Run tim is O(lgk). O(lgk) is better than O(lgm+lgn) since k<m*n

```
int findKthElement(int k, int*array1, int start1, int*array2, int start2) {
    // if (k>m+n) exception
    if (k ==0) {
        return (array1[start1] < array2[start2]) ?array1[start1] :array2[start2];
    }
    int mid = round(k/2.0);
    if (array1[start1+mid] < array2[start2+mid]) {
        return findKthElement(k-mid ,array1, start1+mid, array2, start2);
    } else {
        return findKthElement(k-mid ,array1, start1, array2, start2+mid);
    }
}
```

Reply ↓

+1

Dun Liu

March 18, 2014 at 10:26 pm

I just checked ZhigangZhao’s solution. Then I found mine previous solution have out of bound exception.

Here’s my new solution:

```
int findKthElement(int k, int*array1, int start1, int end1, int*array2, int start2, int end2)
    // if (k>m+n) exception
    if (k ==0) {
        return MIN(array1[start1], array2[start2]);
    }
    if (start1 == end1) {
        return array2[k];
    }
    if (start2 == end2) {
        return array1[k];
    }
    int mid = round(k/2.0);
    int sub1 = MIN(mid, end1-start1);
    int sub2 = MIN(mid, end2-start2);
    if (array1[start1+sub1] < array2[start2+sub2]) {
        return findKthElement(k-mid ,array1, start1+sub1, end1, array2, start2, end2);
    } else {
        return findKthElement(k-mid ,array1, start1, end1, array2, start2+sub2, end2);
    }
}
```

Test:

```
        int a1[] = {1, 3, 6, 8, 9, 13, 18};
    int a2[] = {2, 5, 6, 9, 10, 11, 34, 67, 90};
    NSLog(@"%d", findKthElement(15, a1, 0, 6, a2, 0, 8));
        NSLog(@"%d", findKthElement(1, a1, 0, 6, a2, 0, 8));
        NSLog(@"%d", findKthElement(0, a1, 0, 6, a2, 0, 8));
        NSLog(@"%d", findKthElement(6, a1, 0, 6, a2, 0, 8));
        NSLog(@"%d", findKthElement(16, a1, 0, 6, a2, 0, 8));
```

ZhigangZhao said its run time is O(log(min(m,n,k)) . I don't think so. I think its run time is O(logk).

Reply ↓

+1

Avaln

May 13, 2014 at 1:58 pm

Just want to add a comment to further explain how to slit those two arrays.
When Ai doesn't fall between Bj-1 and Bj, it means that Ai is not the i+j+1th smallest element. There might be an element k (0 <= k <= j-1) that satisfy Bk-1 < Ai < Bk, so Ai is actually the i+k+1th smallest element, which is smaller than i+j+1. This meaning, the element we are looking for must be greater than the current i if its' in array A. Since we also have the invariant i+j=k-1, if we increase i in array A, we have to decrease j in array B, that's why we discard A0 to Ai and also Bj to Bn.

Reply ↓ Report user

0

Prateek

May 22, 2014 at 7:55 am

In the given line above you need to correct the conditions, if Ai < Bj then Ai < B(j+1) and next expression as feel

Errorenous Line:
We make an observation that when Ai < Bj, then it must be true that Ai < Bj-1. On the other hand, if Bj < Ai, then Bj < Ai-1. Why?

Reply ↓

0

Dixit Gokhale

July 14, 2014 at 5:14 am

Here’s my solution:

```
public static int kthSmallestOf2SortedList(int[] list1, int[] list2, int k)
    {
        int m = list1.length;
        int n = list2.length;

        int i= 0, j=0, count=0,kthSmallest,tempSmall = 0;
        while(count < k)
        {

            if(i < m && j  list2[j])
                {
                    tempSmall = list2[j];
                    j++;
                }
                else
                {
                    tempSmall = list1[i];
                    i++;
                }

            }
            else if(i >= m)
            {
                j = j + ( k - count - 1);
                if(j >= n)
                {
                    tempSmall = -1;
                    break;
                }
                else
                    tempSmall = list2[j];
            }
            else if(j >= n)
            {
                i = i + ( k - count - 1);
                if(i >= m)
                {
                    tempSmall = -1;
                    break;
                }
                else
                    tempSmall = list1[i];
            }

            count++;
        }

        kthSmallest = tempSmall;

        return kthSmallest;
    }
```

Reply ↓

0

Ramya

July 26, 2014 at 3:30 pm

Hi ,
I did not go through all the comments above but if any of you has already mentioned my solution, pls let me know if it would work .

According to the problem , we know m, n and k so we can guess from this that k< (m+n) then we can find out if k< (m+n)/2 and keep dividing by 2 again to check if k<(m+n)/4 .

Suppose if, k (m+n)/4 then we get an interval

(m+n)/4 <k (m/4)+(n/4) <k < (m/2)+(n/2)

so now we can set the starting pointer of A to position(m/4) and search till (m/2) and similarly for B.

The time complexity wld be much better that O(k) , according to my calculations, i believe it would be O(log m +log n) similar to the last method mentioned in the explanation.

Reply ↓

0

Ramya
July 26, 2014 at 3:32 pm

Sorry for the error,

I meant

(m+n)/4 <k < (m+n)/2
=(m/4)+(n/4) <k < (m/2)+(n/2)

Reply ↓

0

Peeyush
July 29, 2014 at 4:10 pm

Do we consider in the (log m + log n) solution that the K is less than both m and n? if it is so then we can have a solution with (log k) complexity

Reply ↓

0

calche93
September 7, 2014 at 8:06 am

Hi 1337c0d3r, your solution seems to be right. Thank you for such insightful explanation. I was wondering if there is O(log(K)) , that works for all cases. No limitation like length of 1st array should be less than the 2nd or some other limitation.

Help is appreciated. Thanks.

Reply ↓

0 👍 👎

Chris Zhang

September 13, 2014 at 11:38 am

```
package kthsmallest;
import java.lang.Math;

/**
*
* @author xz2210
*/
public class Kthsmallest {

/**
* @param args the command line arguments
*/
public static void main(String[] args) {
// TODO code application logic here
int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
int[] B={3, 4, 6, 9, 281, 289};
int k0 = 13;
Kthsmallest obj = new Kthsmallest();
System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B.length-1, k0-1), k0));
}

public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
if(xub + yub <k-2) return 0;
if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
if(X.length==0){
return Y[k-1];
}
if(Y.length==0){
return X[k-1];
}

int xpiv = (int)Math.ceil((xlb+xub)/2);
xpiv = Math.min(xpiv, k-2);
xpiv = Math.max(xpiv, k-1-Y.length);
int ypiv = k – xpiv – 2;

// Constri
// 0=<k-xpiv-2<=Y.length -1
// xpiv=k-1-Y.length
// k-1-Y.length Y.length>=1

if (X[xpiv]Y[ypiv])
return Y[ypiv];
else
return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
}

//if (X[xpiv]>Y[ypiv])
else{
if(X[xpiv]<Y[ypiv+1])
return X[xpiv];
else
return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
}
}
}
```

Reply ↓ Report user

0

Chris Zhang

September 13, 2014 at 11:45 am

```
package kthsmallest;
import java.lang.Math;



public class Kthsmallest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
        int[] B={3, 4, 6, 9, 281, 289};
        int k0 = 13;
        Kthsmallest obj = new Kthsmallest();
        System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B
    }

    public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
        if(xub + yub <k-2) return 0;
        if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
        if(X.length==0){
            return Y[k-1];
        }
        if(Y.length==0){
            return X[k-1];
        }

        int xpiv = (int)Math.ceil((xlb+xub)/2);
        xpiv = Math.min(xpiv, k-2);
        xpiv = Math.max(xpiv, k-1-Y.length);
        int ypiv = k - xpiv - 2;
//        Constraints
//        0=<k-xpiv-2<=Y.length -1
//        xpiv=k-1-Y.length
//        k-1-Y.length Y.length>=1

        if (X[xpiv]Y[ypiv])
                return Y[ypiv];
            else
                return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
        }

        //if (X[xpiv]>Y[ypiv])
        else{
            if(X[xpiv]<Y[ypiv+1])
                return X[xpiv];
            else
                return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
        }
    }
}
```

Reply ↓ Report user

0

Chris Zhang

September 13, 2014 at 12:01 pm

```
package kthsmallest;
import java.lang.Math;



public class Kthsmallest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
        int[] B={3, 4, 6, 9, 281, 289};
        int k0 = 13;
        Kthsmallest obj = new Kthsmallest();
        System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B
    }
    //refine the search range of X, Y for the kth smallest number
    //xlb: lower bound of X
    //xub: upper bound of X
    //ylb: lower bound of Y
    //yub: upper bound of Y
    public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
        //If the total # of elements in X, Y is less than k, invalid
        if(xub + yub <k-2) return 0;
        //If the total # of elements match with k, return the bigger number of X's or Y's las
        if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
        //Otherwise, if X is empty, return Y[k-1]
        if(X.length==0){
            return Y[k-1];
        }
        if(Y.length==0){
            return X[k-1];
        }

        //Pick the new pivot in X, Y to refine the search range based on binary search concep
        int xpiv = (int)Math.ceil((xlb+xub)/2);
        //Constraints for ypiv calculated based on xpiv
        //0=<k-xpiv-2<=Y.length -1
        //xpiv=k-1-Y.length
        //so, in general, k-1-Y.length Y.length>=1
        //refine xpiv based on constraints above
        xpiv = Math.min(xpiv, k-2);
        xpiv = Math.max(xpiv, k-1-Y.length);
        int ypiv = k - xpiv - 2;

        if (X[xpiv]Y[ypiv])
                return Y[ypiv];
            else
                return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
        }
        //if (X[xpiv]>Y[ypiv])
        else{
            if(X[xpiv]<Y[ypiv+1])
                return X[xpiv];
            else
                return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
        }
    }
}
```

Reply ↓ Report user

0

Chris Zhang

September 13, 2014 at 12:03 pm

```
package kthsmallest;
import java.lang.Math;



public class Kthsmallest {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
        int[] B={3, 4, 6, 9, 281, 289};
        int k0 = 13;
        Kthsmallest obj = new Kthsmallest();
        System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B
    }

    //refine the search range of X, Y for the kth smallest number
    //xlb: lower bound of X
    //xub: upper bound of X
    //ylb: lower bound of Y
    //yub: upper bound of Y
    public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
        //If the total # of elements in X, Y is less than k, invalid
        if(xub + yub <k-2) return 0;
        //If the total # of elements match with k, return the bigger number of X's or Y's last
        if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
        //Otherwise, if X is empty, return Y[k-1]
        if(X.length==0){
            return Y[k-1];
        }
        if(Y.length==0){
            return X[k-1];
        }

        //Pick the new pivot in X, Y to refine the search range based on binary search concept
        int xpiv = (int)Math.ceil((xlb+xub)/2);
        //Constraints for ypiv calculated based on xpiv
        //0=<k-xpiv-2<=Y.length -1
        //xpiv=k-1-Y.length
        //so, in general, k-1-Y.length Y.length>=1
        //refine xpiv based on constraints above
        xpiv = Math.min(xpiv, k-2);
        xpiv = Math.max(xpiv, k-1-Y.length);
        int ypiv = k - xpiv - 2;

        if (X[xpiv]Y[ypiv])
                return Y[ypiv];
            else
                return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
        }
        //if (X[xpiv]>Y[ypiv])
        else{
            if(X[xpiv]<Y[ypiv+1])
                return X[xpiv];
            else
                return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
        }
    }
}
```

Reply ↓ Report user

0 👍 👎

Chris Zhang

September 13, 2014 at 12:04 pm

```
package kthsmallest;
import java.lang.Math;

public class Kthsmallest {

/**
* @param args the command line arguments
*/
public static void main(String[] args) {
// TODO code application logic here
int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
int[] B={3, 4, 6, 9, 281, 289};
int k0 = 13;
Kthsmallest obj = new Kthsmallest();
System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B.length-1, k0-1), k0));
}

//refine the search range of X, Y for the kth smallest number
//xlb: lower bound of X
//xub: upper bound of X
//ylb: lower bound of Y
//yub: upper bound of Y
public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
//If the total # of elements in X, Y is less than k, invalid
if(xub + yub <k-2) return 0;
//If the total # of elements match with k, return the bigger number of X's or Y's last element
if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
//Otherwise, if X is empty, return Y[k-1]
if(X.length==0){
return Y[k-1];
}
if(Y.length==0){
return X[k-1];
}

//Pick the new pivot in X, Y to refine the search range based on binary search concept
int xpiv = (int)Math.ceil((xlb+xub)/2);
//Constraints for ypiv calculated based on xpiv
//0=<k-xpiv-2<=Y.length -1
//xpiv=k-1-Y.length
//so, in general, k-1-Y.length Y.length>=1
//refine xpiv based on constraints above
xpiv = Math.min(xpiv, k-2);
xpiv = Math.max(xpiv, k-1-Y.length);
int ypiv = k – xpiv – 2;

if (X[xpiv] Y[ypiv])
return Y[ypiv];
else
return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
}
//if (X[xpiv]>Y[ypiv])
else{
if(X[xpiv]<Y[ypiv+1])
return X[xpiv];
else
return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
}
}
}
```

Reply ↓ Report user

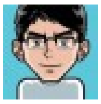
0

Chris Zhang

September 13, 2014 at 12:05 pm

```
package kthsmallest;
import java.lang.Math;



public class Kthsmallest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int[] A={1, 2, 5, 10, 12, 14, 19, 282, 290};
        int[] B={3, 4, 6, 9, 281, 289};
        int k0 = 13;
        Kthsmallest obj = new Kthsmallest();
        System.out.println(obj.Kthsmallest(A, 0, Math.min(A.length-1, k0-1), B, 0, Math.min(B
    }

    //refine the search range of X, Y for the kth smallest number
    //xlb: lower bound of X
    //xub: upper bound of X
    //ylb: lower bound of Y
    //yub: upper bound of Y
    public int Kthsmallest(int[] X, int xlb, int xub, int[] Y, int ylb, int yub, int k){
        //If the total # of elements in X, Y is less than k, invalid
        if(xub + yub <k-2) return 0;
        //If the total # of elements match with k, return the bigger number of X's or Y's las
        if(xub + yub == k-2) return Math.max(Y[yub], X[xub]);
        //Otherwise, if X is empty, return Y[k-1]
        if(X.length==0){
            return Y[k-1];
        }
        if(Y.length==0){
            return X[k-1];
        }

        //Pick the new pivot in X, Y to refine the search range based on binary search concep
        int xpiv = (int)Math.ceil((xlb+xub)/2);
        //Constraints for ypiv calculated based on xpiv
        //0=<k-xpiv-2<=Y.length -1
        //xpiv=k-1-Y.length
        //so, in general, k-1-Y.length Y.length>=1
        //refine xpiv based on constraints above
        xpiv = Math.min(xpiv, k-2);
        xpiv = Math.max(xpiv, k-1-Y.length);
        int ypiv = k - xpiv - 2;

        if (X[xpiv] Y[ypiv])
                return Y[ypiv];
            else
                return Kthsmallest(X, xpiv, xub, Y, ylb, ypiv, k);
        }
        //if (X[xpiv]>Y[ypiv])
        else{
            if(X[xpiv]<Y[ypiv+1])
                return X[xpiv];
            else
                return Kthsmallest(X, xlb, xpiv, Y, ypiv, yub, k);
        }
    }
}
```

Reply ↓ Report user 0



Ahmed Hamdy

October 3, 2014 at 7:29 am

I solved it with 3 lines only … The algorithm works fine and with the best complexity as you mentioned, the problem might be in the stack size as it is a recursive algorithm.

I am assuming that:

1. k is always bounded between 0 and size of first array + size of second array

2. arrays are full with data, i.e. no null array or empty array passed.

```
public static Integer kthSmallest(int[] a, int[] b, int p1, int p2, int k)
    {
        if (k == 0)
            return (p1 != -1? a[p1] : Integer.MAX_VALUE) <= (p2 != -1? b[p2] : Integer.MAX_VAL

        if ((p1 != -1? a[p1] : Integer.MAX_VALUE) <= (p2 != -1? b[p2] : Integer.MAX_VALUE))
            return kthSmallest(a, b, p1 == a.length -1 ? -1 : ++p1, p2, --k);
        else
```

```
            return kthSmallest(a, b, p1, p2 == b.length - 1 ? -1 : ++p2, --k);
    }
```

Reply ↓ Report user

0

johnson
April 3, 2015 at 9:11 pm

urs is linear time O(m+n), not log(M+N)

Reply ↓ Report user

0

Spring
October 27, 2014 at 4:07 pm

This can be made even more efficient as you only need to look into first k elements of either array instead of m and n, so the complexity would be (O(lg k + lg k))

Reply ↓

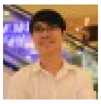
0

priya
January 18, 2015 at 8:28 pm

Can somebody please tell me, how to do the same question for the unsorted arrays.

Reply ↓ Report user

0



Truong Khanh
February 24, 2015 at 9:25 pm

My discussion and java source code can be found here http://www.capacode.com/?p=115

Reply ↓ Report user

0

Akash
March 15, 2015 at 5:20 am

Kudos to you man for the implementation in log m+log n.BTW nice explaination!

Reply ↓

0

johnson

April 3, 2015 at 8:19 pm

I don’t think it’s necessary to bound and shrink the range from BOTH sides, just shrinking from one side is just as fast. since finally you know the number of elements u have to remove from AFTER the 2 final mid points. whether u change the range before those mid points don’t really matter. maybe it helps if you are lucky and hit the mid points early.

Reply ↓ Report user 0

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

To embed your code, please use <code>your code here</code>.

You may use the <code> tag to embed your code.

Post Comment