What is the fastest algorithm for computing the kth smallest element in the union of two sorted lists of size m and n?

John Kurlak, works at Facebook

Updated Jan 15, 2016

Here is a fully coded $O(\lg k)$ solution that addresses as many edge cases as I could think of. It technically finds the kth smallest element of the concatenation of the two sorted lists, but it's easily extendable to the union of two lists if that is what you're really after (Simply remove the duplicate values from the arrays. There are a few ways to do this since the arrays are sorted. One way would be to create a hash table and a counter. Iterate over both arrays in one loop. If the first array's value isn't in the hash table, add it to the hash table, add it to a new first array, and increment your counter. If the second array's value isn't in the hash table, add it to the hash table, add it to a new second array, and increment your counter. When your counter reaches $2k$, you can stop iterating over the arrays. The previous operation can be done in place if you want to optimize for space as well. Then, move elements from the larger list into the smaller list until both lists have k unique elements in them. The reason I didn't code this method is that I don't think the original question poster intended for the union of two arrays. If so, the run-time would be dependent on the number of duplicates in the array, so the worst-case run time would be $O(n)$).

Explanation:

The idea here is to do a modified binary search. There are a few different ways to apply binary search to this problem. I will discuss the simplest approach that I know, which is to binary search for the number of values in the first list that are less than or equal to the kth value.

For example, suppose we have the lists:

$A = [1, 5, 9, 13]$

$B = [3, 4]$

And we want to find the third smallest value. The values before and including the third value from the first list and second list are:

$A = [1, \ldots]$

$B = [3, 4, \ldots]$

The number of values in the first list that are less than or equal to the third value, 4, is 1. We will be binary searching for that value. Let's call the number of values that we select from the first list x.

Now, once we've done our binary search, how do we get our answer? Well, we know that we select a total of k elements from the first and second list. We know that we select x elements from the first list. That means we select $k - x$ elements from the second list. We know the the kth smallest value will be the last element that we select from the first list, or it will be the last element that we select from the second list. Since we know what these elements are (list1[k - 1] and list2[k - x - 1]), we just need to figure out which list has the kth smallest element. The answer is simply the larger of the two values. So in our example, we're deciding between 1 in A and 4 in B, we select the larger, 4. As expected, 4 is the 3rd smallest element.

The first thing we need to decide (and this is probably the hardest part of this algorithm) is the range of values we are binary searching. The values we are searching are the possible number of elements to select from the first list. If the size of the first list is a and the size of the second list is b, our first instinct would be to say that we can select 0 to a elements from the first list. While this is true, we have to be careful... because it isn't true for all possible inputs.

For example, suppose we have the lists:

A = [v1, v2, v3, v4, v5]

B = [u1]

And k is 3.

We can't possibly select 0 elements from A because B doesn't have enough elements to allow us to select k values. Therefore, the lower bound for the number of elements we can select from the first list is $\max(0, k - b)$. When b is larger than k, we *can* select 0 elements from the first list. Otherwise, we have to select at least $k - b$ elements from the first list.

As you might suspect, we run into a similar scenario with the upper bound. For example, suppose we have the lists:

A = [v1, v2, v3, v4, v5]

B = [u1]

And k is 3.

We can't possibly select a values from A because then we would be selecting more than k elements. Therefore, the upper bound for the number of elements we can select from the first list is $\min(a, k)$.

Next, we do binary search as normal over the integers in the range:
$[\min(a, k), \max(0, k - b)]$.

What we need to determine now is when we've found the correct number of elements to select from the first list.

With each iteration of the binary search, we have a guess for the correct number of elements to select from the first list. Let's call that guess x. If that value is correct, then we will select $y = k - x$ values from the second list ($x + y = k$).

When we've found the correct value for x, then the values list1[0...x - 1] and list2[0...y - 1] will all be smaller than or equal to the values list1[x...a-1] and list2[y...b-1]. This is an important observation, and it's crucial that you understand it. I'm basically saying that the first k values must be less than or equal to the remaining $n - k$ values. I hope that makes sense.

We will use this observation to determine when we've selected the first k values. In other words, our binary search is done when all the values from list1[0...x-1] and list2[0...y-1] are less than or equal to the values from list1[x...a-1] and list2[y...b-1]. Since the values in list1 are all monotonically increasing and the values in list2 are all monotonically increasing, then we only need to verify that the values list1[x-1] and list2[y-1] are less than or equal to list2[y] and list1[x]. We can write that in two checks: list1[x - 1] < list2[y] and list2[y - 1] < list1[x].

Therefore, after we have a guess for x and y, we need to find the values at list1[x - 1], list1[x], list2[y - 1], and list2[y]. It's possible that list1[x] and list2[y] are outside of the

Let me know if you have questions!

Java Code:

```java
public class KthSmallestIterative {
    public static void main(String[] args) {
        int[] list1 = new int[] { 3, 4, 10, 23, 45, 55, 56, 58, 60, 65 };
        int[] list2 = new int[] { 3, 3, 3, 15, 16, 28, 50, 70, 71, 72 };
        int k = 13;

        int kthSmallest = kthSmallest(list1, list2, k);
        System.out.println(k + "th smallest is " + kthSmallest);
    }

    public static int kthSmallest(int[] A, int[] B, int k) {
        if (A == null || B == null) {
            throw new IllegalArgumentException("Arrays cannot be null!")
        }

        int a = A.length;
        int b = B.length;

        if (k < 1 || k > a + b) {
            throw new IllegalArgumentException("k is not within range!")
        }

        int minSizeA = Math.max(0, k - b);
        int maxSizeA = Math.min(a, k);

        while (minSizeA <= maxSizeA) {
            int sizeA = minSizeA + (maxSizeA - minSizeA) / 2;
            int sizeB = k - sizeA;
            int indexA = sizeA - 1;
            int indexB = sizeB - 1;
            int indexANext = syzeA;
            int indexBNext = sizeB;
            int valA = (indexA < 0) ? Integer.MIN_VALUE : A[indexA];
            int valB = (indexB < 0) ? Integer.MIN_VALUE : B[indexB];
            int valANext = (indexANext >= a) ? Integer.MAX_VALUE : A[ind
            int valBNext = (indexBNext >= b) ? Integer.MAX_VALUE: B[inde

            if (valA <= valBNext && valB <= valANext) {
                return Math.max(valA, valB);
            } else if (valA > valBNext) {
                maxSizeA = sizeA - 1;
            } else {
                minSizeA = sizeA + 1;
            }
        }

        return 0;
    }
}
```

Output:

```
13th smallest is 55
```