```
        return getUpMedian(shorts, 0, s - 1, longs, kth - s, kth - 1);
    }
```

两个有序数组间相加和的 TOP *K* 问题

【题目】

给定两个有序数组 arr1 和 arr2，再给定一个整数 *k*，返回来自 arr1 和 arr2 的两个数相加和最大的前 *k* 个，两个数必须分别来自两个数组。

【举例】

arr1=[1,2,3,4,5]，arr2=[3,5,7,9,11]，*k*=4。
返回数组[16,15,14,14]。

【要求】

时间复杂度达到 $O(k\log k)$。

【难度】

尉 ★★☆☆

【解答】

哪两个分别来自两个排序数组的数相加最大？自然是 arr1 的最后一个数和 arr2 的最后一个数，假设 arr1 长度为 *N*，arr2 长度为 *M*，如图 9-13 所示。

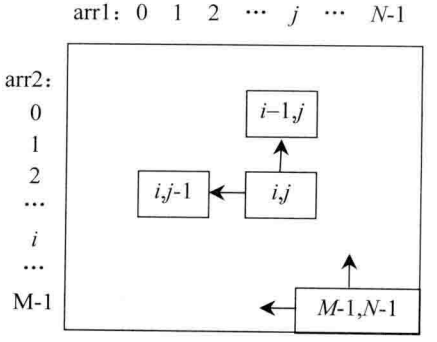

图 9-13

既然 arr2[M-1]+arr1[N-1]无疑是所有和中最大的，那么先把这个和放到大根堆里。然后从堆中弹出一个堆顶，此时这个堆顶肯定是(M-1,N-1)位置的和，即 arr2[M-1]+arr1[N-1]。然后把两个位置的和再放进堆里，分别是(M-2,N-1)和(M-1,N-2)，因为除(M-1,N-1)位置的和之外，其他任何位置的和都不会比(M-2,N-1)和(M-1,N-2)位置的和更大。每放入一个位置的和，都经过堆的调整（heapInsert 调整）。当再从堆中弹出一个堆顶时，此时的堆顶必然是堆中最大的和，假设是(i,j)位置的和。弹出之后再把堆调整成大根堆，即把堆中最后一个元素放到堆顶的位置进行从上到下的 heapify 调整，调整之后再依次把(i,j-1)和(i-1,j)位置的和放入到堆中。也就是说，每次从堆中拿出一个位置和，然后把拿出位置和的左位置和上位置放入到堆里。每次弹出的位置和就是从大到小排列的我们想得到的 Top K。这个过程再次总结为：

1．初始时把位置(M-1,N-1)放入堆中，因为这个位置代表的相加和就是最大的相加和。

2．此时堆顶为(M-1,N-1)，把这个位置代表的相加和(arr2[M-1]+arr1[N-1])收集起来，然后把堆尾放到堆顶的位置，再经历堆的调整(heapify)，最后把(M-2,N-1)和(M-1,N-2)放入堆中，并根据代表的相加和来重新调整堆(heapInsert)。

3．每次堆顶都会有一个位置记为(i,j)，把这个位置代表的相加和(arr2[i]+arr1[j])收集起来，然后把堆尾放到堆顶的位置，再经历堆的调整(heapify)。最后把这个位置上边的(i-1,j)和左边的(i,j-1)放入堆中，并根据代表的相加和调整堆(heapInsert)。

4．直到收集的个数为 k，整个过程结束。

堆的大小为 k，每次堆的调整为 $O(\log K)$级别，并且一共收集 k 个数，所以时间复杂度为 $O(k\log k)$。需要注意的是，要利用哈希表来防止同一个位置重复进堆的情况。

全部过程请参看如下代码中的 topKSum 方法。

```
public class HeapNode {
        public int row;
        public int col;
        public int value;

        public HeapNode(int row, int col, int value) {
                this.row = row;
                this.col = col;
                this.value = value;
        }
}

public int[] topKSum(int[] a1, int[] a2, int topK) {
        if (a1 == null || a2 == null || topK < 1) {
                return null;
        }
```

```
        topK = Math.min(topK, a1.length * a2.length);
        HeapNode[] heap = new HeapNode[topK + 1];
        int heapSize = 0;
        int headR = a1.length - 1;
        int headC = a2.length - 1;
        int uR = -1;
        int uC = -1;
        int lR = -1;
        int lC = -1;
        heapInsert(heap, heapSize++, headR, headC, a1[headR] + a2[headC]);
        HashSet<String> positionSet = new HashSet<String>();
        int[] res = new int[topK];
        int resIndex = 0;
        while (resIndex != topK) {
                HeapNode head = popHead(heap, heapSize--);
                res[resIndex++] = head.value;
                headR = head.row;
                headC = head.col;
                uR = headR - 1;
                uC = headC;
                if (headR != 0 && !isContains(uR, uC, positionSet)) {
                        heapInsert(heap, heapSize++, uR, uC, a1[uR] + a2[uC]);
                        addPositionToSet(uR, uC, positionSet);
                }
                lR = headR;
                lC = headC - 1;
                if (headC != 0 && !isContains(lR, lC, positionSet)) {
                        heapInsert(heap, heapSize++, lR, lC, a1[lR] + a2[lC]);
                        addPositionToSet(lR, lC, positionSet);
                }
        }
        return res;
}

public HeapNode popHead(HeapNode[] heap, int heapSize) {
        HeapNode res = heap[0];
        swap(heap, 0, heapSize - 1);
        heap[--heapSize] = null;
        heapify(heap, 0, heapSize);
        return res;
}

public void heapify(HeapNode[] heap, int index, int heapSize) {
        int left = index * 2 + 1;
        int right = index * 2 + 2;
        int largest = index;
        while (left < heapSize) {
            if (heap[left].value > heap[index].value) {
                largest = left;
            }
            if (right < heapSize && heap[right].value > heap[largest].value) {
                largest = right;
            }
```

```
            if (largest != index) {
                swap(heap, largest, index);
            } else {
                break;
            }
            index = largest;
            left = index * 2 + 1;
            right = index * 2 + 2;
        }
    }

    public void heapInsert(HeapNode[] heap, int index, int row, int col,
            int value) {
        heap[index] = new HeapNode(row, col, value);
        int parent = (index - 1) / 2;
        while (index != 0) {
            if (heap[index].value > heap[parent].value) {
                swap(heap, parent, index);
                index = parent;
                parent = (index - 1) / 2;
            } else {
                break;
            }
        }
    }

    public void swap(HeapNode[] heap, int index1, int index2) {
        HeapNode tmp = heap[index1];
        heap[index1] = heap[index2];
        heap[index2] = tmp;
    }

    public boolean isContains(int row, int col, HashSet<String> set) {
        return set.contains(String.valueOf(row + "_" + col));
    }

    public void addPositionToSet(int row, int col, HashSet<String> set) {
        set.add(String.valueOf(row + "_" + col));
    }
```

出现次数的 TOP *K* 问题

【题目】

给定 String 类型的数组 strArr，再给定整数 k，请严格按照排名顺序打印出现次数前 k 名的字符串。