解法一

Java代码

```
class Solution {
  class node{
    int x,y;
    public node(int x,int y) {
      this.x=x;
      this.y=y;
    }
  }
  class mySort implements Comparator<node>{
    @Override
    public int compare(node o1, node o2) {
      return o1.x+o1.y-o2.x-o2.y;
    }

  }
    public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k)
{

      if(nums1.length==0 || nums2.length==0)
        return new ArrayList<List<Integer>>();
      List<Integer> tmp=new ArrayList<>();
        List<List<Integer>> ans=new ArrayList<>();
        node[] arr=new node[nums1.length*nums2.length];

        int len=0;
        for(int i=0;i<nums1.length;i++)
          for(int j=0;j<nums2.length;j++)
            arr[len++]=new node(nums1[i],nums2[j]);
        Arrays.sort(arr,new mySort());
        for(int i=0;i<Math.min(arr.length, k);i++) {
          tmp.add(arr[i].x);
          tmp.add(arr[i].y);
          ans.add(new ArrayList<>(tmp));
          tmp.clear();
        }

        return ans;
    }
}
```

优化解法

Java代码

```
class Solution {
    public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k)
{
        PriorityQueue<List<Integer>> queue = new PriorityQueue<>(k, (o1, o2) ->
{
            return ((o2.get(0) + o2.get(1)) - (o1.get(0) + o1.get(1)));
        });

        for (int i = 0; i < Math.min(nums1.length, k); i++) {
            for (int j = 0; j < Math.min(nums2.length, k); j++) {
                if (queue.size() != k || nums1[i] + nums2[j] <
queue.peek().get(0) + queue.peek().get(1)) {
                    if (queue.size() == k) queue.poll();
                    List<Integer> pair = new ArrayList<>();
                    pair.add(nums1[i]);
                    pair.add(nums2[j]);
                    queue.add(pair);
                }
            }
        }
        List<List<Integer>> res = new LinkedList<>();
        for (int i = 0; i < k && !queue.isEmpty(); i++) {
            res.add(i, queue.poll());
        }

        return res;

    }
}
```

最优解法

Java代码

```
class Solution {
    public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k)
{
        // 小顶堆
        PriorityQueue<int[]> queue = new PriorityQueue<>(
                (o1, o2) -> (nums1[o1[0]] + nums2[o1[1]]) - (nums1[o2[0]] +
nums2[o2[1]]));
```

```
        List<List<Integer>> res = new LinkedList<>();

       // 两个数组有一个为空，返回空
        if(nums1.length==0 || nums2.length == 0){
            return res;
        }
        // 将我们假想的每个数组的第一项加入小顶堆
        for (int i = 0; i < Math.min(nums1.length, k); i++) {
            queue.add(new int[] { i, 0 }); // 加入的是坐标，小顶堆的比较器也是基于坐标比较
        }

        // 循环k次或者堆空
        while (k > 0 && !queue.isEmpty()) {
            // 弹出堆顶元素
            int[] pair = queue.poll();
            List<Integer> item = new ArrayList<>();
            item.add(nums1[pair[0]]);
            item.add(nums2[pair[1]]);

            // 若我们假想的数组有下一个元素，则加入小顶堆
            if (pair[1] < nums2.length - 1) {
                queue.add(new int[] { pair[0], pair[1] + 1 });
            }
            res.add(item);
            k--;
        }
        return res;
    }
}
```

C++代码

```
class Solution {
public:
    vector<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2, int k) {
        bool exchanged = false;
        if (nums1.size() > nums2.size()) exchanged=true, swap(nums1, nums2);
        vector<vector<int>> ans;
        if (nums1.size() & nums2.size() & k == 0) return ans;
        int n = nums1.size(), m = nums2.size(); // n <= m;
        auto cmp = [&](const pair<int,int>&a, const pair<int,int>&b){
            return nums1[a.first]+nums2[a.second] > nums1[b.first]+nums2[b.second];
        };
        priority_queue<pair<int,int>,vector<pair<int,int>>, decltype(cmp)> q(cmp);
```

```
        for (int i = 0; i < min(k,n); i++){
            q.emplace(i,0);
        }
        while (!q.empty() && k--){
            auto v = q.top(); q.pop();
            ans.push_back(exchanged?vector<int>
({nums2[v.second],nums1[v.first]}):vector<int>({nums1[v.first],
nums2[v.second]}));
            v.second++;
            if (v.second<m) q.emplace(v);
        }
        return ans;
    }
};
```

Python代码

```
class Solution:
    def kSmallestPairs(self, nums1, nums2, k):
        queue = []
        def push(i, j):
            if i < len(nums1) and j < len(nums2):
                heapq.heappush(queue, [nums1[i] + nums2[j], i, j])
        push(0, 0)
        pairs = []
        while queue and len(pairs) < k:
            _, i, j = heapq.heappop(queue)
            pairs.append([nums1[i], nums2[j]])
            push(i, j + 1)
            if j == 0:
                push(i + 1, 0)
        return pairs
```