题目：

Given a *n* x *n* matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,


return 13.
```

Note:

You may assume k is always valid, $1 \le k \le n^2$.

Subscribe to see which companies asked this question.

1.时间：O（（M+N）LOG（M*N））；空间：O（1）

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        if (matrix.empty() || matrix.front().empty()) return -1;
        int lower = matrix.front().front(), upper = matrix.back().back();
```

```
        while (lower < upper){
            int mid = lower + ((upper - lower) >> 1);
            int count = calcLessEqualNum(matrix, mid);
            if (count < k) lower = mid + 1;
            else upper = mid;
        }
        return lower;
    }
private:
    int calcLessEqualNum(const std::vector<std::vector<int>>& matrix, int num){
        int res = 0;
        for (int i = matrix.size() - 1, k = 0; i >= 0 && k < matrix.front().size();){
            if (matrix[i][k] <= num){
                k++;
                res += i + 1;
            } else{
                i -= 1;
            }
        }
        return res;
    }
};
```

2.时间：O（ ）;时间：O（K）

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        if (matrix.empty() || matrix.front().empty()) return -1;
        if (k == 1) return matrix[0][0];
        if (k == matrix.size() * matrix.front().size()) return matrix.back().back();
        const int rows = matrix.size();
        const int cols = matrix.front().size();
        std::priority_queue<int, std::vector<int>, std::greater<int>> queue;
        queue.push(std::numeric_limits<int>::max());
        std::vector<int> vals;
        int row = 0;
        for (int i = 0; i < k; ++i){
            if (row < rows){
                if (matrix[row][0] < queue.top()){
                    for (int j = 0; j < cols; ++j){
                        queue.push(matrix[row][j]);
                    }
                    row++;
                }
            }
```

```
            int tmp = queue.top();
            queue.pop();
            vals.push_back(tmp);
        }
        return vals.back();
    }
};
```