模板：

窗口問題：求最大或最小

2層循環，但指針不回歸，O（n）

for(i=0;i<n;i++){

while(j<n){

if 條件滿足

{ j++; 更新狀態}

else{ break；}

}

更新狀態

}

#### Lint401. Kth Smallest Number in Sorted Matrix

#### 矩陣每行都是sorted，但列不是sorted

#### 用堆解決。建一個類Node記錄x，y坐標和當前值val，一開始把第一列的所有元素放入堆，堆根據val來sort。

#### 每次從堆中poll出最小val的一個Node，然後看看這個這個Node所在行的後一個元素存不存在，存在就加入heap

#### 一共poll K次，第K次就是答案

public class Solution {

/\*\*

\* @param matrix: a matrix of integers

\* @param k: an integer

\* @return: the kth smallest number in the matrix

\*/

class Node{

int x;

int y;

int val;

public Node(int x, int y, int val){

this.x = x;

this.y = y;

this.val = val;

}

}

public int kthSmallest(int[][] matrix, int k) {

// write your code here

int row\_len = matrix.length;

int[] pointers = new int[row\_len];

PriorityQueue<Node> heap = new PriorityQueue<Node>(k, new Comparator<Node>(){

public int compare(Node n1, Node n2){

return n1.val - n2.val;

}

});

//add the first column into heap

for(int i = 0; i < row\_len; i++){

heap.add(new Node(i, 0, matrix[i][0]));

}

//poll k-1 times in total

for(int i = 0; i < k-1; i++){

Node small = heap.poll();

if(small.y + 1 < matrix[0].length){

heap.add(new Node(small.x, small.y+1, matrix[small.x][small.y+1]));

}

}

return heap.poll().val;

}

}

#### Lint401. Two Sum - Greater than target

#### Given an array of integers, find how many pairs in the array such that their sum is bigger than a specific target number. Please return the number of pairs.

Given numbers = [2, 7, 11, 15], target = 24. Return 1. (11 + 15 is the only pair)

思路分析：

暴力法固然可行，但是存在重複掃描。

我們先數組排序

當我們把右邊的元素定下來時，左指針不斷往右掃，掃到第一個滿足sum>target后，左指針不需要回到前邊。例如 arr[right] + arr[left] >sum,到此就可以確定right在這個位置，共有right – left 個符合條件

arr[right -1] +arr[left] 也有可能符合條件

但是arr[right-1] + arr[left-1]< arr[right]+ arr[left-1] <= sum 肯定<=sum所以left不需要回到左邊

#### Lint465. Kth Smallest Sum In Two Sorted Arrays

Given two integer arrays sorted in ascending order and an integer k. Define sum = a + b, where a is an element from the first array and b is an element from the second one. Find the kth smallest sum out of all possible sums.

**Example**

Given [1, 7, 11] and [2, 4, 6].

For k = 3, return 7.

For k = 4, return 9.

For k = 8, return 15.

思路：跟上面的堆解法類似。

對於當前最小的sum的Pair（x,y,sum），之後的sum的可能來源如下，x+1,y x,y+1 或者是之前算過的某個x,y

所以我們需要用一個boolean[][]記錄哪個組合已經放入堆，避免重複插入

最後heap poll 到第K次就是答案

public class Solution {

/\*\*

\* @param A an integer arrays sorted in ascending order

\* @param B an integer arrays sorted in ascending order

\* @param k an integer

\* @return an integer

\*/

class Pair{

int index1;

int index2;

int sum;

public Pair(int index1, int index2, int sum){

this.index1 = index1;

this.index2 = index2;

this.sum = sum;

}

}

public int kthSmallestSum(int[] A, int[] B, int k) {

// Write your code here

int A\_len = A.length;

int B\_len = B.length;

int[] dx = new int[]{0, 1};

int[] dy = new int[]{1, 0};

boolean[][] visited = new boolean[A\_len][B\_len];

PriorityQueue<Pair> heap = new PriorityQueue<Pair>(k, new Comparator<Pair>(){

public int compare(Pair p1, Pair p2){

return p1.sum - p2.sum;

}

});

//initially put the A[0]+B[0] into heap

heap.add(new Pair(0, 0, A[0] + B[0]));

visited[0][0] = true;

//poll k-1 times

for(int i = 1; i <= k - 1; i++){

Pair small = heap.poll();

int x = small.index1;

int y = small.index2;

if(x+1 < A\_len && !visited[x+1][y]){

heap.add(new Pair(x+1, y, A[x+1] + B[y]));

visited[x+1][y] = true;

}

if(y+1 < B\_len && !visited[x][y+1]){

heap.add(new Pair(x, y+1, A[x] + B[y+1]));

visited[x][y+1] = true;

}

}

return heap.poll().sum;

}

}