# 题目

我们有一个由平面上的点组成的列表 points。需要从中找出 K 个距离原点 (0, 0) 最近的点。

（这里，平面上两点之间的距离是欧几里德距离。）

你可以按任何顺序返回答案。除了点坐标的顺序之外，答案确保是唯一的。

**示例 1：**

**输入：**points = [[1,3],[-2,2]], K = 1

**输出：**[[-2,2]]

**解释：**

(1, 3) 和原点之间的距离为 sqrt(10)，

(-2, 2) 和原点之间的距离为 sqrt(8)，

由于 sqrt(8) < sqrt(10)，(-2, 2) 离原点更近。

我们只需要距离原点最近的 K = 1 个点，所以答案就是 [[-2,2]]。

**示例 2：**

**输入：**points = [[3,3],[5,-1],[-2,4]], K = 2

**输出：**[[3,3],[-2,4]]

（答案 [[-2,4],[3,3]] 也会被接受。）

**提示：**

1 <= K <= points.length <= 10000

-10000 < points[i][0] < 10000

-10000 < points[i][1] < 10000

# 分析

## 方法一：堆/优先队列

思路：

代码：

class Solution {

public:

struct cmp {

bool operator () (vector<int> & a, vector<int> & b) {

//不是operator，这是个函数

return pow(long(a[0]),2)+pow(long(a[1]),2) <

pow(long(b[0]),2)+pow(long(b[1]),2);

//该函数作用即如果a的平方和小于b的平方和返回true，否则false

}

}; //结构体后面需要有“;”

vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {

vector<vector<int>> re;

if (points.empty()) return re;

priority\_queue<vector<int>, vector<vector<int>>, cmp> que;

for (int i=0; i<points.size(); i++) {

que.push(points[i]);

//入堆的时候自动根据compare函数确定是否插入

if (que.size()>K) que.pop();

}

while (!que.empty()) {

re.push\_back(que.top());

que.pop();

}

return re;

}

};

另一种写法：

class Solution {

public:

struct compare{

bool operator() (vector<int> &a, vector<int> &b){

return pow(long(a[0]),2)+pow(long(a[1]),2) < pow(long(b[0]),2)+pow(long(b[1]),2);

}

};

public:

vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {

std::priority\_queue<vector<int>,vector<vector<int>>,compare> pri\_queue;

vector<vector<int>> tmpArr;

if(points.empty())

return tmpArr;

int i=0;

for(;i<points.size();i++)

{

pri\_queue.push(points.at(i));

if(i>=K)

{

pri\_queue.pop();

}

}

while(!pri\_queue.empty())

{

tmpArr.push\_back(pri\_queue.top());

pri\_queue.pop();

}

return tmpArr;

}

};