```
/*
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

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/k-closest-points-to-origin
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:
方法一:

- 筛选前K个的问题,首先想到的是优先队列这个数据结构,要筛选出前K小的问题,想到了使用小顶堆,由于要比较的是距离,所以想到了pair这个数据结构.
- 由于欧氏距离与平方和成正相关关系,且$-10000 < points[i][0] < 10000$,所以比较对象使用unsigned long 类型即可,不需要用到浮点运算.
  方法二:
- 维护一个容量只有K的大顶堆,最后将K个数据出堆,然后为了保证顺序$reverse$一下即可.

- Ref(向两篇文章的作者致谢):
  - queue pair
  - priority_queue 中存放pair时，自定义排序的写法

方法一:C++_小顶堆优先队列

```
class Solution
{
    private:
        struct cmp
        {
            template<typename T, typename U>
            bool operator()(T const& left, U const &right)
            {
                /*小顶堆*/
                if (left.second > right.second)
                {
                    return true;
                }
                else
                {
                return false;
                }
            }
        };


    public:
        vector<vector<int>> kClosest(vector<vector<int>>& points, int K)
        {
            priority_queue<pair<int,unsigned long>,vector<pair<int,unsigned long >>,cmp> pq;
            vector<vector<int>>     ret_vvi                 ;
            int                     i           =   0       ;


            for(i=0;i<points.size();i++)
            {
                pq.push(make_pair(i,points[i][0]*points[i][0]+points[i][1]*points[i][1]));
            }
            for(i=0;i<K;i++)
            {
                ret_vvi.push_back(points[pq.top ().first]);
                pq.pop();
            }
            return ret_vvi;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :316 ms, 在所有 cpp 提交中击败了74.78% 的用户
内存消耗 :45 MB, 在所有 cpp 提交中击败了45.59%的用户
*/
```

方法二:C++_大顶堆优先队列

```
class Solution
{
    private:
        struct cmp
        {
            template<typename T, typename U>
            bool operator()(T const& left, U const &right)
            {
                /*大顶堆*/
                if (left.second < right.second)
                {
                    return true;
                }
                else
                {
                return false;
                }
            }
        };
    public:
        vector<vector<int>> kClosest(vector<vector<int>>& points, int K)
        {
            priority_queue<pair<int,unsigned long>,vector<pair<int,unsigned long >>,cmp> pq;
            vector<vector<int>>     ret_vvi                 ;
            int                     i           =   0       ;
            unsigned long           square_sum  =   0       ;

            for(i=0;i<points.size();i++)
            {
                square_sum = points[i][0]*points[i][0]+points[i][1]*points[i][1];
                if(pq.size() < K )
                {
                    pq.push(make_pair(i,square_sum));
                }
                else
                {
                    if(square_sum < pq.top().second)
                    {
                        pq.pop();
                        pq.push(make_pair(i,square_sum));
                    }
                }
            }


            for(i=0;i<K;i++)
            {
                ret_vvi.push_back(points[pq.top ().first]);
                pq.pop();
            }


            reverse(ret_vvi.begin(),ret_vvi.end());
```

```
54
55
56              return ret_vvi;
57          }
58  };
59  /*
60  执行结果:
61  通过
62  显示详情
63  执行用时 :260 ms, 在所有 cpp 提交中击败了95.03% 的用户
64  内存消耗 :42.1 MB, 在所有 cpp 提交中击败了63.61%的用户
65  */
```

AlimyBreak
2019.11.09