

```

1  /*
2  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。
3
4  （这里，平面上两点之间的距离是欧几里德距离。）
5
6  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。
7
8
9
10 示例 1:
11
12 输入: points = [[1,3],[-2,2]], K = 1
13 输出: [[-2,2]]
14 解释:
15  (1, 3) 和原点之间的距离为 sqrt(10),
16  (-2, 2) 和原点之间的距离为 sqrt(8),
17  由于 sqrt(8) < sqrt(10), (-2, 2) 离原点更近。
18  我们只需要距离原点最近的 K = 1 个点，所以答案就是 [[-2,2]]。
19
20 示例 2:
21
22 输入: points = [[3,3],[5,-1],[-2,4]], K = 2
23 输出: [[3,3],[-2,4]]
24  （答案 [[-2,4],[3,3]] 也会被接受。）
25
26
27
28 提示:
29
30 1 <= K <= points.length <= 10000
31 -10000 < points[i][0] < 10000
32 -10000 < points[i][1] < 10000
33
34 来源：力扣（LeetCode）
35 链接：https://leetcode-cn.com/problems/k-closest-points-to-origin
36 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。
37 */

```

分析:

- 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.
- 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++\_优先队列

```

1  class solution
2  {
3      private:
4          struct cmp
5          {
6              template<typename T, typename U>
7              bool operator()(T const& left, U const &right)
8              {
9                  /*小顶堆*/
10                 if (left.second > right.second)
11                 {
12                     return true;
13                 }
14                 else
15                 {
16                     return false;
17                 }
18             }
19         };
20
21
22     public:
23         vector<vector<int>> kClosest(vector<vector<int>>& points, int K)
24         {
25             priority_queue<pair<int,unsigned long>,vector<pair<int,unsigned
26             long >>,cmp> pq;
27             vector<vector<int>> ret_vvi ;
28             int i = 0 ;
29
30             for(i=0;i<points.size();i++)
31             {
32                 pq.push(make_pair(i,points[i][0]*points[i][0]+points[i]
33                 [1]*points[i][1]));
34             }
35             for(i=0;i<K;i++)
36             {
37                 ret_vvi.push_back(points[pq.top ().first]);
38                 pq.pop();
39             }
40             return ret_vvi;
41         };
42
43         /*
44         执行结果:
45         通过
46         显示详情
47         执行用时 :316 ms, 在所有 cpp 提交中击败了74.78% 的用户
48         内存消耗 :45 MB, 在所有 cpp 提交中击败了45.59%的用户
49         */

```