Leetcode May Challenge DAY: 30

1. Python

*When you see in the statement of the problem that you need you find k biggest or k smallest elements, you should immediately think about heaps or sort. Here we need to find the k smallest elements, and hence we need to keep **max** heap. Why max and not min? We always keep in the root of our heap the `k`-th smallest element. Let us go through example: `points = [[1,2],[2,3],[0,1]], [3,4]`, `k = 2`.*

First we put points `[1,2]` and `[2,3]` into our heap. In the root of the heap we have maximum element `[2,3]`

Now, we see new element `[0,1]`, what should we do? We compare it with the root, see, that it is smaller than root, so we need to remove it from our heap and put new element instead, now we have elements `[1,2]` and `[0,1]` in our heap, root is `[1,2]`

Next element is `[3,4]` and it is greater than our root, it means we do not need to do anything.

Complexity

We process elements one by one, there are `n` of them and push it into heap and pop root from our heap, both have `O(log k)` complexity, so finally we have `O(n log k)` complexity, which is faster than `O(n log n)` algorighm using sorting.

Code:

*First, we create heap (which is by definition **min** heap in python, so we use negative distances). Then we visit all the rest elements one by one and update our heap if needed.*

```
class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:
        heap = [[-i*i-j*j, i, j] for i,j in points[:K]]
        rest_elem = [[-i*i-j*j, i, j] for i,j in points[K:]]
        heapq.heapify(heap)
        for s, x, y in rest_elem:
            if s > heap[0][0]:
                heapq.heappush(heap, [s,x,y])
                heapq.heappop(heap)

        return [[x,y] for s,x,y in heap]
```

2. C++

```
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>> &p, int k) {
    multimap<int,int> m;
    for(int i=0; i<p.size();i++)
    m.insert({p[i][0]*p[i][0]+p[i][1]*p[i][1],i});
    vector<vector<int>>v;
    for(auto it=m.begin();k>0;it++,k--)
    v.push_back(p[it->second]);
    return v;
  }
};
```

3. JAVA

Idea

Same problem: 215. Kth Largest Element in an Array

We use `maxHeap` to keep `k` smallest elements in array `n` elements. In the max heap, the top is the max element of the heap and it costs in `O(1)` in time complexity. By using max heap, we can remove `(n-k)` largest elements and keep `k` smallest elements in array.

```
class Solution {
    public int[][] kClosest(int[][] points, int K) {
        PriorityQueue<int[]> maxHeap = new PriorityQueue<>((a, b) -> (b[0] * b[0] +
b[1] * b[1]) - (a[0] * a[0] + a[1] * a[1]));
        for (int[] p : points) {
            maxHeap.offer(p);
            if (maxHeap.size() > K) {
                maxHeap.poll();
            }
        }
        int[][] ans = new int[maxHeap.size()][2];
        int i = 0;
        while (!maxHeap.isEmpty()) {
            ans[i++] = maxHeap.poll();
        }
        return ans;
    }
}
```