一、题目说明

题目215. Kth Largest Element in an Array，在一个无序数组中找第k大的元素。难度是Medium！

二、我的解答

这个题目最直观的解答是，先对数组排序，然后直接返回：

```
class Solution{
    public:
        int findKthLargest(vector<int>& nums,int k){
            sort(nums.begin(),nums.end());

            return nums[nums.size()-k];
        }
};
```

性能如下：

```
Runtime: 8 ms, faster than 97.67% of C++ online submissions for Kth Largest
Element in an Array.
Memory Usage: 9.2 MB, less than 93.94% of C++ online submissions for Kth Largest
Element in an Array.
```

三、优化措施

用小根堆实现，无需多言：

```
class Solution{
    public:
        //queue first in first out with priority delete
        int findKthLargest(vector<int>& nums,int k){
            //升序队列 ，小根堆
            priority_queue<int,vector<int>,greater<int> > q;

            for(auto it: nums){
                q.push(it);
                if(q.size()>k){
                    cout<<q.top()<<"->";
                    q.pop();
                }
            }

            return q.top();
        }
};
```

```
Runtime: 12 ms, faster than 80.01% of C++ online submissions for Kth Largest Element in an Array.
Memory Usage: 9.5 MB, less than 39.39% of C++ online submissions for Kth Largest Element in an Array.
```

上面2个方法都不是最好的办法：方法1胜在简单易实现，方法2直观，方法3利用快速排序的思想：

```
class Solution{
    public:
        //利用快速排序的思想，不断将集合划分为左右两部分，
        //如果划分的位置pivot>k-1，则第k大的数在左边
        //如果划分的位置pivot<k-1，则第k大的数在右边
        int findKthLargest(vector<int>& nums,int k){
            int low = 0,high = nums.size()-1,mid = 0;
            while(low<=high){
                mid = partation(nums,low,high);
                if(mid == k-1){
                    return nums[mid];
                }else if(mid<k-1){
                    low = mid + 1;
                }else{
                    high = mid -1;
                }
            }
            return -1;
        }
        int partation(vector<int>& nums,int low,int high){
            int left = low+1;
            int right = high;
            swap(nums[low],nums[(low+high)/2]);
            int bound = nums[low];
            while(left<=right){
                while(left<high && nums[left] >= bound) left++;
                while(nums[right]<bound) right--;
                if(left<right){
                    swap(nums[left++],nums[right--]);
                }else{
                    break;
                }
            }
            swap(nums[low],nums[right]);
            return right;
        }
};
```