# O(nlgn) time

def findKthLargest1(self, nums, k):

return sorted(nums, reverse=True)[k-1]

# O(nk) time, bubble sort idea, TLE

def findKthLargest2(self, nums, k):

for i in xrange(k):

for j in xrange(len(nums)-i-1):

if nums[j] > nums[j+1]:

# exchange elements, time consuming

nums[j], nums[j+1] = nums[j+1], nums[j]

return nums[len(nums)-k]

# O(nk) time, selection sort idea

def findKthLargest3(self, nums, k):

for i in xrange(len(nums), len(nums)-k, -1):

tmp = 0

for j in xrange(i):

if nums[j] > nums[tmp]:

tmp = j

nums[tmp], nums[i-1] = nums[i-1], nums[tmp]

return nums[len(nums)-k]

# O(k+(n-k)lgk) time, min-heap

def findKthLargest4(self, nums, k):

heap = []

for num in nums:

heapq.heappush(heap, num)

for \_ in xrange(len(nums)-k):

heapq.heappop(heap)

return heapq.heappop(heap)

def findKthLargest(self, nums, k):

heap = nums[:k]

heapify(heap)

for n in nums[k:]:

heappushpop(heap, n)

return heap[0]

# O(k+(n-k)lgk) time, min-heap

def findKthLargest5(self, nums, k):

return heapq.nlargest(k, nums)[-1]

# O(n) time, quick selection

def findKthLargest(self, nums, k):

# convert the kth largest to smallest

return self.findKthSmallest(nums, len(nums)+1-k)

def findKthSmallest(self, nums, k):

if nums:

pos = self.partition(nums, 0, len(nums)-1)

if k > pos+1:

return self.findKthSmallest(nums[pos+1:], k-pos-1)

elif k < pos+1:

return self.findKthSmallest(nums[:pos], k)

else:

return nums[pos]

# choose the right-most element as pivot

def partition(self, nums, l, r):

low = l

while l < r:

if nums[l] < nums[r]:

nums[l], nums[low] = nums[low], nums[l]

low += 1

l += 1

nums[low], nums[r] = nums[r], nums[low]

return low