class KthLargest {

constructor(k, nums) {

this.k = k;

this.minHeap = new MinHeap();

// 初始化堆，保持堆的大小为k

for (const num of nums) {

this.add(num);

}

}

add(val) {

if (this.minHeap.size() < this.k) {

this.minHeap.insert(val);

} else if (val > this.minHeap.peek()) {

this.minHeap.extractMin();

this.minHeap.insert(val);

}

return this.minHeap.peek();

}

}

// 最小堆实现

class MinHeap {

constructor() {

this.heap = [];

}

size() {

return this.heap.length;

}

peek() {

return this.heap[0];

}

insert(val) {

this.heap.push(val);

this.bubbleUp(this.heap.length - 1);

}

extractMin() {

const min = this.heap[0];

const end = this.heap.pop();

if (this.heap.length > 0) {

this.heap[0] = end;

this.bubbleDown(0);

}

return min;

}

bubbleUp(index) {

while (index > 0) {

const parentIndex = Math.floor((index - 1) / 2);

if (this.heap[parentIndex] <= this.heap[index]) break;

[this.heap[parentIndex], this.heap[index]] = [this.heap[index], this.heap[parentIndex]];

index = parentIndex;

}

}

bubbleDown(index) {

const length = this.heap.length;

while (true) {

const leftChildIndex = 2 \* index + 1;

const rightChildIndex = 2 \* index + 2;

let smallestIndex = index;

if (leftChildIndex < length && this.heap[leftChildIndex] < this.heap[smallestIndex]) {

smallestIndex = leftChildIndex;

}

if (rightChildIndex < length && this.heap[rightChildIndex] < this.heap[smallestIndex]) {

smallestIndex = rightChildIndex;

}

if (smallestIndex === index) break;

[this.heap[index], this.heap[smallestIndex]] = [this.heap[smallestIndex], this.heap[index]];

index = smallestIndex;

}

}

}