**Sorting**

**Insertion sort**

插入排序像是扑克整理手上的牌一样，每抽一张牌就把这张牌放到对应的位置，而在转换成code时候，用array实现，放到对应位置即包含array的shift，对应code如下：

//将第j个元素插入到0-（j-1）排好序的元素中

public static void insertionSort (int[] nums) {

for (int j = 1; j < nums.length; j ++) {

int tmp = nums[j];

for (int i = j-1; j >= 0 && nums[i] >tmp; j--) {

nums[i+1] = nums[i]; //shift

}

nums[i+1] = tmp;

}

}

分析：insertion sort在average case时的时间复杂度为O(n2)，对于数量级较小的排序是很好的选择。

**Heap sort**

Heap即堆数据结构，heap sort是指利用heap的数据结构来实现的排序。要理解heap sort首先就要了解heap的属性。Heap分为maxHeap 和minHeap。就maxHeap来讲即除了根节点外其余全部节点满足A[parent(i)] > A[i]。其结构为近似满二叉树，中间不会有空的节点，操作永远自底向上。Heap sort的原理是首先将数组构建成Heap，然后对heap进行delete min/max的操作得到顺序数组。

Build Heap:

//为了满足heap的性质，一般不用index 0来储存元素

public static void buildHeap(nums[i]) {

//一个heap的叶子节点是从num.lenth/2 + 1这个index开始的，所以只要从不是叶子节点的nums.length/2开始一直检查到index为1的节点保证它们都符合heap的性质即可。

for (int i = nums.length/2; i >=1; i--) {

maxHeapify(nums, i);

}

}

//此处的nums.length都为heap的size

public static void maxHeapify (int[] nums, int i) {

int left = 2\*i;

int right = 2\*i + 1;

int largest;

if (left <= nums.length && nums[left] > nums[i]) {

largest = left

} else {

largest = i

}

if (right <= nums.length && nums[right] > largest) {

larget = right

}

if (largest != i) {

int tmp = nums[i];

nums[i] = nums[largest];

nums[largetst] = tmp;

maxHeapify(nums, largetst);

}

}

Heap sort:

//建立了maxHeap后进行delete max操作得到sorted array，即每删除一个元素把它放至队尾。

public static void heapSort(int[] nums) {

for (int i = nums.length; i >= 2; i --) {

int tmp = nums[i];

nums[i] = nums[1];

nums[1] = tmp;

nums.length --;

maxHeapify(nums, 1);

}

}

分析: buildHeap的case为O(n), heapSort的case为O(nlogn)且为upper bound即worst case也为O(nlogn), only requires constant space