aardio 范例: 选择排序

```
import console;

/*------------------------------------------------------
 * 堆排序( 原地排序 )
 *------------------------------------------------------*/

/*
二叉堆是完全二叉树。
索引总是以2倍的形式递增,从上到下从左到右顺序递增

           [1]
           ∧
       [2]     [3]
       ∧       ∧
    [4] [5]   [6] [7]

因此可以用数组来表示二叉堆。

设 heap= {}
heap[1] = 29;
heap[2] = 19;
heap[3] = 18;
heap[4] = 9;
heap[5] = 8;
heap[6] = 7;
heap[7] = 6;

则 heap如下排列 {29;19;18;9;8;7;6}

           29
           ∧
       19       18
       ∧        ∧
     9    9   7    6

二叉堆有小根堆,大根堆,大根堆最大的元素在顶部,所有元素按索引排序。
*/

var getParentIndex = function(i){
    //二叉堆的父子索引总是以2倍的形式递增、堆左边索引是2的n次方。
    return math.floor(i/2)//所以除2总是能向上倒退一层到父节点的索引。
}
var getLeftNodeIndex = function(i){
    //父层到子层,总是一变二,所以索引也增加2倍
    return 2*i
}
var getRightNodeIndex = function(i){
    return 2*i+1
}

/*
假定一个数组A的元素i,假定getLeftNodeIndex[i],getRightNodeIndex[i]都是最大堆。
如果array[i]小于其子女,则调节array[i],使之下降。
*/
var maxHeapify;
maxHeapify = function(array,i){
    var l = getLeftNodeIndex(i);
    var r = getRightNodeIndex(i);

    var largest;
    if( ( l<=array.heapsize ) and( array[l] > array[i]) )
        largest = l;//左子树比较大
    else{
        largest = i;//父节点比较大
    }

    if( ( r<=array.heapsize ) and( array[r] > array[largest]) )
        largest = r;//右子树比较大

    //如果array[i]小于其子女,则调节 array[i],使之下降。
    if( largest != i){
        var exchange = array[i]
        array[i] = array[largest]
        array[largest] = exchange;
```

```
        maxHeapify(array,largest);//现在子树 largest 是最大的了
    }
}

/*
对于数组 A,最后一个页节点索引为 #A
#A 的父节点为 math.floor(#A/2) ,而且是最后一个父节点。而之后都是叶节点。

因此,我们从最后一个父节点开始,从右至左,自下而上调用 maxHeapify 检查所有的堆。使之符合二叉堆定义.
如此：小的通过 maxHeapify 函数的递归调用一降到底,大的数通过 buildMaxHeap 的 for 循环逐步上升到顶,清升而浊降。
*/

//建堆
var buildMaxHeap = function(array){
    array.heapsize = #array;
    for(i=math.floor(#array/2);1;-1){
        maxHeapify(array,i);
    }
}

/*
堆建好了,根节点一定比子结点大。
但是左边的不一定比右边的大,因为二叉堆不比较左右子树。
而且在不同的子二叉堆里,他们也没有排序关系
console.dump( array ) //输出看一下,整体上仍然是乱的
*/

/*
堆虽然不是完全线性有序的数组,但是根节点总是最大的元素。
通常用于实现 高效的优先级队列
*/

//那么下面我们要利用建好的堆来排序
var heapSort = function(array){
    buildMaxHeap(array)//建堆

    //现在array[1]是根节点了,并且一定是最大的
    //我们把最大的放在数组最右侧,后把最右侧较小的元素放到根节点来破坏二叉堆,再用maxHeapify来修复堆取到下一个最大的数
    //不断循环此过程,不断的将最大的元素移到右侧,而堆向左收缩,就达到了排序的效果
    for(sorted=#array;2;-1){
        var max = array[1]
        array[1] = array[sorted] //把小的移到二叉堆的顶部来破坏堆定义
        array[sorted] = max;//把最大的移到右侧已排序的数组中

        array.heapsize--;
        maxHeapify(array,1);//修复二叉堆根节点,取下一个最大数

    }

}

//终于排序好了输出看一下

console.log("----------------")
console.log("堆排序")
console.log("----------------")

array ={2;46;5;17;1;2;3;99;12;56;66;21};
heapSort(array)

//输出结果
for(i=1;#array;1){
    console.log( array[i] )
}

/*
排序算法的稳定性：保证排序前2个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同

稳定排序算法：冒泡排序 插入排序 合并排序
不稳定排序算法：堆排序 快速排序
*/

execute("pause")
```

Markdown 格式