最大最小数问题

给定一个有n个整数的数组，设计一个算法求最大最小数。

算法描述 1

1. 定义两个变量min和max分别来存储最小和最大值
2. 将第一个元素赋给min和max
3. 从第二个元素开始遍历整个数组，并用当前元素分别去比较min和max
    1. 若当前元素的值大于max则把max用当前这个元素来替换
    2. 若当前元素的值小于min的值则把min用当前元素来替换
4. 遍历结束后，min和max的值就是这个数组的最小和最大值

算法描述 2

```
getMinAndMax(array)
{
    //将第一个元素赋予min和max
    min = max = array[0]

    //从第二个元素(index = 1)开始遍历
    foreach(item in array start from index 1)
    {
        if( item > max)
        {
            max = item;
        }
        else if( item < min)
        {
            min = item;
        }
        else
        {
            //Nothing to do.
        }
    }
    return min, max
}
```

算法描述 3

使用javascript语言来实现这个算法，只需要做简单的修改以符合其语法规则即可。

```
function getMinAndMax(array)
{
    //将第一个元素赋予min和max
    var min = array[0];
    var max =  array[0];

    //从第二个元素(index = 1)开始遍历
    for(var i=1; i < array.length;i++)
    {
        var item = array[i];
        if( item > max)
        {
            max = item;
        }
        else if( item < min)
        {
            min = item;
        }
        else
        {
            //Nothing to do.
        }
    }
    return [min, max];
}
```

复杂度分析

根据上面的算法描述可以看出：该算法中主要的操作是比比较操作，在for循环中没次循环更有至多两次比较和最多一次赋值，也就是最多有3次基本操作，一共循环了n-1次。所以T(n) = (n-1)*3 = O(n)

使用递归方法的实现

算法描述 1

1. 将n个元素的数组s分成两个子数组s1 和 s2.
2. 分别对s1和s2求最大，最小值。

3. 将s1和s2各自的最大值的大者作为最终的最大值， 将s1和s2各自的最小值的小者作为最终的最小值，

算法描述 2

getMinMax(s)
{
//当只有一个元素时，最大最小值都是自身
if(length(s) == 1)
{
return [s[0],s[0]]
}
middleIndex = $\lceil length(s)/2 \rceil$
s1 = s[0:middleIndex]
s2 = s[middleIndex+1:length(s) -1]
[s1_min, s1_max] = getMinMax(s1)
[s2_min, s2_max] = getMinMax(s2)

```
return [min(s1_min,s2_min),max(s1_max,s2_max)]
```

}

算法描述 3

```
function getMinMax(s)
{
    if(s.length == 1)
    {
        return [s[0],s[0]];
    }
    var middleIndex = s.length / 2;
    var s1 = s.slice(0,middleIndex);
    var s2 = s.slice(middleIndex);
    var s1_min_max = getMinMax(s1);
    var s2_min_max = getMinMax(s2);

    return [Math.min(s1_min_max[0],s2_min_max[0]),Math.max(s1_min_max[1],s2_min_max[1])];
}
```

复杂度分析

由上述算法描述可以发现， 对于长度为n的数组每次分解成长度为 $\lceil n/2 \rceil$ 和 $n - \lceil n/2 \rceil$ 的子数组。因为

$n - \lceil n/2 \rceil$ <=$\lceil n/2 \rceil$. 取其大者，不妨设每次都分解成两个长度为$\lceil n/2 \rceil$ 的子数组,每次递归还有至多3次比较操作，一次比较长度是否为1和两次比较子数组的返回值。
可以得出

$$T(n) = 2 * T(\lceil n/2 \rceil) + 3(n > 1)$$

$$T(1) = 1$$

展开公式可以得到
$T(n) = 2 * T(\lceil n/2 \rceil) + 3$

$= 2^2 T(\lceil n/4 \rceil) + 2 * 3 + 3$

$= 2^3 T(\lceil n/8 \rceil) + 2^2 * 3 + 2 * 3 + 3$

= ...
$= 2^{\lceil log(n) \rceil} * T(1) + 2^{\lceil log(n) \rceil - 1} * 3 + 2^{\lceil log(n) \rceil - 2} * 3 + \ldots + 3$
$= 2^{\lceil log(n) \rceil} + 2^{\lceil log(n) \rceil - 1} * 3 + 2^{\lceil log(n) \rceil - 2} * 3 + \ldots + 3$
$<= 2^{\lceil log(n) \rceil} * 3 + 2^{\lceil log(n) \rceil - 1} * 3 + 2^{\lceil log(n) \rceil - 2} * 3 + \ldots + 3$
$= 3 * \frac{1 * (1 - 2^{\lceil log(n) \rceil + 1})}{1 - 2}$
$= 3 * (2^{\lceil log(n) \rceil + 1} - 1)$
=O(n)