2-10：

> *若在习题2-9 中，数组T中元素不存在序关系，只能测试任意两个元素是否相等，试设计一个有效算法确定T 是否有一主元素。算法的计算复杂性应为$O(nlogn)$。更进一步，能找到一个线性时间算法吗？*

题目分析：

本题与2-9的区别在于，数组 T 中不存在序关系。因此，没办法利用2-9的思路，因为不再能线性时间选出中位数。因此，本题需要用别的方法求解。

对于本题，必有如下结论：

- 如果 x 为 $T[0:n-1]$ 中的主元素，则 x 必为 $T_1[0:\frac{n-1}{2}]$ 或 $T_2[\frac{n-1}{2}:n-1]$ 中的主元素，如不然，使用反证法：若 x 不是 T_1 也不是 T_2 的主元素，即 $|S_1(x)| \leq \frac{n}{4}\&\&|S_2(x)| \leq \frac{n}{4}$，则对于整个数组 T，有 $|S(x)| \leq \frac{n}{2}$，即 x 不是 T 的主元素。

既然只能比较任意两个元素是否相等，采用分治法，对原数组 T 进行左右对半划分，依次循环，直到数组大小为 1。对于该“子数组”而言，显然剩下的那个元素为其主元素。之后再两两合并数组，对两个数组中的“主元素”都进行一次遍历，判断是否为合并后数组的主元素。如果是，则将该子元素继续向上传递；否则，返回一个无穷大表示该数组不存在主元素。

代码实现：

在本代码中，仍然使用 int 数组来模拟数组 T，但只比较大小，而不利用线性数组的序关系。可将 T 更换为其他类型，譬如结构、类等。

代码具体实现见2-10 Code

输出示例：

```
输入样例1:
16
3 9 8 6 11 21 49 38 5 4 7 1 2 15 23 8
输出1：不存在
输入样例2:
16
2 3 5 2 1 8 2 3 2 2 2 2 5 2 3 2
输出2：存在，且主元素为2（#因为有9个2）
输入样例3:
7
3 1 2 3 3 7 3
输出3：存在，且主元素为3（#因为有4个3）
输入样例4:
7
1 7 3 2 3 3 1
```

```
输出4：不存在（#因为只有3个3）
```

算法分析：

该程序显然是一个递归式，其复杂度为：

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

由主定理，$f(n) = O(n) = \Theta(n^{log_2 2})$可得：$T(n) = O(nlogn)$

更进一步：

能否找到线性时间的算法呢？

用递归的方法，算法下界应为$T(n) = O(nlogn)$（不会证明，但较为显然）。

思考一下，2-9为什么能在$O(n)$时间内完成？因为元素有序，也就是说，可以在$O(n)$时间内将数组的中位数选择出来。而本题由于元素不存在序关系，既没有办法排序，也没有办法选择。

对于这个问题，我们可以将之分解为两步：

- 寻找“主元素”的备选者。
- 验证备选者为“主元素”（$O(n)$）

因此，我们只需要能在$O(n)$的时间内找出候选者，就可以在线性时间内解决此题。

由于主元素至少占了数组的一半大小，因此我们可以用一种“抵消”的方式进行筛选：

- Step0：用一指针指向数组0位置。
- Step1：选取指针所在位置的元素为“候选者”。
- Step2：将指针后挪一位，若元素与“候选者”相同，则一计数cnt++；若不同，则cnt--。若cnt已为0，则回到Step1，且cnt重置为1。
- Step3：指针遍历完数组后，若没有“候选者”（即cnt = 0），则必定没有主元素（易证前面是一对一对消除的，总共有偶数个元素，而最多的元素个数不超过$\frac{n}{2}$）；若有“候选者”，则检验该“候选者”是否为主元素。

通过这个过程，相当于遍历了两次数组，时间复杂度为$O(n)$。