DSA Homework2

袁逸聪 19302010020

2.26

How does the recursion terminate?

本次递归只剩下一个元素时，它是主元素候选者，不再递归(检测它是否是主元素，是则返回它，否则返回否定信息)

本次递归不剩下任何元素时，说明没有候选者(即没有主元素，返回否定信息)

How is the case where N is odd handled?

N是偶数时，可以正常处理。找到主元素或是认为没有主元素，但多出的一项将改变结果。

如121，对12的处理将认为没有主元素，而最后一位使得1成为主元素。故前N-1项没有主元素时，需要检测最后一项。

所以，对偶数项正常递归即可，对奇数项，在对前N-1项递归找不到候选者后，额外检测最后一项即可。

What is running time of the algorithm?

选出候选者时总需要O(n)去检测

而挑选候选者的过程每次递归近似为O(n),最坏的情况下每次递归问题集规模减半

f(n)<=n+n/2+...+1+n=3n

O(3n)=O(n)

How can we avoid using an extra array B?

将A的前B.length项作为B的空间即可

为使A中原本的信息不损失(以便检测候选者是否合格),将想要放入B的元素和A在此原本位置上的元素互换即可

Write a program to compute the majority element.

```
public class Majority {
    public static void main(String[] args) {
        int[] array = {1,1,2,3,2};
        int result = findMajority(array,array.length - 1);
        if (result == -666) System.out.println("数组中没有主元素");
        else System.out.println("数组的主元素是：" + result);
    }

    //递归函数，输入数组打印主元素(出现次数超过数组元素一半的元素)
    public static int findMajority(int[] array,int index) {
```

```
        //终止条件
        //只剩一个元素时，检测其是否为主元素
        if (index == 0) {
            if (IsMajority(array,array[0])) return array[0];
            else return -666;
        }
        //不剩下元素时，说明没有主元素
        if (index == -1) return -666;

        /*
        为了避免开辟数组B的空间，将A的前n(两两比较的相等次数)项作为B的代替，
        将原本的元素和需要放入的元素交换来保证A数组内容信息不损失(以便检查是否是主元)
         */
        int currentIndex = 0;
        int result = -666;
        for (int i = 0; i <= index; i += 2) {
            //还没到末两位，进行pk
            if (i + 1 < index) {
                if (array[i] == array[i + 1]) {
                    swap(array,i,currentIndex);
                    currentIndex++;
                }
            }
            //末两位已经到结尾了，递归
            if (i + 1 == index) return findMajority(array,currentIndex);
            //i本身就到结尾了，奇数项
            if (i == index) {
                if (IsMajority(array,array[i])) return array[i];
                else return -666;
            }
        }
        return result;
    }

    //检测某元素是否为主元素,O(n)
    public static boolean IsMajority(int[] array,int candidate) {
        int count = 0;
        for (int value : array) {
            if (candidate == value) count++;
        }
        return count > array.length / 2;
    }

    public static void swap(int[] array,int p,int q) {
        int temp = array[p];
        array[p] = array[q];
        array[q] = temp;
    }
}
```

2.32

二分比较中减少了等于的判断，也就不会因为好运而提前结束，不过同样在O(logN)范围内

另外，两边的不对称是的mid迭代方式尤为重要，在代码注释中说明

```
public class BinarySearch {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        System.out.println(binarySearch(array, 5));
    }

    //用int来简化书上的泛型，返回值为对象在数组中的index，找不到返回-1
    public static int binarySearch(int[] array, int element) {
        int low = 0, high = array.length - 1, mid;
        while (low < high) {
            /*迭代公式很关键！因为整除的结果是向下取整，导致low和high差1时，mid为low
            如果以element<array[mid]为判准，则low high差1, mid等于答案的情况下会不更新
low
            在不允许三分判断(如果额外判断是否跟目标相等，则变相地使用了三分判断),就是该
成向上取整比较合适了*/
            /*mid = (int) Math.ceil((low + high) / 2.0);
            if (element < array[mid]) {
                high = mid - 1;
            } else low = mid;*/
            //↑这里是element<array[mid]的实现
            mid = (low + high) / 2;
            if (element > array[mid]) {
                low = mid + 1;
            } else high = mid;
        }
        mid = (low + high) / 2;
        if (element == array[mid]) return mid;
        else return -1;
    }
}
```