# Algorithms Course Project 1: Finding The Majority Element

**17069130027 吕德锐**

1. **Problem Description**

Let A[1..n] be a sequence of integers. An integer in is called the *majority* if it appears more than times in . For example, 3 is the majority in the sequence 1, 3, 2, 3, 3, 4, 3, since it appears four times and the number of elements is seven. There are several ways to solve this problem. However, the original ways are either too inefficient or fairly complex. How to design a efficiency algorithm is the another problem.

1. **Algorithm Description**

There is an elegant solution that uses much fewer comparisons. We derive this algorithm using induction. The essence of the algorithm is based on the following observation:

If two *different* elements in the original sequence are removed, then the majority in the original sequence remains the majority in the new sequence.

This observation suggests the following procedure for finding an element that is a *candidate* for being the majority. Set a counter to zero and let . Starting from , scan the elements one by one increasing the counter by one if the current element is equal to and decreasing the counter by one if the current element is not equal to. If all the elements have been scanned and the counter is greater than zero, then return as the candidate. If the counter becomes zero when comparing with , , then call procedure candidate recursively on the elements . Notice that decrementing the counter implements the idea of throwing two different elements as stated in observation above. This method is described more precisely in Algorithm MAJORITY.

We analyze the time and space complexity of the Algorithm MAJORITY as follow: the algorithm only needs to traverse the array once to find the majority elements, that is, to compare the elements in the array with . Then the time complexity of the algorithm is . Thanks to there is no additional space except the array space, the space complexity of the algorithm is .

**Algorithm** MAJORITY

**Input:** An array A[1..n] of n elements;

**Output:** The majority element if it exists; otherwise none;

1. x← candidate(1)

2. count← 0

3. **for** j ← 1 to n

4. **if** A[j]=x **then** count← count+1

5. **end for**

6. **if** count＞⎣n/2⎦ **then return** c

7. **else return** none

**Procedure** candidate(m)

1. j ← m; x ← A[m]; count← 1

2. **while** j < n **and** count > 0

3. j ←j+1

4. **if** A[j] = c **then** count← count+1

5. **else** count ←count-1

6. **end while**

7. **if** j = n **then return** c

8. **else return** candidate(j + 1)

1. **Source Code (C)**

#include<stdio.h>

#define n 7

int MAJORITY();

int candidate(int m);

int A[n] = {1,3,2,3,3,4,3};

int main()

{

int major = MAJORITY();

if (major) printf("The majority element of the array is: %d",major);

else printf("There is no majority elemrnt in the array.");

return 0;

}

int MAJORITY()

{

int c = candidate(1),count = 0,j;

for(j = 0;j < n;j ++)

if (A[j] == c) count ++;

if (count > n / 2) return c;

else return 0;

}

int candidate(int m)

{

int j = m,c = A[m - 1],count = 1;

while (j < n && count > 0)

{

j ++;

if (A[j - 1] == c) count ++;

else count --;

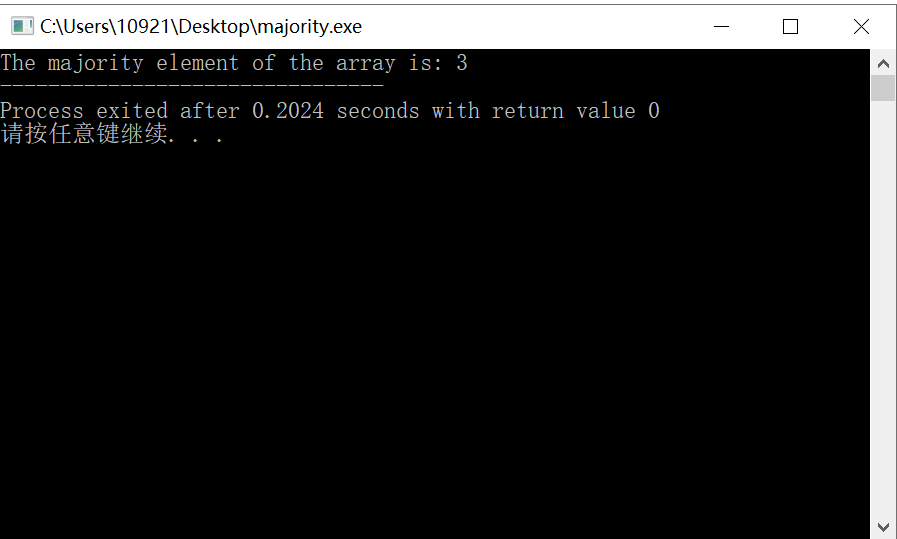
}

if (j == n) return c;

else return candidate(j + 1);

}

1. **Example Analysis**
   1. If A[n] = {1,3,2,3,3,4,3}, 3 is the majority element. The output is:



* 1. If A[n] = {1,3,2,3,2,4,3}, there is no majority element. The output is:

