2 Majority Element

Problem Introduction

Majority rule is a decision rule that selects the alternative which has a majority, that is, more than half the votes.
Given a sequence of elements $a_1, a_2, \ldots, a_n$, you would like to check whether it contains an element that appears more than $n/2$ times. A naive way to do this is the following.

```
MajorityElement(a_1, a_2, ..., a_n):
for i from 1 to n:
  currentElement ← a_i
  count ← 0
  for j from 1 to n:
    if a_j = currentElement:
      count ← count + 1
  if count > n/2:
    return a_i
return "no majority element"
```

The running time of this algorithm is quadratic. Your goal is to use the divide-and-conquer technique to design an $O(n \log n)$ algorithm.

Problem Description

Task. The goal in this code problem is to check whether an input sequence contains a majority element.

Input Format. The first line contains an integer n, the next one contains a sequence of n non-negative integers $a_0, a_1, \ldots, a_{n-1}$.

Constraints. $1 \leq n \leq 10^5$; $0 \leq a_i \leq 10^9$ for all $0 \leq i < n$.

Output Format. Output 1 if the sequence contains an element that appears strictly more than $n/2$ times, and 0 otherwise.

Sample 1.

Input:

```
5
2 3 9 2 2
```

Output:

```
1
```

2 is the majority element.

Sample 2.

Input:

```
4
1 2 3 4
```

Output:

```
0
```

There is no majority element in this sequence.

Sample 3.

Input:

```
4
1 2 3 1
```

Output:

```
0
```

This sequence also does not have a majority element (note that the element 1 appears twice and hence is not a majority element).

What To Do

As you might have already guessed, this problem can be solved by the divide-and-conquer algorithm in time $O(n \log n)$. Indeed, if a sequence of length n contains a majority element, then the same element is also a majority element for one of its halves. Thus, to solve this problem you first split a given sequence into halves and make two recursive calls. Do you see how to combine the results of two recursive calls?

It is interesting to note that this problem can also be solved in $O(n)$ time by a more advanced (non-divide and conquer) algorithm that just scans the given sequence twice.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.