Solutions to Homework 4

Debasish Das
EECS Department, Northwestern University
ddas@northwestern.edu

1 Problem 2.23

Definition 1 *Majority element of an array A[1 ... n]: An array is said to have a majority element if more than half of its entries are the same.*

A constraint on this problem is that only equality is defined on the objects of the array. You can check if the array elements are equal but there is no $>$ or $<$ relation defined on the elements
(a)We split the array A into 2 subarrays A_1 and A_2 of half the size. We choose the majority element of A_1 and A_2. After that we do a linear time equality operation to decide whether it is possible to find a majority element. The recurrence therefore is given by

$$T(n) = 2T(\frac{n}{2}) + O(n) \tag{1}$$

The complexity of algorithm comes to $O(n \log n)$

```
procedure GetMajorityElement(a[1...n])
Input:  Array a of objects
Output:  Majority element of a
if n = 1:  return a[1]
k = ⌊n/2⌋
elem_lsub = GetMajorityElement(a[1...k])
elem_rsub = GetMajorityElement(a[k+1...n]
if elem_lsub = elem_rsub:
  return elem_lsub
lcount = GetFrequency(a[1...n],elem_lsub)
rcount = GetFrequency(a[1...n],elem_rsub)
if lcount > k+1:
  return elem_lsub
else if rcount > k+1:
  return elem_rsub
else return NO-MAJORITY-ELEMENT
```

GetFrequency computes the number of times an element ($elem_{lsub}$ or $elem_{rsub}$) appears in the given array a[1...n]. Two calls to GetFrequency is O(n). After that comparisons are done to validate the existence of majority element. GetFrequency is the linear time equality operation.
(b)Using the proposed divide-and-conquer operation, indeed it is possible to give a linear time algorithm. Idea is to pair up the elements arbitrarily to get $\frac{n}{2}$ pairs. In each pair if the two elements are different we discard both of them. If they are same only one of them is kept. Before we give the algorithm, we have to prove the following lemma

Lemma 1 *After the proposed procedure, there are atmost $\frac{n}{2}$ elements left and if A has a majority element, then remaining elements will have the same majority element*

Proof: Let m be a majority element of A. Frequency of m is greater than $\frac{n}{2}$ where n is the number of elements of A. Since we are forming pairs of 2, there has to be at least one pair where both the elements of the pair are m since otherwise frequency of m in A cannot be greater than $\frac{n}{2}$. At most all pairs can have both the elements of the pair as m. So after the procedure at least one element m will be left or at most $\frac{n}{2}$ will be left where each of them is m. Therefore out of at most $\frac{n}{2}$ elements, one of the element must be m.

Consider arbitrary pairs (p,q) formed by the procedure. There are four possible cases

$$p = q \tag{2}$$
$$p \neq q \tag{3}$$
$$p = m \quad \wedge \quad q \neq m \tag{4}$$
$$p \neq m \quad \wedge \quad q = m \tag{5}$$

All pairs (p,q) satisfy one of the 4 equations. For Equations 3-5 majority of m is maintained because while removing one occurence of majority element m we also remove another element. For pairs satisfying Equation 2 we keep p. p may or may not be equal to m but keeping one occurence of p guarantees majority of m. □

Note that the lemma holds only if the array A has a majority element m. Therefore once we apply the algorithm and come up with a majority element, it is mandatory to check if m is indeed a majority element of the array. That can be done by a GetFrequency call followed by a check whether the frequency of m in A[1...n] is greater than $\frac{n}{2}$.

```
procedure GetMajorityElementLinear(a[1...n])
Input:  Array a of objects
Output:  Majority element of a
if n = 2:
  if a[1] = a[2] return a[1]
  else return NO-MAJORITY-ELEMENT
Create a temporary array temp
for i = 1 to n:
  if a[i] = a[i+1]:
    Insert a[i] into temp
  i = i+1
return GetMajorityElementLinear(temp)


procedure CheckSanity(a[1...n])
Input:  Array a of objects
Output:  Majority element of a
m = GetMajorityElementLinear(a[1...n])
freq = GetFrequency(a[1...n],m)
if freq > ⌊n/2⌋ + 1:
  return m
else return NO-MAJORITY-ELEMENT
```

Recurrence relation for the algorithm is given as follows

$$T(n) = T(\frac{n}{2}) + O(n) \tag{6}$$

as we can see the processing of array in the recursive function is done in O(n) time. Complexity of the function GetMajorityElementLinear is O(n). CheckSanity is also O(n). Therefore the whole algorithm is linear in terms of n.