CS 157 Problem Set 2 SeungJi Lee

Problem 3

a)

① [X, X, X, X, X, Y]

⇓

[X, X,]

⇓

X

majority element : X.

② [X, Y, X, Y, X, Y]. majority element : none.

⇓

∅

③ [X, Y, X, Y, Y] majority element : Y

∅ ∅

⇓

Y

④ [Y, Y, X] majority element : Y.

⇓

Y

On odd case, simply compare ~~the~~ one element to all others of the current level to determine if the element is the majority element.

If so, return the element.

otherwise, simply discard the element.

b) Let A be the array of size n in the consideration.

Let downsize be the operation defined by a).

(i.e. ① pair up the elements of A to get $\frac{n}{2}$ pairs,

② look at each pair ;

if same elements, keep one.

if different, discard both.

③ if n = odd, the compare one element, say e, to all other elements in the given array to find whether the element e is the majority.

if so, simply return {e}

else, discard e, then compute ①, ②.)

Then the algorithm is.

1. if n is 0, return "no-majority-element".
2. if n is 1, return the only element.
3. B = "downsize"(A)
4. recursively call the algorithm with input B (1–4).
5. After finding k, majority element found by (1–4), go through A to make sure k is the majority element.

~~Here's~~ Please see next page for the implementation.

Note that the indices of an array starts from 1.

- Implementation -

downsize (input ; array A of size n, output ; an array of size $\lfloor \frac{n}{2} \rfloor$ or 1)

```
if (n ≤ 1)
    return A.
if (n is odd)
    count ← 1.
    for i ← 2 to n.
        if ( A[1] == A[i] )
            count ++
    if (count ≥ ⌊n/2⌋ + 1)
        return { A[1] }
    else
        A = A[2 ... n].
B ← ∅
for i ← 1, 3, 5, ..., n-1.
    if (A[i] == A[i+1])
        add A[i] to B.
return B
```

---

find_ME_helper (input ; array A of size n, output ; the majority element or ∅) :

```
if (n == 0)
    return ∅
if (n == 1)
    return A[1].
B ← downsize (A)
return find_ME_helper (B)
```

find_majority_element (input: an Array A of size n, output: a majority element or $\emptyset$):

```
K <-- find_ME_helper.
count <- 0.
for i <- 1 to n.
    if (K == A[i]).
        count ++
if (count ≥ ⌊n/2⌋ + 1)
    return K
else
    return ∅.
```

---

c) Let A be the array in the consideration and n be its size.

Let m be the majority element of A.

Let downsize be the operation explained in a).

- Claim: if $\exists$ m in A, then m is still the majority element in the downsized A.

proof:

if n is odd, we perform two operations

① if A[1] is the majority element, we are done, since the majority element of {A[1]} is A[1].

② if A[1] is NOT the majority element, we remove it. Removing non-majority element does not change the majority element.

if n is even, for each pair we either keep one or discard both elements.

Now, pairs that does not contain the majority element does not effect the majority element of the new array since the operations does not increase the elements.

Thus only consider the pairs that contain the majority element.

If discarding both elements,

then exactly one element is m and the other is a non-majority element.

Then, since $num(m) \geq \lfloor \frac{n}{2} \rfloor + 1$

$$num(m) - 1 \geq \lfloor \frac{n-2}{2} \rfloor + 1,$$

discarding both elements does not change m.

If keeping one element

then both elements are m.

Since n is even, and $num(m) \geq \frac{n}{2} + 1$,

$$num(\text{non-majority elements}) \leq \frac{n}{2} - 1,$$

$$num(m) \geq num(\text{non-majority elements}) + 2.$$

Then, subtracting one majority elements still does not change the majority element.

Note that throughout the proof, I assumed that $\exists$ only one majority element. This is trivially true.

Indeed, if $\exists$ two majority elements m_1, m_2, then,

$$num(m_1) \geq \lfloor \frac{n}{2} \rfloor + 1$$

$$num(m_2) \geq \lfloor \frac{n}{2} \rfloor + 1$$

$$num(m_1) + num(m_2) \geq 2 \cdot \lfloor \frac{n}{2} \rfloor + 2 > n.$$

Contradiction!

- Now, if you look at the algorithm, we simply downsize A repeatedly until n become 1 or 0.

Since downsizing keeps the majority element invariant (if $\exists$ one),

- if n becomes 0, $\therefore \exists$ no majority element,
- if n becomes 1 and if majority element exists, then the output is the majority element.

d)

<u>downsize is $O(n)$.</u>

- inside if(n is odd)
  the for loop has $n-1$ iterations of constant operations.
- the for loop has $\frac{n}{2}$ iterations of constant operations.

<u>find_ME-helper</u> is $O(n)$.

$$T(n) = \underbrace{O(1)}_{\text{if-then Statements}} + \underbrace{O(n)}_{\text{downsize}} + \underbrace{T\left(\frac{n}{2}\right)}_{\text{recursive call.}}$$

$$T(n) = O(1) \quad \text{if } n \leq 1$$

Then,

$$\begin{aligned}
T(n) &= O(n) + T\left(\frac{n}{2}\right) \\
&= O(n) + O\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) \\
&= O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) + \cdots + O(1). \\
&= O\left(n\left(1+\frac{1}{2}+\cdots+\frac{1}{n}\right)\right) \\
&= O\left(n\left(1+\frac{1}{2}+\cdots\right)\right) \\
&\doteq O\left(n\left(\frac{1}{1-\frac{1}{2}}\right)\right) = O(2n) = O(n)
\end{aligned}$$

<u>find_majority_element</u> is $O(n)$

- find_ME_helper is $O(n)$
- for loop has n iterations of constant operations.