Finding k-majority element

Problem: Given an array A storing n elements and a parameter k which is a positive integer, an element of A is said to be k-majority of A if it appears more than n/k times in A. Design an algorithm which computes a k-majority element, if exists, in array A. The algorithm must have O(nk) time complexity. O(k) extra space can be used.

Solution:

Let p be a k majority element in A. we state the following lemma that establishes the presence of p as a k-majority element in A after deleting k different elements from it.

Lemma 1. *If p is a k-majority element in an array of size n; then it is a k-majority element in an array of size $n - k$ obtained after neglecting k different elements from the original array.*

Proof. Case1: (p is among the neglected k 'different' elements)
Let *count* be the number of times p appears in original array, clearly *count* $> n/k$ Now *count* decresases by 1 after neglecting k elements. and number of elements in new array is *n-k*. So p appears greater than n/k -1 = (*n-k*)/(*k*) times in an array of size *n-k*. Hence it still a k-majority element in new array.

Case2: (p is not among the neglected k 'different' elements)
Here, after neglecting k different elements, count> n/k and number of elements in new array are (*n-k*). As, n/k is greater than (*n-k*)/k; so p is a k-majority element of the new array. □

Proving correctness of the algorithm:
NOTATIONS:
a) 'Corresponding count' of an element $B[i]$ is the value $C[i]$
b)B is said to be full when count corresponding to each of its element in C is non-zero.
c)First empty location in B is the minimum i for which $C[i]$=0
d) Corresponding count of $B[i]$ is $C[i]$
e) 'Remembered elements' of A are those which have not yet been scanned or are present as some element in array B.
We need to prove that Algorithm 1 is correct.Note that if a k-majority element does not exist, our algorithm outputs nothing.
Further, It follows from the discussion preceding Lemma 1 that it suffices if we can show that Algorithm 1 indeed neglect sets of k distinct elements[if they exist] traversing from 0th element to end of array A.
If a k-majority element exists, its presence is guaranteed in final array B which is obtained by exhausting elements of array A by lemma 1 and since we are checking the number of occurences of all elements of B in the end, we can identify the k-majority element if it exists.

Lemma 2. *Whenever array B is full, it neglects k different elements of array A and it does this until the size of 'remembered elements' of A becomes less than or equal to k.*

Proof. Each element of array B stores distinct values from A at any instant and C stores their corresponding count.
We keep on filling B untill its full and then neglect all current entries of B which are k in number.This is signified by decreasing their count by one.
Note, this is as if those k entries never existed in array A.
Hence, after each instant of array B being full, k distinct entries of A are forgotten.

```
Data: Input will be an array A[0.....n − 1] and the number k.The algorithm will use two
      additional arrays B[0.....k-1] and C[0.....k-1] each of size k such that B[i] will store
      some element of A and c[i] will store the corresponding count of that element in a
      certain 'sense' which gets clear as you read the algorithm.
Result: The algorithm outputs a k-majority element of A if it exists.
Initialise all elements of C to be equal to 0.;
Initialise all elements of B to a value unexpected/not present in A
for i ← 0 to n − 1 do
    if A[i] is present in B then
        Let B[t] = A[i];
        C[t] ← C[t] + 1;
    else
        find the first empty location of B, say e. B[e] ← A[i];
        C[e] ← 1;
    end
    if array B is full then
        decrease the count of each element of C by 1;
    end
end
for each element in B whose corresponding count > 0 do
    scan array A to check its number of occurences;
    If number of occurences of a certain number say B[x] > n/k, output B[x].break;
end
```

Algorithm 1: Algorithm to find k-majority element in an array

This aspect, along with the fact that repetition of an element already present in B adds no new entry in B ensures that after exhausting all elements of A, only k or less distinct elements remain behind as entries of B. Since maximum number of neglects are less than or equal to floor function applied to n/k (which is further strictly less than occurences of a k-majority element if it exists); hence such an element is bound to be present in the final array B. □

Analysis of Time and Space complexity the Algorithm

Algorithm 1 executes "for" loop $n-1$ times, and in each iteration it spends $O(k)$ time. Thereafter, it spends a total of $O(nk)$ time in the "for" loop. Thereafter, it scans 'non-zero count' entries of array B and finds their number of occurences in array A.This also takes $O(nk)$ time. So overall time complexity of the algorithm is $O(nk)$. The algorithm uses two additional arrays B and C each of size k in addition to a few variables. Hence, the algorithm uses $O(k)$ extra space.