Homework Assignment #4
Due: February 6, 11:30 a.m.

1. Let C be a class of objects. Let $A[1..n]$ be an array of references to objects of class C. You can test whether two references refer to the same object (as with the == operator in Java). However, you cannot test whether one object of type C is greater than another, because there is no comparison operator defined for class C.

An object o is called a *majority element* of the array $A[1..n]$ (where $n > 0$) if more than $\frac{n}{2}$ entries of A contain a reference to o.

(a) The following DISCARD routine splits the array $A[1..n]$ into $\left\lfloor \frac{n}{2} \right\rfloor$ pairs. For each pair, if the two elements of the pair are identical, one copy of that element is placed in another array B; otherwise neither element is copied into B. Prove that the following algorithm satisfies its postconditions.

```
DISCARD(A[1..n])
    preconditions: n > 0 and for each i, A[i] is a reference to an object of class C
    j = 0
    for i = 1.. ⌊n/2⌋
        if A[2i − 1] = A[2i] then
            j = j + 1
            B[j] = A[2i]
        end if
    end for
    return B[1..j]
    postconditions: for all o, if o is a majority element of A[1..n] and n is even then
            j > 0 and o is a majority element of B[1..j]
end DISCARD
```

(b) Give an example of an array $A[1..n]$ (where n is a positive even number) such that $A[1..n]$ does not have a majority element, but the array $B[1..j]$ computed by the DISCARD routine does have a majority element.

(c) Prove that if n is odd and o is a majority element of $A[1..n]$, then either o is a majority element of $A[1..n-1]$ or $o = A[n]$.

(d) Give an algorithm that returns a majority element of $A[1..n]$ if it exists, or returns "no majority element" otherwise. Your algorithm should run in $O(n)$ time. You should explain why your algorithm is correct and why its worst-case running time is $O(n)$, but you need not give a full, formal proof of correctness. You may assume that checking the equality of two references can be done in constant time.

Hint: use the previous parts to design a divide-and-conquer algorithm.