Sam Hopkins
Hw 5
2.19, 2.23

2.19) a) using 2.3 merge, if we can assume we have k arrays, each with n elements, To merge the first two arrays, we have nxn or 2n (each array is size n) The second two are 2nxn, third take 3nxn and so fourth. The total time complexity would be $O(k^2n)$ for all arrays

b) Instead of merging each array into one big one, we merge all the smaller ones together first, so after first iteration, we have $k/2$ arrays of size 2n, then we repeat and wind up with $k/4$ arrays of size 4n, by repeating this $\log_2 k$ times, each merge is only 2n, so we get complexity $O(nk \log k)$

2.23) a) if a variable x is in more than half of A_1 and A_2, then A has a majority element. when checking if both have a majority element, time is $O(n)$ if element is the same, If not, still $O(n)$ to count total found in either A_1 or A_2. Can be done recursively, so:

$T(n) = 2T(n/2) + n$ $a=2$ $b=2$ $d=1$, which leads to $O(n \log n)$

(2 probles; each size n/2; takes n time for checking for majority)

b) We pair up elements and get $n/2$ pairs, if they are the same element, we keep one, otherwise we through it out. At most, there are n majority elements in A. We divide that in pairs, and toss out a single one from each pair, we are left with $n/2$ elements. As we recombine those elements in the same manner we are either left with one element or none. If left with one, we iterate over the original array and check for majority, if it is then array A has a majority element. Every step here only leads to $O(n)$ complexity, and by master theorm, we end up with $O(n^d)$ with $d=0$, so total time of the function is $O(n)$ time, with $n/2$ elements maximum.