Algorithm 2

**Description of algorithm:**

After finding a recursive algorithm that took O(nlog(n)) time, we were curious how we could make the algorithm even quicker.

This O(n) algorithm to find number of delegates in the majority party two, individual for loops. The first loop to find the majority element and the second to count how many times that element is in the list. To find the majority element we choose the first element in the list as the candidate index and start a count at 1. As we traverse the array, we compare the candidate index to every element and either increment or decrement the counter. If the candidate is in the same party as the current index, then the counter is increased by 1 otherwise the counter is decreased by 1. If at any point the counter drops to 0 then the current index becomes the new candidate, and we continue through the array comparing the new candidate to whatever elements are left. When the whole array has been traversed whatever index holds the candidate must be the majority index.

**Run Time Analysis:**

Analyzing the running time for this algorithm we see that it is composed of two for loops, which each perform the function same\_party() n times. The rest of the algorithm is composed of assignments and conditional statements which are just constant time. Therefore the time complexity for this algorithm is T(n) = 2O(n) + O(1) = O(n)

**Proof of Correctness:**

I will attempt to prove this using induction and give a more long winded answer after in case my induction is incorrect. We just need to prove that when the first element in the list is a majority element then the algorithm ends with the majority element still:

Base case is n=1: The algorithm works because there is only one element in the list and the majority element has been found.

Induction hypothesis: We assume that for when n > 1 the algorithm will end on the majority element

Case 1: The first element is the majority element. Since there are n elements and the majority element appears at least n/2 times if the count starts at 1 then n+1/2 > n/2

Case 2: the first m elements are not the majority element. In this case we will eventually just hit a majority element and refer to case 1.

Need to fix this probably

For an array of n numbers this algorithm will output any number that occurs more than n/2 times. This leaves us with a maximum of n/2-1 elements that are not the majority and the majority element will occur at least n/2+1 times. We now have an intuition to why the algorithm will always end on a majority index but we need to show that this is the case. We can use induction to prove this. We know that the list will end on a majority candidate if the list starts on one. This happens because when you cancel out every non-majority element with a majority element (essentially what the algorithm does) there is still one majority element left. If the list does not start on a majority element it will switch off between non-majority elements until it finally does land on a majority element and we can now call this the beginning of the list and we can refer to the case of a majority index starting the list.