**Algorithm:** Balanced Array Check

**Input:** Array

**Output:** boolean

i←0

j←0

balanced←true

**If** n%2=0 **{**

j←0

**While** **(**(i ≤ n-1) & (balanced = true)**)**{

**While** **(**(j ≤ n-1) & (A[j] ≠ A[i])**)**{

j++

}

If j=6{

balanced←false

return false

}

i++

}

If balanced = true {

**return** true

}

**}**

**Else** return false **{**

**}**

**Proof of Correctness for Balanced Array Check**

**a) Algorithm Balanced Array Check terminates after a finite amount of time**

There are many instances in the above algorithm that ensures it terminates. To prove this, the algorithm first checks if the amount of elements in the array is an even number with the command **If n%2=0**. If the array has an odd number of elements it will return **false** and terminate immediately, otherwise, the algorithm will continue. Observe that the variables *i* and *j* are initialized to be zero and the boolean variable *balanced* is set to true. The variables *i* and *j* are used to iterate and compare elements in the array*.* The boolean variable *balanced* exists so that the outer **while** loop terminates as soon as the algorithm discovers that the array is not balanced, this will be further discussed later.

The variable *i* initially takes value zero and after each iteration of the outer **while** loop the value of *i* increases by 1. The condition of the outer **while** loop **i ≤ n-1** limits the number of iterations so that the loop will end as soon as *i* = *n*.

The variable *j* initially takes value zero and after each iteration of the inner **while** loop the value of *j* increases by 1. The condition of the inner **while** loop **j ≤ n-1** limits the number of iterations so that the loop will end as soon as *j* = *n*. The other condition **(A[j] ≠ A[i])** ensures that as soon as the additive inverse of the element at *i* is found then the inner **while** loop will end.

If there is no additive inverse in the array, that is, the array is unbalanced, then the outer **while** loop will terminate. This happens because if there is no additive inverse in the array then *j* will take the value of 6. This means that the pointer variable *j* has transverse through the entire array, could not find the respective additive inverse for the element at *i,* and through the last iteration of the inner **while** loop incremented to be 6. When *j* takes value 6 the algorithm will not iterate through the inner **while** loop because of the condition, and instead, will go to the **If j=6** statement. This **if** statement makes the variable *balanced* take **false** and will violate the outer **while** loop’s condition **(balanced = true)**, thus ending the outer loop.

**b) Algorithm Balanced Array Check outputs the correct answer**

This algorithm returns a Boolean value regarding whether the inputted array is balanced or not. The algorithm first checks if the amount of elements in the array is an even number because if the array has an odd number of elements then one of the elements will not have a respective additive inverse. If the array contains an odd number of elements then the algorithm will return **false** because the array is not balanced. If the array contains an even number of elements then the algorithm continues

**Worst Case and Complexity for Balanced Array Check**