Time Spent: 2 hours

Collaborators and Resources: Worked with Nathaniel MacArthur-Warner on coming up with strategies and discussing run-time

---

Problem 1 (a)

For part (a), I initially considered using a hash table but realized that the purpose of the problem was not just to solve it but also to approach it in a more thoughtful way. Having had some initial exposure to arrays in data structures, I wanted the run-time to be a linear multiple of n (that is, to have n increments in the worst-case scenario and make a linear number of comparisons in each case) ...

Below I've created the pseudo-code for my actual algorithm ...

Given a sorted array $A[1, \ldots, n]$ of n integer (bias) values

1. Initialize i = 0 and initialize j = length(A) - 1
2. Initialize response = "No such pair exists"
3. While i $\leq$ j:
   (a) If $A[i] + A[j] > 0$, decrement j
   (b) Else if $A[i] + A[j] < 0$, increment i
   (c) Else if $A[i] + A[j] == 0$,
   response = "Debater " + i + " and Debater " + j
4. Print response

Claim 1. The algorithm has worst-case runtime of O(n)

Proof. The reason it has linear runtime is that it can only make n total increments (an increment is defined as either an increase of i or a decrease of j in this case). In order to make each increment, we just check whether i $\leq$ j (one comparison) and then can only make a maximum of 3 more comparisons

(>, < or == 0).
The difference between i and j is n, so we can only make 4*n comparisons maximum which suggests a runtime of O(4n) which is O(n). ... □

Claim 2. The algorithm returns a pair, if it exists, of debaters whose bias add up to zero. It indicates if no such pair exists.

Proof. Because the array is sorted from smallest to largest, i initially denotes the smallest possible value in the array and j initially denotes the greatest possible value in the array.
There is no case in which we return a false negative. $A[i]$ is always $\leq A[j]$, so incrementing i instead of decrementing j in part (a) would actually bring the sum further away from zero.
A similar situation occurs for part (b), which means that we are always making the correct increments. We're not missing any pairs because increasing the absolute value of the sum will bring us further from zero. ... □

Problem 1 (b)

For part (b), which is predicated on the fact that the algorithm in part (a) is O(n), I elected to place the third debater of comparison outside of the scope of algorithm 1, keeping everything else equal except for the definition of the array. Working with the same concept and comparing three debaters instead of two, we print the default response (false) unless we find a sum that equals 0 ...

I've written out the algorithm below ...

Given a sorted array $A[1, \ldots, n]$ of n integer (bias) values

1. Initialize response = "No such pair exists"
2. For k in 1:n
   (a) newA = A with $A[k]$ removed
   (b) Initialize i = 0
   (c) Initialize j = length(newA) - 1
   (d) While i $\leq$ j:

i. If new$A[i]$ + new$A[j]$ + new$A[k] > 0$, decrement j
ii. Else if new$A[i]$ + new$A[j]$ + new$A[k] < 0$, increment i
iii. Else if new$A[i]$ + new$A[j]$ + new$A[k] == 0$,
response = "Debater " + i + " and Debater " + j

3. Print response

Claim 3. The algorithm has worst-case runtime of O(n^2)

Proof. Based on the fact that we are now running a O(n) algorithm on n different arrays (for each k value from 1 to n), we know that we are running the previous algorithm a multiple of n times. ... □

Claim 4. The algorithm returns a triple, if it exists, of debaters whose bias add up to zero. It indicates if no such triple exists.

Proof. The reason that this is the case is that for each $A[k]$, all necessary pairs are compared to determine if they and $A[k]$ add up to equal 0. We do not need to make any more comparisons because we avoid certain redundancies (if we've compared -5 to 3, we don't need to compare -7 to 3 as well). ... □