Problem 3:

Pre-Implementation Notes:

- I would like to start with a prefix sum of all of the elemnts of the array
- Then binary search until there is a position that is just the breaking point of being close to equal ie the first time the `first_half` > `second_half`.
- Then go through the first half until there is a position that is equivalent to half of the difference which implies that it is possible

ISSUE: i realised that there is an issue with this algorithm firstly, I really wanted to use a binary search because i thought this would be possible, secondly I assumed that there wouldnt be any other positions of the array where the split down is okay to create a equivalent 2 arrays after 1 move.

There might be a position where the difference is really big but there exists an element in the array that's enough to make up for it so just because i'm close to the equivalent point doesn't necessarily bring us closer to the solution.

Second attempt:

1. Prefix sum off the array (I think this is going to be necessary for the implementation, also because i can create a prefix sum while receiving the flow of cin values technically it's free in terms of time complexity)
2. Initialise counter hashed maps between the elemtns & its occurence count for the first and second half of the position.
3. Iterate through the input array adding & removing from the hash maps. As of now we will have 4 key data structures:
    a. `given_array` Just a `vector<long long>` of input values
    b. `given_array_prefix_sum` Again just a `vector<long long>` sum of input values
    c. `first_half_counter` This is a map of the input values and it's occurence `unordered_map<ll, int>` this is going to be empty

d. `second_half_counter` This is a map of the input values and it's occurence `unordered_map<ll, int>` this is going to start filled out.

4. Iterate through `i` for the range of `n`. This is the reason why we wanted the first half hashed map to be empty because we're not starting with anything in the first half and we should start with the second half.
5. Everytime we iterate through, calculate the difference between the sum of the first and second half of the arrays. and check if there is a an element equivalent to `diff / 2` in the bigger side (can be checked quickly with the hashed map) and if there is that means the answer is YES otherwise keep going with next values of `i`

Post-Implementation Notes:

Accepted on first attempt