0002\_Medium\_3Sum\_#15\_Breakdown

Problem:

* Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.
* Notice that the solution set must not contain duplicate triplets.

Examples:

* **Example 1:**
* **Input:** nums = [-1,0,1,2,-1,-4]
* **Output:** [[-1,-1,2],[-1,0,1]]
* **Explanation:**
* nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
* nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
* nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
* The distinct triplets are [-1,0,1] and [-1,-1,2].
* Notice that the order of the output and the order of the triplets does not matter.
* **Example 2:**
* **Input:** nums = [0,1,1]
* **Output:** []
* **Explanation:** The only possible triplet does not sum up to 0.
* **Example 3:**
* **Input:** nums = [0,0,0]
* **Output:** [[0,0,0]]
* **Explanation:** The only possible triplet sums up to 0.

Examples of Note:

* The first example notes that the order output and order of the triplets does not matter.

Questions:

* Can I sort the array?
* Is space an issue?

What needs to be true for the problem to work:

* 3 distinct elements need to add up to 0, they cannot have the same index.

How would a person solve this:

* Add the first two numbers, iterate through the array looking for a third number that would result in zero, check the entire array, as there are duplicates. Move on to the second and third number and repeat.
* Sort the array. Look at the first element, iterate through the rest of the array, and for each of those elements iterate through the array again, looking for 3 numbers that add up to 0.

Brute Force:

* 3 nested iterations, comparing each set of numbers for sum to 0.
* Time Complexity: O(n^3), for each element of the array the array is iterated through, and for each of those elements the array is iterated through.
  + Specifically: O(n((n-1)/2(n-2)/2)), as each iteration occurs the subsequent iterations get smaller.
* Space Complexity: O(1), no extra space is needed.

Optimize (BUD, bottlenecks, unnecessary work, duplicated work)

* The array can be sorted, O(nlogn) is smaller than O(n^3), so as long as the end time complexity is less O(n^3), sorting makes the algorithm more efficient.
* Once the array is sorted, instead of 3 numbers adding to 0, the problem can be manipulated to looking for two numbers that equal zero.
  + To equal zero, either all the numbers are 0, or there must exist negative numbers.
  + Because the array is sorted, the left-most numbers are negative or starts with 0.
  + So taking each element that is less than or equal to zero, and then iterating through the rest of the array to find two elements that equal a positive of the current element will result in a sum of 0.
* For each element of the array less than or equal to zero, iterate through the rest of the array, two pointers can be used to represent a low and high starting at one element to the right of the current element for low and one at the end of the array as high, the pointers are then manipulated until a sum is found that is a positive to the current number.
* Time Complexity: O(n^2), for each element of the array the array is iterated through.
  + Specifically: O(nlogn + n(n-1)/2), the array must first be sorted, then upon each iteration of the array subsequent iterations get smaller.
* Space Complexity: O(1), depending on the sort algorithm used.

Pseudocode:

* Sort the array.
* Create a list to hold the triplets.
* Iterate through the array for each element that is less than or equal to zero.
  + Check if the previous element is the same, if it is then the triplet will be the same, so skip it.
  + A method can be created to be reused
* After the array is iterated through return the master list

Method:

* Create two pointers, a low that is directly to the left of the current position, and a high that is at the end of the array.
* While low is less than high
  + Create a integer called sum that current, low, and high.
  + If sum is less than zero, increment low by one.
  + Else If the sum is greater than zero, decrement high by one.
  + Else add the index of current, the index of low, and the index of high to the list and add it to the master list.
    - Because the array still needs to be search for other matches, increment low and decrement high.
  + While low is still less than high and that the next low is the same as the previous, move to the next low.