Neetcode

Arrays and hashing

Hash table has a unique way of storing items using a hash function in case where a hash function computes the same thing for two items we have what we call the hash collision.

To solve this we use collision resolution( an example is separate chaining where we attach a linked list to each index ) -> open addressing

In python a hash map is a dictionary, that has a key value pair

NB: keys in a hash map are immutable.

In Python, a **defaultdict** is a specialized dictionary from the **collections** module. It behaves like a regular dictionary but has a key difference: if you try to access or modify a key that doesn't exist in the dictionary, the **defaultdict** will automatically create an entry for that key with a default value. The default value is specified when the **defaultdict** is created and can be any type such as int, list, set, etc.

First Question – Contains Duplicate

My Brute force solution

class Solution:

def containsDuplicate(self, nums: List[int]) -> bool:

# brute force approah iterate to every single element twice and ignore the element of comparison using two for loops

n = len(nums)

for i in range(n):

for j in range(n):

if i != j:

if nums[i] == nums[j]:

return True

return False

This solution I know has O(n^2) hence it is very inefficient and not the best, but it is always good to comprehend the question from the brute force perspective then begin to optimize from there.

Second solution

class Solution:

def containsDuplicate(self, nums: List[int]) -> bool:

# using the dictionary to count each element then we and find if a count is more than 1

dict = {}

for i in nums:

if i in dict:

dict[i] += 1

else:

dict[i] = 1

for i in dict.values():

if i > 1:

return True

return False

This solution used hashmaps -> dictionary and has a time complexity of O(n ) with the most expensive computation being checking each elements, however this must be done hence it is time efficient, the only problem might be with the space complexity.

def containsDuplicate(nums):

# I believe this should be the most efficient one

# here we will use a set and then check if an element is in the set, once we find it the code ends and returns saving space and time

# create a set called seen to keep track of the elements

seen = set()

#itereate throguh and add elements to seen if there are new

for num in nums:

if num in seen:

# once we see the number already in the seen set, we end implying there is a duplicate

return True

seen.add(num)

print(containsDuplicate([1,2,1]))

This is the most efficient since it has a space and time complexity of O(n), hence in an interview this is what I will go for in the end as the most optimized solution.