ARRAYS AND HASHING

(1) CONTAINS DUPLICATES

```
def contains_duplicate(nums):        #o(1)
  saw = set()
  for num in nums:                   #o(n)
    if num in saw:
      return True                    #o(1)
    else:
      saw.add(num)
  return False
print(contains_duplicate([1, 2, 3, 4]))
print(contains_duplicate([1, 2, 1, 1]))
```

(2) VALID ANAGRAMS

```
class Solution:
  def isAnagram(self, s: str, t: str) -> bool:
      if len(s) != len(t):
          return False

      s_dict = {}
      t_dict = {}

      for i in range(len(s)):

          if s[i] in s_dict:
              s_dict[s[i]] += 1
          else:
              s_dict[s[i]] = 1


          if t[i] in t_dict:
              t_dict[t[i]] += 1
          else:
              t_dict[t[i]] = 1

      return s_dict == t_dict

solution = Solution()


print(solution.isAnagram("anagram", "nagaram"))
print(solution.isAnagram("rat", "car"))
```

(3) ENCODE DECODE STRINGS

```
class Codec:
  def encode(self, strs):
      res = ""
      for s in strs:
          res += str(len(s)) + "#" + s + "#"
      return res



  def decode(self, encoded_str):
      res = []
      i = 0
      while i < len(encoded_str):
          j = i
          while encoded_str[j] != "#":
              j += 1
          length = int(encoded_str[i:j]) #extracting the length of the string
          start = j + 1
          end = start + length
          res.append(encoded_str[start:end])
          i = end + 1

      return res

# Example usage
codec = Codec()
encoded = codec.encode(["hello", "world"])
print("Encoded:", encoded)

decoded = codec.decode(encoded)
print("Decoded:", decoded)


(4) GROUP ANAGRAMS

def group_anagrams(strs):
  anagrams = {}
  for word in strs:
      count = [0] * 26  # There are 26 letters in the English alphabet
      for char in word:
          count[ord(char) - ord('z')] += 1
      key = tuple(count)
      if key not in anagrams:
          anagrams[key] = [] #[eat]
      anagrams[key].append(word)
  else:
      return list(anagrams.values())

# Example usage
```

```
strs = ["eat", "tea", "tan","etn", "at", "na", "bat"]
print(group_anagrams(strs))
```

(5) LONGEST CONSECUTIVE SEQUENCE

```
def longest_consecutive_sequence(numbers):
  unique_numbers = set(numbers)
  max_length = 0

  for number in numbers:
      # Check if this is the start of a sequence
      if number - 1 not in unique_numbers:
          current_number = number
          current_sequence_length = 1

          # Extend the sequence
          while current_number + 1 in unique_numbers:
              current_number += 1
              current_sequence_length += 1

          # Update the maximum length
          max_length = max(max_length, current_sequence_length)

  return max_length

print(longest_consecutive_sequence([2, 20, 4, 10, 3, 4, 5]))
print(longest_consecutive_sequence([0, 3, 2, 5, 4, 6, 1, 1]))
```

(6) TWO SUM

```
def two_sum(nums, target):
  num_dict = {}
  for index, num in enumerate(nums):
      complement = target - num
      if complement in num_dict:
            return {num_dict[complement], index}
      else:
            num_dict[num] = index

print(two_sum([2, 1, 7, 15], 9))
print(two_sum([3, 6, 4], 6))
print(two_sum([3, 3], 6))
```

(7) PRODUCT OF ARRAY EXCEPT SELF

```
def product_except_self(nums):
```

```
  n = len(nums)
  result = [1] * n

  #  Calculate prefix products
  prefix_product = 1
  for i in range(n):
    result[i] = prefix_product
    prefix_product *= nums[i]

  #  Calculate suffix products and combine with prefix products
  suffix_product = 1
  for i in reversed(range(n)):
    result[i] *= suffix_product
    suffix_product *= nums[i]

  return result

# Example usage:
nums = [1, 2, 9, 4]
print(product_except_self(nums))


(8) TOP K FREQUENT ELEMENTS

from collections import Counter


def topKFrequent(nums, k):
    # Step 1: Count the frequencies
    frequency = Counter(nums)

    # Step 2: Create buckets
    bucket = []  # Initialize an empty list
    for i in range(len(nums) + 1):  # Loop through the range
        bucket.append([])  # Append an empty list to 'bucket'


    # Step 3: Fill the buckets
    for num, freq in frequency.items():
        bucket[freq].append(num)

    # Step 4: Collect the top k frequent elements
    result = []
    for i in range(len(bucket) - 1, 0, -1):
        for num in bucket[i]:
            result.append(num)
            if len(result) == k:
                return result

# Example usage
```

```
nums = [2,2,2,1,1,1,3,3,3]
k = 2
print(topKFrequent(nums, k))  # Output: [1, 2]
```