**Contains Duplicate => Leetcode 217**

Data structure(s): Hashset.

Steps:

* Initialize a HashSet
* Insert each item in the input array into the HashSet, a HashSet does not retain duplicates, so if the element already exists, it won’t be inserted.
* If the hashset can’t insert the element, then return true => duplicate exists.
* If the hashset inserts all elements, then no duplicate exists. Return false.

Big O:

* Time Complexity: O(n): where n is the number of elements in the array
* Space Complexity: O(n): at most, if all the items in the array are unique, they would be inserted into the hashset

**VALID ANAGRAM => Leetcode 242**

Data structure: Array

Steps:

* Initialize 2 arrays of size 26 each, representing every element of the alphabet
* For each element in each word, get the ascii value and increment the count of that value in the corresponding array.
* Compare both arrays. If they have equal elements then it’s a valid anagram

Big O:

* Time Complexity: O(n) + O(1) => O(n)
* Space: O(1)

**TWO SUM => Leetcode 1**

Data Structure: Dictionary

Steps:

* Initialize a dictionary of K,V int
* Foreach element in the array, check if the complement exists in the dictionary
* Complement = Target – element
* If it doesn’t exist, insert the element as the key, and the integer as the value into the dictionary
* Otherwise, return the index of the complement and the index of the element in a new array.

Big O:

Time complexity: O(n).

Space Complexity: O(n)

**GROUP ANAGRAMS => Leetcode 49**

Data structure(s): dictionary, array

Steps

* Initialize a dictionary of K,V => string, list of strings
* For each element in the array:

1. Initialize a char array of size 26
2. For each item in the char in the element, get the ascii value and increment it in the char array
3. Convert the array to a string and check if this string exists as a key in the dictionary
4. If it does, add the element to the list of values in the dictionary
5. In the end, return the values in the dictionary as a list

Big O:

Time: O(n)

Space: (On)

**TOP K FREQUENT ELEMENTS => Leetcode 347**

DS: dictionary, sorted set, array

Steps:

* Initialize a dictionary of KV => int, int
* For each element in the array:

1. If the dictionary contains the element, increment the value by 1.
2. Else, add the element.

* Initialize a sorted set with element and frequency
* For each element in the dictionary:

1. Add the element and its value to the set
2. If the count of the set is greater than k, remove the minimum element
3. Do this till we’ve touched every element in the dictionary

* Initialize an output array of size k
* Add each element in the set to the out array
* Return the output

Big O:

Time: O(n + k)

Space: O(n + k)

**PRODUCT OF ARRAY EXCEPT SELF => Leetcode 238**

DS: array

Steps:

* Initialize a temp variable of type int
* Initialize a result array
* Forward pass:

1. For each element in the input array:
2. Set the element at the current index in result to temp
3. Multiply temp with the element at the current index in the input array and reassign temp

* Initialize temp to 1
* Backward pass:

1. Iterate through the input array from the end
2. At each iteration, multiply the corresponding item in result with temp and reassign the item
3. Multiply temp with the corresponding item in the input array and reassign temp

* Return the result array

Big O:

* Time: O(n)
* Space: O(n)

**VALID SUDOKU => Leetcode 36**

DS: HashSet, Array

**Steps**:

* Initialize an array of sets of 9 each for the row, the column, and the 3\*3 boxes
* For every row and column get the current char at that rc index
* Check if the char is a number. If it is:

1. Check if it exists in the corresponding set for the row and the column.
2. If it doesn’t add it in, if it does, return false.
3. Repeat step 2 for the box.

* If the char isn’t a number, continue.

Big O:

Time: O(n2)

Space: O(n2)

**LONGEST CONSECUTIVE SEQUENCE => Leetcode 128**

DS: HashSet

Steps:

* Initialize a set of the input array
* Initialize an integer variable for maxSize 0
* Foreach element in the input array, if the set doesn’t contain (element-1):

1. Initialize a count variable with the value of 1
2. While the set contains (element + 1):

* Increment count

1. Compare count and MaxSize and pick the greater of the two

* Return maxSize