ARRAYS

No.	Problem Statement	Solution	Time complexity	Space complexity
1	**Contains Duplicate**			
	Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.	Approach_1: **Sorting** - Sort the array - `If nums[i-1] == nums[i] return true`	O(N*logN)	O(1)
		Approach_2: **Unordered_Map** - mp[x] = # of occurrences of x in the array - Insert element in the map one after the other and increase their frequency count - `If mp[x] >= 2 return true`	O(N)	O(N)
		Approach_3: **Unordered_Set** - Insert element in the set one after the other - If the size of the set still remains the same after inserting a new element return true	O(N)	O(N)
2	**Find All Duplicates**			

	Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or twice, return an array of all the integers that appears twice.	- Use the sign of elements in the array to mark visited elements
		`for(auto x: nums)`
		`    int i = abs(x);`
		`    nums[i-1] *= -1;`
		`    if(nums[i-1]>0) ans.push_back(i);` // Indicates that 'i' is the duplicate
			O(N)	O(1)

3	**Valid Anagram**			
	Given two strings s and t, return true if t is an anagram of s, and false otherwise. *(An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.)*	Approach_1: **Sorting** - `return s == t`	O(N*logN)	O(1)
		Approach_2: **Unordered_Map** - Insert characters of string 's' in mp - Traverse string 't' - `if(mp.find(c)==mp.end()) return false` - `mp[c]--` - `if(mp[c]==0) mp.erase(c)`	O(N)	O(N)

		Approach_3: **Vector of size 26**
		- for string s: `v[c] ++`
		- for string t
		`  if(v[c] == 0) return false;`
		`    v[c] --`
			O(N)	O(N)

4	**Two Sum**			
	Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. *(You may not use the same element twice)*	- Unordered_map - **mp[x] = i --> at index 'i' we need 'x' so that nums[i] + x = target** - Traverse the 'nums' array - `if(mp.find(nums[i]!=mp.end())` - `    if(mp[nums[i]] != i) return {mp[nums[i]], i}`	O(N)	O(N)
5	**Group Anagrams**			
Ama zon	Given an array of strings strs, group the anagrams together. You can return the answer in any order.	- **Unordered_map<string, vector<string>> + Sort** - Idea: For each string 's' the **key** in the unordered_map will be **sorted version of 's'** - So, all the anagrams will have the same 'key' in the unordered_map	O(N*L*logL) *L = max length of a string in 'strs'*	O(N*L)
		- **Unordered_map<string, vector<string>> + Counting Sort** - Idea: Same as above	O(N*L) *L = max length of a string in 'strs'*	O(N*L)
6	**Top K Frequent Elements**			

ARRAYS

No.	Problem Statement	Solution	Time complexity	Space complexity
	Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.	Approach_1: 'Priority_queue + Hashing' - Unordered_map: **{key, value} = {frequency of 'x', x}** - Priority_Queue<pair<int, int>> : key = frequency of x	O(N* logN)	O(N)
		Approach_2: 'Bucket sort' - **vector<vector<int>> bucket(n+1)** --> 2D Vector as there can be multiple elements with same frequency - **bucket[i] : vector of all the elements with frequency = i** - Maximum freq an element can have is n --> Size of bucket array can be atmost n+1	O(N)	O(N)
7	**Product of Array Except Self**			
	Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].	- Approach_1: '**Calculating Product of All the Elements in the Array** ' - Edge cases: nums[i] can be 0 - if num_zeros = 1 then 'ans' vector is all zeros except at index 'i' where nums[i] = 0 - if num_zeros = 2 then 'ans' vector is all zeros - ans[i] = product / nums[i]	O(N)	O(1)
		- Approach_2: '**Prefix product & postfix product** ' - prefix product: ans[i] = product of all the elements before nums[i] - postfix product: ans[i] = product of all the elements after nums[i] - Eventually, ans[i] = product of all the elements except nums[i]	O(N)	O(1)
8	**Valid Sudoku**			

	Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:
	- Each row must contain the digits 1-9 without repetition.
	- Each column must contain the digits 1-9 without repetition.
	- Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.
		- **Three 2-D boolean arrays** to keep track of numbers between '1 - 9'
		- **Idea**: To check 'val' can exist only **once** in any **row, col and box**
		- `vector<vector<bool>> row(9, vector<bool>(9, false));`
		`vector<vector<bool>> col(9, vector<bool>(9, false));`
		`vector<vector<bool>> box(9, vector<bool>(9, false));` // box[i] represents box number = 'i'
		`for(int i=0; i<9; i++)`
		`   for(int j=0; j<9; j++)`
		`       if(board[i][j]=='.') continue;`
		
		`       int num = board[i][j] - '0';`
		`       int box_number = ((i/3) * 3) + (j/3);`
		
		`       if(row[i][num-1] || col[num-1][j] || box[box_number][num-1]) return false;`
		`       row[i][num-1] = true; col[num-1][j] = true; box[box_number][num-1] = true;`
			O(N^2)	O(N^2)

9	**Encode Decode Strings**			

	Design an algorithm to encode: list of strings -> string
	Design an algorithm to decode: string -> list of strings
		- Encode:
		For **each word** in a list of strings encode: **{len(word), #, word}**
		Note: Keep in mind that the len(word) can be more than one digit hence follow it by '#'
		- Decode:
		`int i = 0;`
		`while(i<s.size())`
		`     int j=i;`
		`     while(s[j]!='#') j++;`
		`     int len = stoi(s.substr(i, j-i));`
		`     ans.push_back(s.substr(j+1, len));`
		`     i = j + len + 1;`
			O(N)	O(1)

10	**Longest Consequitive Sequence**			
	Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.	- **Unordered_set** : Traverse through each element 'x' in set - Only check for longer sequence if 'x' is the beginning of the sequence --> `s.find(x-1)==false` - If 'x' is the beginning of the sequence then enter a loop to count the consecutive numbers starting from x `cnt = 1;     while(s.find(x+cnt)) cnt++`	O(N)	O(N)