DSA_LEETCODE_MASTERY

Table of Contents

Array & Hashing

1 Concatenation of Array [Easy] [Google, Adobe, Facebook]

- https://leetcode.com/problems/concatenation-of-array/

Ques

You are given an integer array nums of length n. Create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (0-indexed).

Specifically, ans is the concatenation of two nums arrays.

Return the array ans.

Example 1:

```
Input: nums = [1,4,1,2]

Output: [1,4,1,2,1,4,1,2]
```

Example 2:

```
Input: nums = [22,21,20,1]

Output: [22,21,20,1,22,21,20,1]
Constraints:
```

1 <= nums.length <= 1000. 1 <= nums[i] <= 1000

PreReqs

```
#include<bits/stdc++.h>

int main(){
    // Vector Array
    std::vector<int> arr1; // empty vector
    std::vector<int> arr2={1,2,3}; // initialised vector
    std::vector<int> arr3(5); // size =5, each element value=default value=0
    std::cout<<arr3[0]<<std::endl; //0 // Direct Access (0based)
    std::vector<int> arr4(5,90); // size =5, each element value=90
    std::cout<<arr4[0]<<std::endl; //90
    std::cout<<arr1.size()<<std::endl; //0 // Get Lenght
    arr4.push_back(5); // Append // btw each push in dynamic array is O(1)
    std::cout<<arr4[5]<<std::endl; //5

    // For Loop
    for ( int i=0; i<5; i++) std::cout<<i;
    // 01234

    return 0;
}

// 0
// 90
// 0
// 5
// 0123
```

Solutions

- Basically We have to make final array of 2x input array
- We can't do just Vector Concatnation like strings
- Solution 1
    - assume nums as input vector
    - n = size of vector
    - make new arr1 of size 2n default 0
    - for loop 0 to n-1
        - arr1[i]= nums[i]
    - for loop 0 to n-1
        - arr1[i+n]=arr[i]
    - return arr1

```
// Solution 1
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
```

```
        vector<int> ans(2*n);
        for (int i=0; i<n; i++){
            ans[i]= nums[i];
        }
        for (int i=0; i<n; i++){
            ans[n+i]= nums[i];
        }
        return ans;
    }
};
// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 2
  - Iteration (One Pass)
  - assume nums as input vector
  - n = size of vector
  - make new arr1 of size 2n default 0
  - for loop 0 to n-1
    - arr1[i]= nums[i]
    - arr1[i+n]=arr[i]
  - return arr1

```
// Solution 2
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        vector<int> ans(2*n);
        for (int i=0; i<n; i++){
            ans[i]= nums[i];
            ans[i+n]= ans[i];
            // ans[i] = ans[i + n] = nums[i]; // we can write this too !!!

        }
        return ans;
    }
};
// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 3 -- optimal
  - not making any new array ,saving space
  - assume nums as input vector
  - n = size of vector

    - for loop 0 to n-1
        - nums.push_back(nums[i])
    - return nums
- btw each push in dynamic array is O(1)

```cpp
// Solution 3
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();

        for (int i=0; i<n; i++){
            nums.push_back(nums[i]);
        }

        return nums;
    }
};

// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 4
    - Iteration (Two Pass)
    - generic sol. with times var, here =2
    - assume nums as input vector
    - n = size of vector
    - for loop 1 to times-1
        - for loop 0 to n-1
            - nums.push_back(nums[i])
    - return nums
- btw each push in dynamic array is O(1)

```cpp
// Solution 4
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        int times= 2;
        for (int j=1;j<times; j++)// j is 1 because ones occurence is already here
        {
            for (int i=0; i<n; i++){
                nums.push_back(nums[i]);
            }
        }
```

```
        return nums;
    }
};

// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

---

2 Contains Duplicate [Easy] [Airbnb, Amazon, Apple ,Microsoft, Tcs, Google, Yahoo, Oracle, Palantir-technologies, Adobe, Uber, Facebook, Bloomberg]

- https://leetcode.com/problems/contains-duplicate/

Ques

Given an integer array nums, return true if any value appears more than once in the array, otherwise return false.

Example 1:

```
Input: nums = [1, 2, 3, 3]

Output: true
```

Example 2:

```
Input: nums = [1, 2, 3, 4]

Output: false
```

Constraints:

1 <= nums.length <= 105 -109 <= nums[i] <= 109

Recommended Time & Space Complexity You should aim for a solution with O(n) time and O(n) space, where n is the size of the input array.

Hint 1 A brute force solution would be to check every element against every other element in the array. This would be an O(n^2) solution. Can you think of a better way?

Hint 2 Is there a way to check if an element is a duplicate without comparing it to every other element? Maybe there's a data structure that is useful here.

Hint 3 We can use a hash data structure like a hash set or hash map to store elements we've already seen. This will allow us to check if an element is a duplicate in constant time.

PreReqs

```
#include<bits/stdc++.h>
int main(){
    // Hashset
    std::unordered_set<int> hashSet;
    hashSet.insert(1);
    std::cout<<(
        hashSet.find(1)
        !=
        hashSet.end() // i.e if find give hashSet.end() ptr, then element doesnt
exist
    )<<std::endl; // true as 1 exists in hashset
    // 1


    // Sorting
    // Sorting in Vector
    // std::sort sorts the vector in-place and does not return a sorted vector.
Its return type is void
    std::vector<int> v1 = {1,4,2,0};
    std::cout<<"v1: "<<v1[0]<<" "<< v1[1] << " " << v1[2] << " " << v1[3] <<
std::endl;
    // v1: 1 4 2 0
    std::sort(v1.begin(), v1.end()); // asc default
    std::cout<<"v1(sorted): "<<v1[0]<<" "<< v1[1] << " " << v1[2] << " " << v1[3]
<< std::endl;
    // v1(sorted): 0 1 2 4
    std::sort(v1.begin(), v1.end(), std::greater<int>()); // dsc
    std::cout<<"v1(sorted dsc): "<<v1[0]<<" "<< v1[1] << " " << v1[2] << " " <<
v1[3] << std::endl;
    // v1(sorted dsc): 4 2 1 0


    // unordered sets in c++ contains unique elements
    // if we even insert duplicate, it will reject it
    // we can make unord set with method to copy full the vector array, rejecting
the duplicates
    v1.push_back(1);
    v1.push_back(1);
    std::unordered_set<int> hashSet2(v1.begin(), v1.end()) ;
    std::cout<< hashSet2.size() << std::endl; // 4 // unique elements in unord set
    std::cout<< v1.size() << std::endl; // 6 // duplicates elements in vect arr



    return 0;
}
```

```
// 1
// v1: 1 4 2 0
// v1(sorted): 0 1 2 4
// v1(sorted dsc): 4 2 1 0
// 4
// 6
```

Solutions

- So in this ques array
  - if any any element occurence >1 or have duplicates
    - return true
  - else
    - then the array have distinct elements
    - return false
- Solution 1 -- brute force
  - we are checking each element to find it's same value by traversing each array eachtime
  - given nums vect array
  - let n = nums vect array length
  - let counter = 0
  - if counter become more than 1 , i.e. item has multiple occurence
  - for i 0->n-1
    - for j 0->n-1
      - if nums[i]==nums[j] (not i==j)
        - counter++
        - if counter>1
          - return true
    - counter = 0 // reset counter for next elements turn
  - return false // at case where counter didnt inc from 1 , i.e. not even onces occurence

```
// Solution 1
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size();
        int counter = 0;
        for ( int i=0; i<n; i++){
            for ( int j=0; j<n; j++){
                if (nums[i]==nums[j]){
                    counter++;
                    if (counter> 1){
                        return true;
                    }
                }
            }
            counter=0;
        }
```

```
        return false;
    }
};
// // Time & Space Complexity
// // Time complexity:
// // O(n**2)
// // Space complexity:
// // O(1)
```

- Solution 2 -- optimal
  - hashset
  - efficient tc O(n)
  - directly checking if element's occurence > 1
  - using another ds hash set
  - we are inserting in hash set one by one & parallely checking if incoming element already exists in hash set, if it already exists, so it mean ;at that point of time its duplicate is incoming
    - so this hashset contain duplicates
    - return true and exit, no need to continue, we got our ans
  - else try until any occurence is duplicate
  - if not true at any case and didn't exited earlier
    - return false
  - now only 1 loop, which is even just insertion in ds is only req.
  - earlier i thought for an approach where we can remove that element in an array and still find if it's exist as duplicate or not. this Solution similar acts , by not deleting it but checking during genesis of array

```
// Solution 2
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size();
        unordered_set<int> hashSet;
        for ( int i=0; i<n; i++){ // or // for (int num : nums) {
            if (hashSet.find(nums[i])!=hashSet.end()){ // or // if
(find.count(num)) {
                return true;
            }
            else {
                hashSet.insert(nums[i]);
            }
        }
        return false;
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

- Solution 3
  - 2 Pointer Approach & Sorting
  - no need of new space
  - sort the array
    - O(nlogn)
    - then the duplicates would be adjacent to each other
    - even even once 2 adjacent elements are same, then we got our duplicate array proof
  - do one loop to just check if arr[i]==arr[i+1]
    - if true, return true
  - if loops ends without returning true;then array is distinct
    - return false

```
// Solution 3
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size(); // or just use the method in loop too
        sort(nums.begin(), nums.end());
        for ( int i=0; i<n-1; i++){ // we took n-1 as to not going to case (where
i=n-1 index(last index), i+1=n index(exceeding limit)), and now at i= n-2 (2nd
last element index), then i+1 = n-1(last element index) :)
            if( nums[i]==nums[i+1]) return true;
        }
        return false;

    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn)
// Space complexity:
// O(1) or O(n) depending on the sorting algorithm.
```

- Solution 4 -- optimal
  - Hash Set Length
  - we can just make set of distinct elements and compare size of old array, if same, then false, else true
  - unordered sets in c++ contains unique elements
    - if we even insert duplicate, it will reject it
  - we can make unord set with method to copy full the vector array, rejecting the duplicates
  - if set size < vector size, it means that vec had duplicate elements
    - return true
  - else return false

```
// Solution 4
class Solution {
```

```
public:
    bool containsDuplicate(vector<int>& nums) {
        return (
            unordered_set<int>(nums.begin(), nums.end())
            .size()
            <
            nums.size()
        );

    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

3 Valid Anagram [Easy] [Expedia, Affirm, Docusign, Yahoo, Cisco, Servicenow, Goldman Sachs, Amazon, Microsoft, Oracle, Morgan-stanley, Uber, Spotify, Zulily, Google, Paypal, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg]

Given two strings s and t, return true if the two strings are anagrams of each other, otherwise return false.

An anagram is a string that contains the exact same characters as another string, but the order of the characters can be different.

Example 1:

```
Input: s = "racecar", t = "carrace"

Output: true
```

Example 2:

```
Input: s = "jar", t = "jam"

Output: false
```

Constraints:

```
s and t consist of lowercase English letters.
```

Recommended Time & Space Complexity You should aim for a solution with O(n + m) time and O(1) space, where n is the length of the string s and m is the length of the string t.

Hint 1 A brute force solution would be to sort the given strings and check for their equality. This would be an O(nlogn + mlogm) solution. Though this solution is acceptable, can you think of a better way without sorting the given strings?

Hint 2 By the definition of the anagram, we can rearrange the characters. Does the order of characters matter in both the strings? Then what matters?

Hint 3 We can just consider maintaining the frequency of each character. We can do this by having two separate hash tables for the two strings. Then, we can check whether the frequency of each character in string s is equal to that in string t and vice versa.

Solutions

- So basically if both string array have same elements despite any order , they are anagram
- Solution 1
    - Sort Both String Arrays
    - check 1 if both lenght are equal or not
    - check 2 traverse and check each element of both array for equality

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        if (s.size() != t.size()) { return false; }
        for(int i=0; i<s.size(); i++){
            if (s[i]!=t[i]){
                return false;
            }
        }
        return true;


    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn+mlogm)
// Space complexity:
// O(1) or
// O(n+m) depending on the sorting algorithm.
// Where n is the length of string s and m is the length of string t.
```