array and hash

217. Contains Duplicate

```
#include<unordered_map>
#include<vector>
#include<iostream>
using namespace std;

class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_map<int,int> mp;
        for (int i = 0; i < nums.size(); ++i) {
            if (mp.count(nums[i]) > 0) return true;
            mp[nums[i]]++;
        }
        return false;
    }
};
```

242. Valid Anagram

```
bool isAnagram(string s, string t) {
        vector<int> coutS(26,0);
        vector<int> coutT(26,0);
        for (int i = 0; i < s.length(); ++i) {
            coutS[s[i] - 'a']++;
        }
        for (int i = 0; i < t.length(); ++i) {
            coutT[t[i] - 'a']++;
        }

        if (coutS == coutT)
            return true;
        else
            return false;
    }
test case: s.length() != t.length() //
```

1. Two Sum

```
vector<int> twoSum(vector<int>& nums, int target) {
        //hash table
```

```
        unordered_map<int, int> hash; //
        vector<int> res;
        for (int i = 0; i < nums.size(); ++i) {
            int num = target - nums[i];
            if (hash.count(num) > 0) {
                res.push_back(i);
                res.push_back(hash[num]);
                break;
            } else {
                hash[nums[i]] = i;
            }
        }
        return res;
    }
```

49. group anagram

```
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        // hash table with sorting method??
        unordered_map <string, vector<string>> mp;
        vector<vector<string>> res;
        for (string& s : strs) {
            string t = s;
            sort(t.begin(), t.end());
            mp[t].push_back(s);
        }

        for (auto& p : mp) {
            res.push_back(p.second);

        }
        return res;
    }
```

347. Top K Frequent Elements

```
vector<int> topKFrequent(vector<int>& nums, int k) {

        //hashtable
        //bucket sort
        unordered_map<int, int> counts;
        int max_count = 0;

        for(const int & num : nums){
            max_count = max(max_count, ++counts[num]);
        }

        vector<vector<int>> buckets(max_count+1);
```

```
        for(const auto & p: counts){
            buckets[p.second].push_back(p.first);
        }

        vector<int> ans;
        for(int i = max_count; i>=0 && ans.size()<k; --i){
            for(const int & num: buckets[i]){
                ans.push_back(num);
            }
            if( ans.size() == k) break;
        }

        return ans;
    }
```

238. Product of Array Except Self

```
vector<int> productExceptSelf(vector<int>& nums) {
        // preproduct;;
        //O(n);

        int n = nums.size();
        vector<int> prefix(n);
        vector<int> postfix(n);
        vector<int> res(n);
        int pro = 1, post = 1;
        for (int i = 0; i < nums.size(); ++i) {
            pro *= nums[i];
            prefix[i] = pro;
        }

        for (int i = n - 1; i >= 0; --i) {
            post *= nums[i];
            postfix[i] = post;
        }
        res[0] = postfix[1];
        res[n-1] = prefix[n-2];
        for (int i = 1; i < n - 1; ++i) {
            int proEx = prefix[i - 1] * postfix[i+1];
            res[i] = proEx;

        }

        return res;
    }
```

128. Longest Consecutive Sequence

```
int longestConsecutive(vector<int>& nums) {
        // unordered_set <int> hash_set;
        // search in set find n+1 or n-1;
        // if find it, remove the element from array until element left
        // len = next - prev -1;
        //time O(n); space O(n);

        if (nums.size() == 0) return 0;
        unordered_set<int> hashSet(nums.begin(), nums.end()); // hash_set  O(n)
        int res = 0;
        for (int i = 0; i < nums.size(); ++i) { //O(n)
            int n = nums[i];
            if (hashSet.count(n)) {
                hashSet.erase(n);
                int prev = n - 1;
                int next = n + 1;
                while (hashSet.count(prev) > 0)  hashSet.erase(prev--);
                while (hashSet.count(next) > 0)  hashSet.erase(next++);
                res = max(res, next - prev - 1);
            }
        }

        return res;

    }
```

36. Valid Sudoku

```
bool isValidSudoku(vector<vector<char>>& board) {
        for (int i = 0; i < board.size(); ++i) {
            for (int j = 0; j < board[0].size(); ++j) {
                if (board[i][j] == '.') continue;
                else{
                    char val = board[i][j];
                    if (!isValid(i, j, val, board)) return false;

                }
            }
        }
        return true;

    }

private:
    bool isValid(int row, int col, char val, vector<vector<char>>& board) {
        for (int i = 0; i < board.size(); ++i) {
            if (i == col) continue;
            else if ( board[row][i] == val)
                return false;
```

```
        }
        for (int j = 0; j < board[0].size(); ++j) {
            if (j == row) continue;
            else if (board[j][col] == val)
                return false;
        }
        int startrow = (row / 3) * 3;
        int startcol = (col / 3) * 3;
        for(int i = startrow; i < startrow + 3; ++i) {
            for (int j = startcol; j < startcol + 3; ++j) {
                if (i == row && j == col) continue;
                else if (board[i][j] == val)
                    return false;
            }
        }
        return true;
    }
```