[**16. 3 Sum Closest**](https://leetcode.com/problems/3sum-closest/)

**Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to the target. Return the sum of the three integers. You may assume that each input would have exactly one solution.**

class Solution {

public:

int threeSumClosest(vector<int>& nums, int target) {

int closest= nums[0]+nums[1]+nums[2];

sort(nums.begin(),nums.end());

if(nums.size()<3)

{

return 0;

}

for(int i=0;i<nums.size();i++)

{

int j=i+1;

int k=nums.size()-1;

while(j<k){

int sum=nums[i]+nums[j]+nums[k];

if(sum==target)

{

j++;

k--;

}

if(abs(target-sum)<abs(target-closest))

{

closest=sum;

}

if(sum<target){

j++;

}

else{

k--;

}

}

}

return closest;

}

};

Time complexity : O(n^2)

Space complexity : O(1)

[**45. Jump Game II**](https://leetcode.com/problems/jump-game-ii/)

**You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].**

**Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:**

* **0 <= j <= nums[i] and**
* **i + j < n**

**Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].**

class Solution {

public:

int jump(vector<int>& nums) {

int n = nums.size();

if (n == 1) {

return 0;

}

int jumps = 0;

int far = 0;

int cur = 0;

for (int i = 0; i < n - 1; i++) {

far = max(far, i + nums[i]);

if (i == cur) {

jumps++;

cur = far;

if (cur >= n - 1) {

break;

}

}

}

return jumps;

}

};

Time complexity : O(n)

Space complexity : O(1)est Result

Test Result

[**49. Group Anagrams**](https://leetcode.com/problems/group-anagrams/)

**Given an array of strings strs, group the anagrams together. You can return the answer in any order.**

class Solution {

public:

string sign(string s){

map<char, int> mp;

for(auto i : s){

mp[i]++;

}

string ans = "";

for(auto i : mp){

ans += i.first;

ans += i.second;

}

return ans;

}

vector<vector<string>> groupAnagrams(vector<string>& strs) {

unordered\_map<string , vector<string>> mp;

for(auto i : strs){

mp[sign(i)].push\_back(i);

}

vector<vector<string>> ans;

for(auto i : mp) ans.push\_back(i.second);

return ans;

}

};

Time complexity : O(m\*n)

Space complexity : O(m\*n)

[**91. Decode Ways**](https://leetcode.com/problems/decode-ways/)

**You have intercepted a secret message encoded as a string of numbers. The message is decoded via the following mapping:**

**"1" -> 'A’**

**"2" -> 'B'**

**"25" -> 'Y'**

**"26" -> 'Z'**

**However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").**

**For example, "11106" can be decoded into:**

* **"AAJF" with the grouping (1, 1, 10, 6)**
* **"KJF" with the grouping (11, 10, 6)**
* **The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).**

**Note: there may be strings that are impossible to decode.  
  
Given a string s containing only digits, return the number of ways to decode it. If the entire string cannot be decoded in any valid way, return 0.**

**The test cases are generated so that the answer fits in a 32-bit integer.**

class Solution {

public:

int numDecodings(string s) {

vector<int> d(s.size()+1, 0);

string t = s.substr(0,2);

d[0] = (s[0] == '0' ? 0 : 1),

d[1] = (t[1] != '0' ? d[0] : 0) +

(stoi(t) <= 26 && t[0] != '0' ? 1 : 0);

for(int i = 2; i < s.size(); i++) {

t = s.substr(i-1,2);

d[i] = (t[1] != '0' ? d[i-1] : 0) +

(stoi(t) <= 26 && t[0] != '0' ? d[i-2] : 0);

}

return d[s.size()-1];

}

};

Time complexity : O(n)

Space complexity : O(n)

[**122. Best Time to Buy and Sell Stock II**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/)

**You are given an integer array prices where prices[i] is the price of a given stock on the ith day. On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.**

**Find and return the maximum profit you can achieve**

class Solution {

public:

int maxProfit(vector<int>& prices) {

int n = prices.size();

if (n == 1) return 0;

int l,r;

l = 0;

r = 1;

int profit = 0;

while (r < n){

if (prices[l] < prices[r]){

profit+= prices[r]-prices[l];

}

l++;

r++;

}

return profit;

}

};

Time complexity : O(n)

Space complexity : O(1)

[**200. Number of Islands**](https://leetcode.com/problems/number-of-islands/)

**Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.**

**An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.**

class Solution {

public:

int res = 0;

void dfs(int r, int c, vector<vector<int>>& newGrid)

{

newGrid[r][c] = -1;

if(newGrid[r + 1][c] == 1) dfs(r + 1, c, newGrid);

if(newGrid[r][c + 1] == 1) dfs(r, c + 1, newGrid);

if(newGrid[r - 1][c] == 1) dfs(r - 1, c, newGrid);

if(newGrid[r][c - 1] == 1) dfs(r, c - 1, newGrid);

return;

}

int numIslands(vector<vector<char>>& grid) {

int rows = grid.size();

int cols = grid[0].size();

vector<vector<int>> newGrid(rows + 2, vector<int> (cols + 2, 0));

for(int i = 1; i <= rows; i++)

for(int j = 1; j <= cols; j++)

newGrid[i][j] = grid[i-1][j-1] -'0';

for(int i = 1; i <= rows; i++)

{

for(int j = 1; j <= cols; j++)

{

if(newGrid[i][j] == 1)

{

dfs(i , j, newGrid);

res++;

}

}

}

return res;

}

};

Time complexity : O(r\*c)

Space complexity : O(r\*c)