class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

for i in range(len(nums)):

for j in range(i + 1, len(nums)):

if nums[j] == target - nums[i]:

return [i, j]

1 py3

class Solution {

public:

ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {

ListNode\* dummyHead = new ListNode(0);

ListNode\* tail = dummyHead;

int carry = 0;

while (l1 != nullptr || l2 != nullptr || carry != 0) {

int digit1 = (l1 != nullptr) ? l1->val : 0;

int digit2 = (l2 != nullptr) ? l2->val : 0;

int sum = digit1 + digit2 + carry;

int digit = sum % 10;

carry = sum / 10;

ListNode\* newNode = new ListNode(digit);

tail->next = newNode;

tail = tail->next;

l1 = (l1 != nullptr) ? l1->next : nullptr;

l2 = (l2 != nullptr) ? l2->next : nullptr;

}

ListNode\* result = dummyHead->next;

delete dummyHead;

return result;

}

};

2 c++

class Solution {

public:

int lengthOfLongestSubstring(string s) {

int n = s.length();

int maxLength = 0;

unordered\_set<char> charSet;

int left = 0;

for (int right = 0; right < n; right++) {

if (charSet.count(s[right]) == 0) {

charSet.insert(s[right]);

maxLength = max(maxLength, right - left + 1);

} else {

while (charSet.count(s[right])) {

charSet.erase(s[left]);

left++;

}

charSet.insert(s[right]);

}

}

return maxLength;

}

};

3 c++

class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

// Get the sizes of both input arrays.

int n = nums1.size();

int m = nums2.size();

// Merge the arrays into a single sorted array.

vector<int> merged;

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

merged.push\_back(nums1[i]);

}

for (int i = 0; i < m; i++) {

merged.push\_back(nums2[i]);

}

// Sort the merged array.

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

// Calculate the total number of elements in the merged array.

int total = merged.size();

if (total % 2 == 1) {

// If the total number of elements is odd, return the middle element as the median.

return static\_cast<double>(merged[total / 2]);

} else {

// If the total number of elements is even, calculate the average of the two middle elements as the median.

int middle1 = merged[total / 2 - 1];

int middle2 = merged[total / 2];

return (static\_cast<double>(middle1) + static\_cast<double>(middle2)) / 2.0;

}

}

};

4 c++

class Solution:

def longestPalindrome(self, s: str) -> str:

if len(s) <= 1:

return s

Max\_Len=1

Max\_Str=s[0]

for i in range(len(s)-1):

for j in range(i+1,len(s)):

if j-i+1 > Max\_Len and s[i:j+1] == s[i:j+1][::-1]:

Max\_Len = j-i+1

Max\_Str = s[i:j+1]

return Max\_Str

5 py3

class Solution {

public:

string convert(string s, int numRows) {

if(numRows <= 1) return s;

vector<string>v(numRows, "");

int j = 0, dir = -1;

for(int i = 0; i < s.length(); i++)

{

if(j == numRows - 1 || j == 0) dir \*= (-1);

v[j] += s[i];

if(dir == 1) j++;

else j--;

}

string res;

for(auto &it : v) res += it;

return res;

}

};

6 c++

class Solution {

public:

int reverse(int x) {

long r=0; // decleare r

while(x){

r=r\*10+x%10; // find remainder and add its to r

x=x/10; // Update the value of x

}

if(r>INT\_MAX || r<INT\_MIN) return 0; // check range if r is outside the range then return 0

return int(r); // if r in the 32 bit range then return r

}

};

7 c++

class Solution {

public:

int myAtoi(string s) {

int len = s.size();

double num = 0;

int i=0;

while(s[i] == ' '){

i++;

}

bool positive = s[i] == '+';

bool negative = s[i] == '-';

positive == true ? i++ : i;

negative == true ? i++ : i;

while(i < len && s[i] >= '0' && s[i] <= '9'){

num = num\*10 + (s[i]-'0');

i++;

}

num = negative ? -num : num;

cout<<num<<endl;

num = (num > INT\_MAX) ? INT\_MAX : num;

num = (num < INT\_MIN) ? INT\_MIN : num;

cout<<num<<endl;

return int(num);

}

};

8 c++

class Solution {

public:

bool isPalindrome(int x) {

if (x < 0) {

return false;

}

long long reversed = 0;

long long temp = x;

while (temp != 0) {

int digit = temp % 10;

reversed = reversed \* 10 + digit;

temp /= 10;

}

return (reversed == x);

}

};

9 c++

class Solution {

public boolean isMatch(String s, String p) {

Boolean dp[][] = new Boolean[s.length() + 1][p.length() + 1];// dp array to skip

return helper(s, 0, p, 0, dp);// overlapping subproblems

}

public static boolean helper(String s, int i, String p, int j, Boolean dp[][]){

if(i == s.length() && j == p.length()) return true;// if we have reached end of

//both strings means both string matches return true

if(j == p.length()) return false;// if j reached end of pattern, we cant compare more

if(dp[i][j] != null) return dp[i][j];// if dp doesn't contains null means we already calculated

boolean ans = false;// variable to store the answer

boolean charMatch = (i < s.length()) && (s.charAt(i) == p.charAt(j) || p.charAt(j) == '.');

// variable to pre compute is i not exceeding length is s or char at both string matches

// or pattern contains '.'

if(j + 1 < p.length() && p.charAt(j + 1) == '\*'){

ans = helper(s, i, p, j + 2, dp) || charMatch && helper(s, i + 1, p, j, dp);

}// first Condition -> j + 2 because if character repeats 0 times then same string s

// and pattern should start from j + 2

// Or if character matches or in pattern character at j is \* that means, Also charater matches

// so keep j at same position there maybe character repeats more than 0 times

// and then check from i + 1

else ans = charMatch && helper(s, i + 1, p, j + 1, dp);

return dp[i][j] = ans;

}

}

10 java

class Solution {

public:

int maxArea(vector<int>& height) {

int left = 0;

int right = height.size() - 1;

int maxArea = 0;

while (left < right) {

int currentArea = min(height[left], height[right]) \* (right - left);

maxArea = max(maxArea, currentArea);

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return maxArea;

}

};

11 c++

class Solution:

def intToRoman(self, num: int) -> str:

romans = {

1: 'I',

4: 'IV',

5: 'V',

9: 'IX',

10: 'X',

40: 'XL',

50: 'L',

90: 'XC',

100: 'C',

400: 'CD',

500: 'D',

900: 'CM',

1000: 'M'

}

sortedKeys = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]

result = ""

while num > 0:

print(num)

for key in sortedKeys:

if num >= key:

quo = num // key

result += romans[key] \* quo

num = num % key

return result

12 py3

class Solution:

def romanToInt(self, s: str) -> int:

translations = {

"I": 1,

"V": 5,

"X": 10,

"L": 50,

"C": 100,

"D": 500,

"M": 1000

}

number = 0

s = s.replace("IV", "IIII").replace("IX", "VIIII")

s = s.replace("XL", "XXXX").replace("XC", "LXXXX")

s = s.replace("CD", "CCCC").replace("CM", "DCCCC")

for char in s:

number += translations[char]

return number

13 py3

class Solution:

def longestCommonPrefix(self, v: List[str]) -> str:

ans=""

v=sorted(v)

first=v[0]

last=v[-1]

for i in range(min(len(first),len(last))):

if(first[i]!=last[i]):

return ans

ans+=first[i]

return ans

14 py3

class Solution {

public:

vector<vector<int>> threeSum(vector<int>& nums) {

int target = 0;

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

set<vector<int>> s;

vector<vector<int>> output;

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) {

s.insert({nums[i], nums[j], nums[k]});

j++;

k--;

} else if (sum < target) {

j++;

} else {

k--;

}

}

}

for(auto triplets : s)

output.push\_back(triplets);

return output;

}

};

15 c++

import java.util.Arrays;

class Solution {

public int threeSumClosest(int[] nums, int target) {

Arrays.sort(nums);

int closestSum = nums[0] + nums[1] + nums[2]; // Initialize closest sum with the sum of the first three elements

for (int i = 0; i < nums.length - 2; i++) {

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

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

if (Math.abs(target - sum) < Math.abs(target - closestSum)) {

closestSum = sum; // Update closest sum if the current sum is closer to the target

}

if (sum < target) {

j++; // Increment j to increase the sum

} else {

k--; // Decrement k to decrease the sum

}

}

}

return closestSum;

}

}

16 java

class Solution {

private:

void solve(string digit, string output, int index, vector<string>& ans, string mapping[] ) {

//base case

if(index >= digit.length()) {

ans.push\_back(output);

return;

}

int number = digit[index] - '0';

string value = mapping[number];

for(int i=0; i<value.length(); i++) {

output.push\_back(value[i]);

solve(digit, output, index+1, ans, mapping);

output.pop\_back();

}

}

public:

vector<string> letterCombinations(string digits) {

vector<string> ans;

if(digits.length()==0)

return ans;

string output;

int index = 0;

string mapping[10] = {"", "", "abc", "def", "ghi", "jkl","mno","pqrs","tuv","wxyz"};

solve(digits, output, index, ans, mapping);

return ans;

}

};

17 c++

class Solution {

public:

vector<vector<int> > fourSum(vector<int> &num, int target) {

vector<vector<int> > res;

if (num.empty())

return res;

std::sort(num.begin(),num.end());

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

long long target\_3 = (long long)target - (long long)num[i];

for (int j = i + 1; j < num.size(); j++) {

long long target\_2 = target\_3 - (long long)num[j];

int front = j + 1;

int back = num.size() - 1;

while(front < back) {

int two\_sum = num[front] + num[back];

if (two\_sum < target\_2) front++;

else if (two\_sum > target\_2) back--;

else {

vector<int> quadruplet(4, 0);

quadruplet[0] = num[i];

quadruplet[1] = num[j];

quadruplet[2] = num[front];

quadruplet[3] = num[back];

res.push\_back(quadruplet);

// Processing the duplicates of number 3

while (front < back && num[front] == quadruplet[2]) ++front;

// Processing the duplicates of number 4

while (front < back && num[back] == quadruplet[3]) --back;

}

}

// Processing the duplicates of number 2

while(j + 1 < num.size() && num[j + 1] == num[j]) ++j;

}

// Processing the duplicates of number 1

while (i + 1 < num.size() && num[i + 1] == num[i]) ++i;

}

return res;

}

};

18 c++

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

if(head==null||head.next==null)

return null;

ListNode slow = head;

ListNode fast=head;

int i=0,j=0;

while(fast.next!=null){

if(i>=n){

slow=slow.next;

}

fast=fast.next;

i++;

}

if(i+1 == n){

head = head.next;

return head;

}

slow.next=slow.next.next;

return head;

}

}

19 java

class Solution {

public:

bool isValid(string s) {

stack<char> st; //taking stack for keep tracking the order of the brackets..

for(auto i:s) //iterate over each and every elements

{

if(i=='(' or i=='{' or i=='[') st.push(i); //if current element of the string will be opening bracket then we will just simply push it into the stack

else //if control comes to else part, it means that current element is a closing bracket, so check two conditions current element matches with top of the stack and the stack must not be empty...

{

if(st.empty() or (st.top()=='(' and i!=')') or (st.top()=='{' and i!='}') or (st.top()=='[' and i!=']')) return false;

st.pop(); //if control reaches to that line, it means we have got the right pair of brackets, so just pop it.

}

}

return st.empty(); //at last, it may possible that we left something into the stack unpair so return checking stack is empty or not..

}

};

20 c++

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2)

{

// if list1 happen to be NULL

// we will simply return list2.

if(l1 == NULL)

{

return l2;

}

// if list2 happen to be NULL

// we will simply return list1.

if(l2 == NULL)

{

return l1;

}

// if value pointend by l1 pointer is less than equal to value pointed by l2 pointer

// we wall call recursively l1 -> next and whole l2 list.

if(l1 -> val <= l2 -> val)

{

l1 -> next = mergeTwoLists(l1 -> next, l2);

return l1;

}

// we will call recursive l1 whole list and l2 -> next

else

{

l2 -> next = mergeTwoLists(l1, l2 -> next);

return l2;

}

}

};

21 c++