1. 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;

    }

};

2. class Solution {

public:

    string intToRoman(int num) {

        string ones[] = {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"};

        string tens[] = {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"};

        string hundreds[] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"};

        string thousands[]= {"", "M", "MM", "MMM"};

        string Roman =  thousands[num / 1000] + hundreds[(num % 1000) / 100] + tens[(num % 100) / 10] + ones[num % 10];

        return Roman;

    }

};

3. class Solution {

public:

    int romanToInt(string s) {

        unordered\_map<char, int> m;

        m['I'] = 1;

        m['V'] = 5;

        m['X'] = 10;

        m['L'] = 50;

        m['C'] = 100;

        m['D'] = 500;

        m['M'] = 1000;

        int ans = 0;

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

            if(m[s[i]] < m[s[i+1]]){

                ans -= m[s[i]];

            }

            else{

                ans += m[s[i]];

            }

        }

        return ans;

    }

};

4. class Solution {

public:

    string longestCommonPrefix(vector<string>& v) {

        string ans="";

        sort(v.begin(),v.end());

        int n=v.size();

        string first=v[0],last=v[n-1];

        for(int i=0;i<min(first.size(),last.size());i++){

            if(first[i]!=last[i]){

                return ans;

            }

            ans+=first[i];

        }

        return ans;

    }

};

**5.** class Solution {

public:

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

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

        vector<vector<int>> result;

        auto first = nums.begin();

        do

        {

            if (first != nums.begin())

            {

                first = upper\_bound(first, nums.end(), \*prev(first, 1));

            }

            auto second = next(first, 1);

            auto third = prev(nums.end(), 1);

            while (second < third)

            {

                const int sum = \*first + \*second + \*third;

                if (sum == 0)

                {

                    result.push\_back({\*first, \*second, \*third});

                    second = prev(upper\_bound(second, prev(third, 1), \*second), 1);

                    third = lower\_bound(next(second, 1), third, \*third);

                    advance(second, 1);

                    advance(third, -1);

                }

                else if (sum < 0)

                {

                    advance(second, 1);

                }

                else

                {

                    advance(third, -1);

                }

            }

            advance(first, 1);

        }

        while (first < prev(nums.end(), 2));

        return result;

    }

};

**6.** class Solution {

public:

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

        int n=nums.size();

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

        int diff=INT\_MAX;

        int ans;

        for(int i=0;i<n;i++)

        {

            int s=i+1;

            int e=n-1;

            while(s<e)

            {

                if(nums[i]+nums[s]+nums[e]==target)

                    return target;

                 else if(abs(nums[i]+nums[s]+nums[e]-target)<diff)

                 {

                    diff=abs(nums[i]+nums[s]+nums[e]-target);

                    ans=nums[i]+nums[s]+nums[e];

                 }

                 if(nums[i]+nums[s]+nums[e]<target)

                    s++;

                  else if(nums[i]+nums[s]+nums[e]>target)

                        e--;

            }

        }

        return ans;

    }

};

7. class Solution {

public:

    vector<string> letterCombinations(string digits) {

        if (digits.empty()) {

            return {};

        }

        // Mapping of digits to corresponding characters

        unordered\_map<char, string> mp = {

            {'2', "abc"},

            {'3', "def"},

            {'4', "ghi"},

            {'5', "jkl"},

            {'6', "mno"},

            {'7', "pqrs"},

            {'8', "tuv"},

            {'9', "wxyz"}

        };

        string cur = ""; // Current combination

        vector<string> result; // Resulting combinations

        // DFS function to generate combinations

        dfs(digits, 0, mp, cur, result);

        return result;

    }

private:

    // DFS function

    void dfs(string digits, int index, unordered\_map<char, string>& mp, string& cur, vector<string>& result) {

        // Base case: if we've reached the end of the digits string

        if (index == digits.size()) {

            // Add the current combination to the result

            result.push\_back(cur);

            return;

        }

        // Get the characters corresponding to the current digit

        string str = mp[digits[index]];

        // Iterate through each character and explore all possible combinations

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

            // Append the current character to the combination

            cur.push\_back(str[i]);

            // Explore the next digit by recursively calling DFS

            dfs(digits, index + 1, mp, cur, result);

            // Backtrack: remove the last character to explore other combinations

            cur.pop\_back();

        }

    }

};

8. class Solution {

public:

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

        int n=nums.size();

        vector<vector<int>> ans;

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

        for(int i=0;i<n;i++){

            if(i!=0  && nums[i]==nums[i-1]) {

                continue;

            }

            for(int j=i+1;j<n;j++){

            if(j!=i+1 && nums[j]==nums[j-1]) {

                continue;

            }

                int k=j+1;

                int l=n-1;

                while(k<l){

                    long long sum=nums[i];

                    sum+=nums[j];

                    sum+=nums[k];

                    sum+=nums[l];

                    if(sum==target){

                        vector<int>temp={nums[i],nums[j],nums[k],nums[l]};

                        ans.push\_back(temp);

                        k++;

                        l--;

                        while(k<l&& nums[k]==nums[k-1]){

                            k++;

                        }

                        while(k<l&& nums[l]==nums[l+1]){

                            l--;

                        }

                    }

                    else if(sum<target){

                        k++;

                    }else{

                        l--;

                    }

                }

            }

        }

        return ans;

    }

};

9. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

int getLength(ListNode\* &head){

ListNode\* temp = head;

int len = 0;

while(temp != NULL){

len++;

temp = temp->next;

}

return len;

}

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

int len = getLength(head);

int pos = len - n;

int count = 0;

ListNode\* curr = head;

ListNode\* prev = head;

if(pos == 0){

head = head->next;

return head;

}

while(curr != NULL){

if(count == pos){

prev->next = curr->next;

curr->next = NULL;

break;

}

else{

prev = curr;

curr = curr->next;

count++;

}

}

return head;

}

};

10.

class Solution {

public:

bool isValid(string s) {

stack<char> st; // create an empty stack to store opening brackets

for (char c : s) { // loop through each character in the string

if (c == '(' || c == '{' || c == '[') { // if the character is an opening bracket

st.push(c); // push it onto the stack

} else { // if the character is a closing bracket

if (st.empty() || // if the stack is empty or

(c == ')' && st.top() != '(') || // the closing bracket doesn't match the corresponding opening bracket at the top of the stack

(c == '}' && st.top() != '{') ||

(c == ']' && st.top() != '[')) {

return false; // the string is not valid, so return false

}

st.pop(); // otherwise, pop the opening bracket from the stack

}

}

return st.empty(); // if the stack is empty, all opening brackets have been matched with their corresponding closing brackets,

// so the string is valid, otherwise, there are unmatched opening brackets, so return false

}

};