**1.Valid palindrome**

#include <string>

#include <cctype>

#include <algorithm>

using namespace std;

bool isPalindrome(string s) {

string res = "";

for (char i : s) {

if (isalnum(i)) {

res += tolower(i);

}

}

string reversedRes = res;

reverse(reversedRes.begin(), reversedRes.end());

return res == reversedRes;

}

**2.Is Subsequence**

#include <string>

using namespace std;

class Solution {

public:

bool isSubsequence(string s, string t) {

for (char i : s) {

int ind = t.find(i);

if (ind != -1) {

t = t.substr(ind + 1);

} else {

return false;

}

}

return true;

}

};

**3.Two sum**

for (int i=0;i<nums.size()-1;i++){

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

if (nums[j]==target-nums[i])

return [i,j];

}

}

**4.Container with most water**

Int l=0;

Int tot=0;

Int r=height.size()-1;

while (l<r){

tot=max(tot,min(height[l],height[r])\*(r-l));

if (height[l]<=height[r])

l+=1;

elif (height[r]<height[l])

r-=1;

}

return tot;

**5. 3sum**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

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

vector<vector<int>> result;

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

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

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

continue;

}

int j = i + 1;

int k = nums.size() - 1;

while (j < k) {

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

if (sum < 0) {

j++;

} else if (sum == 0) {

result.push\_back({nums[i], nums[j], nums[k]});

j++;

k--;

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

j++;

}

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

k--;

}

} else {

k--;

}

}

}

return result;

}

**6.Valid Parenthesis**

#include <stack>

#include <string>

using namespace std;

class Solution {

public:

bool isValid(string s) {

stack<char> stack;

for (char c : s) {

if (c == '{' || c == '(' || c == '[') {

stack.push(c);

} else {

if (stack.empty()) {

return false;

} else {

if ((c == '}' && stack.top() == '{') ||

(c == ')' && stack.top() == '(') ||

(c == ']' && stack.top() == '[')) {

stack.pop();

} else {

return false;

}

}

}

}

return stack.empty();

}

};

**7.simplify path**

#include <string>

#include <vector>

#include <sstream>

using namespace std;

class Solution {

public:

string simplifyPath(string path) {

vector<string> s;

string curr = "";

path += "/";

for (char c : path) {

if (c == '/') {

if (curr == "..") {

if (!s.empty()) {

s.pop\_back();

}

} else if (!curr.empty() && curr != ".") {

s.push\_back(curr);

}

curr = "";

} else {

curr += c;

}

}

string res = "/";

for (size\_t i = 0; i < s.size(); ++i) {

res += s[i];

if (i != s.size() - 1) {

res += "/";

}

}

return res;

}

};

**8.min stack**

#include <stack>

#include <climits>

class MinStack {

private:

stack<int> s;

stack<int> minStack;

public:

MinStack() {

}

void push(int val) {

s.push(val);

if (minStack.empty() || val <= minStack.top()) {

minStack.push(val);

}

}

void pop() {

if (!s.empty()) {

if (s.top() == minStack.top()) {

minStack.pop();

}

s.pop();

}

}

int top() {

if (!s.empty()) {

return s.top();

}

return -1;

}

int getMin() {

if (!minStack.empty()) {

return minStack.top();

}

return INT\_MAX;

}

};

**9.search insert position**

class Solution {

public:

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

int l = 0, h = nums.size() - 1;

if (target > nums.back()) {

return nums.size();

}

if (target < nums.front()) {

return 0;

}

while (l <= h) {

int m = l + (h - l) / 2;

if (nums[m] == target) {

return m;

} else if (nums[m] > target) {

if (l == h) {

return l + 1;

} else {

h = m - 1;

}

} else {

if (l == h) {

return l + 1;

} else {

l = m + 1;

}

}

}

return l;

}

};

**10.Search a 2D matrix:**

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int maxElement = \*max\_element(nums.begin(), nums.end());

return distance(nums.begin(), find(nums.begin(), nums.end(), maxElement));

}

};

**11.find Peak element**

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int maxElement = \*max\_element(nums.begin(), nums.end());

return distance(nums.begin(), find(nums.begin(), nums.end(), maxElement));

}

};  
 **12.search in rotated sorted array**

class Solution {

public:

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

auto it = find(nums.begin(), nums.end(), target);

if (it != nums.end()) {

return distance(nums.begin(), it);

}

return -1;

}

};