[**125. Valid Palindrome**](https://leetcode.com/problems/valid-palindrome/)

**A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.**

**Given a string s, return true if it is a palindrome, or false otherwise.**

class Solution {

public:

bool isPal(string s)

{

string origin=s;

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

return origin==s;

}

void lower(string &s)

{

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

s[i]=tolower(s[i]);

}

bool isPalindrome(string s) {

string pal;

lower(s);

for(char i:s)

{

if(isdigit(i) || isalpha(i))

pal+=i;

}

if(isPal(pal)) return true;

else return false;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**392. Is Subsequence**](https://leetcode.com/problems/is-subsequence/)

**Given two strings s and t, return true if s is a subsequence of t, or false otherwise.**

**A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).**

class Solution {

public:

bool isSubsequence(string s, string t) {

bool ans=0;

int sl=s.length(),tl=t.length();

if(sl==0)return 1;

if(sl>tl) return 0;

int i=0,j=0;

while(i<sl && j<tl){

if(s[i]!=t[j]){

j++;

}

else{

i++;

j++;

}

}

if(i!=sl) return 0;

return 1;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**167. Two Sum II - Input Array Is Sorted**](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/)

**Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.**

**Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.**

**The tests are generated such that there is exactly one solution. You may not use the same element twice.**

**Your solution must use only constant extra space.**

class Solution {

public:

vector<int> twoSum(vector<int>& numbers, int target) {

int first ;

int second ;

int j = numbers.size()-1;

cout<<j<<endl;

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

int sum = numbers[i]+numbers[j];

if(sum>target){

j--;

}

else if(sum<target){

i++;

}

else{

first = i;

second = j;

break;

}

}

vector<int> v;

v.push\_back(first+1);

v.push\_back(second+1);

return v;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**11. Container With Most Water**](https://leetcode.com/problems/container-with-most-water/)

**You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).**

**Find two lines that together with the x-axis form a container, such that the container contains the most water.**

**Return the maximum amount of water a container can store.**

**Notice that you may not slant the container.**

class Solution {

public:

int maxArea(vector<int>& height) {

int l=0,r=height.size()-1;

int arr = 0;

while (l < r){

int ll = height[l];

int rr = height[r];

int area = min(ll,rr)\*(r-l);

arr = max(area,arr);

if (ll < rr){

l++;

}

else{r--;}

}

return arr;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**15. 3Sum**](https://leetcode.com/problems/3sum/)

**Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.**

**Notice that the solution set must not contain duplicate triplets.**

class Solution {

public:

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

vector<vector<int>> ans;

int n = nums.size();

set<vector<int>> st;

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

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

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

int j = i+1;

int k = n-1;

while(j<k){

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

if(sum < 0){

j++;

}

else if(sum >0 ){

k--;

}

else{

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

j++;

k--;

}

}

}

for(const vector<int> pair : st){

ans.push\_back(pair);

}

return ans;

}

};

**Time complexity : O(N^2)**

**Space complexity : O(K)**

[**209. Minimum Size Subarray Sum**](https://leetcode.com/problems/minimum-size-subarray-sum/)

**Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.**

class Solution {

public:

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

int n=nums.size();

int i=0,j=1;

int curr=nums[0],ans=INT\_MAX;

while(j<=n){

if(curr>=target){

ans=min(ans,j-i);

curr-=nums[i];

i++;

}

else{

if(j==n)

break;

curr+=nums[j];

j++;

}

}

return ans!=INT\_MAX ? ans : 0;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**3. Longest Substring Without Repeating Characters**](https://leetcode.com/problems/longest-substring-without-repeating-characters/)

**Given a string s, find the length of the longest substring without repeating characters.**

class Solution {

public:

int lengthOfLongestSubstring(string s) {

int n=s.length();

int maxlen=0;

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

int hash[256]={0};

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

if(hash[s[j]]==1){

break;

}

else{

hash[s[j]]=1;

}

int len=j-i+1;

maxlen=max(len,maxlen);

}

}

return maxlen;

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**76. Minimum Window Substring**](https://leetcode.com/problems/minimum-window-substring/)

**Given two strings s and t of lengths m and n respectively, return the minimum window**

**substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "". The testcases will be generated such that the answer is unique.**

class Solution {

public:

string minWindow(string s, string t) {

int m = s.size();

int n = t.size();

if(n>m)

return "";

unordered\_map<char, int>mp;

for(auto ch: t)

mp[ch]++;

int count = mp.size();

int i=0;

int j=0;

int l1=-1;

int r1 =-1;

int ans = INT\_MAX;

while(j<s.size())

{

if(mp.count(s[j])) {

mp[s[j]]--;

if(mp[s[j]]==0)

count--;

}

j++;

while (count==0 and i<j) {

if(ans>(j-i))

{

ans = j-i;

l1=i;

r1=j;

}

if(mp.count(s[i])) {

mp[s[i]]++;

if(mp[s[i]]> 0)

count++;

}

i++;

}

}

if(l1==-1 and r1==-1)

return "";

return s.substr(l1,r1-l1);

}

};

**Time complexity : O(M)**

**Space complexity : O(N)**

[**20. Valid Parentheses**](https://leetcode.com/problems/valid-parentheses/)

**Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.**

**An input string is valid if:**

1. **Open brackets must be closed by the same type of brackets.**
2. **Open brackets must be closed in the correct order.**
3. **Every close bracket has a corresponding open bracket of the same type.**

class Solution {

public:

bool isValid(string s) {

stack<char> st;

for(char ch:s){

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

st.push(ch);

}

else if(st.empty()){

return false;

}

else if(st.top() == '(' && ch == ')'){

st.pop();

}

else if(st.top() == '{' && ch == '}'){

st.pop();

}

else if(st.top() == '[' && ch == ']'){

st.pop();

}

else{

return false;

}

}

return st.empty();

}

};

**Time complexity : O(N)**

**Space complexity : O(1)**

[**71. Simplify Path**](https://leetcode.com/problems/simplify-path/)

**You are given an absolute path for a Unix-style file system, which always begins with a slash '/'. Your task is to transform this absolute path into its simplified canonical path.**

**The rules of a Unix-style file system are as follows:**

* **A single period '.' represents the current directory.**
* **A double period '..' represents the previous/parent directory.**
* **Multiple consecutive slashes such as '//' and '///' are treated as a single slash '/'.**
* **Any sequence of periods that does not match the rules above should be treated as a valid directory or file name. For example, '...' and '....' are valid directory or file names.**

**The simplified canonical path should follow these rules:**

* **The path must start with a single slash '/'.**
* **Directories within the path must be separated by exactly one slash '/'.**
* **The path must not end with a slash '/', unless it is the root directory.**
* **The path must not have any single or double periods ('.' and '..') used to denote current or parent directories.**

**Return the simplified canonical path.**

class Solution {

public:

string simplifyPath(string path) {

stack<string> st;

int i = 0;

string cur = "";

path += '/';

while (i < path.size()) {

if (path[i] == '/') {

if (!cur.empty()) {

if (cur == ".") {

}

else if (cur == "..") {

if (!st.empty()) {

st.pop();

}

}

else {

st.push(cur);

}

}

cur.clear();

i += 1;

}

else {

cur += path[i];

i += 1;

}

}

string rst = "";

while (!st.empty()) {

rst = "/" + st.top() + rst;

st.pop();

}

if (rst.empty()) rst += '/';

return rst;

}

};

**Time complexity : O(N)**

**Space complexity : O(M)**