**1.Floor in sorted array**  
class solution{  
public:

int findFloor(vector<int>& arr, int k) {

int l=0;

int h=arr.size()-1;

int res=-1;

while (l<=h){

int mid=(l+h)/2;

if (arr[mid]<=k)

{

res=mid;

l=mid+1;

}

else

h=mid-1;

}

return res;

}

};

**2.Check Equal Arrays**  
class solution{  
public:

bool check(vector<int>& arr1, vector<int>& arr2) {

sort(arr1.begin(),arr1.end());

sort(arr2.begin(),arr2.end());

return arr1==arr2;

}};

**3.Palindrome linkedlist**

/\*\*

\* 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:

bool isPalindrome(ListNode\* head) {

stack<int> s;

ListNode\* curr=head;

while (curr){

s.push(curr->val);

curr=curr->next;

}

curr=head;

while (curr && curr->val==s.top()) {

s.pop();

curr=curr->next;

}

return curr == nullptr;

}

};

**3.Balanced Binary Tree**  
/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

bool isBalanced(TreeNode\* root) {

return (height(root)>=0);

}

int height(TreeNode\* root){

if (root==NULL)

return 0;

int lh=height(root->left);

int rh=height(root->right);

if (lh<0 || rh<0 || abs(lh-rh)>1)

return -1;

return max(lh,rh)+1;

}

};

**4. Triplet sum**

class solution{

public:

bool hasTripletSum(vector<int> &arr, int target) {

int n=arr.size();

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

for (int i=0;i<n-2;i++){

int l=i+1;

int r=n-1;

while (l<r){

int sum=arr[i]+arr[l]+arr[r];

if (sum>target)

r--;

else if (sum<target)

l++;

else

return true;

}

}

return false;

}

};

**5. Row with max 1’s**

Class solution {

public:

int rowWithMax1s(vector<vector<int>> &arr) {

int row=arr.size();

int col=arr[0].size();

int m=0;

int ind=-1;

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

int c=0;

for (int j=0;j<col;j++){

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

c+=1;

}

if (c>m){

m=c;

ind=i;

}

}

return ind;

}

};

**6.Anagram**  
#include <bits/stdc++.h>

using namespace std;

int main() {

string s1="allergy";

string s2="allergic";

sort(s1.begin(),s1.end());

sort(s2.begin(),s2.end());

if (s1==s2)

cout<<"True";

else

cout<<"False";

}

**7.Longest Palindrome**  
#include <iostream>

#include <string>

using namespace std;

string expand(string s,int l,int r){

while (l>=0 && r<s.size() && s[l]==s[r]){

l--;

r++;

}

return s.substr(l+1,r-l-1);

}

int main() {

string s="abc";

string lo="";

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

string odd=expand(s,i,i);

if (odd.size()>lo.size()){

lo=odd;

}

string even=expand(s,i,i+1);

if (even.size()>lo.size()){

lo=even;

}

}

cout<<lo;}

**8.longest consecutive subsequence**   
class Solution {

public:

int longestConsecutive(vector<int>& nums) {

if (nums.empty()) return 0;

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

vector<int> uniqueNums;

for (int num : nums) {

if (uniqueNums.empty() || uniqueNums.back() != num) {

uniqueNums.push\_back(num);

}

}

int maxLen = 0;

int left = 0;

for (int right = 0; right < uniqueNums.size() - 1; right++) {

if (uniqueNums[right] + 1 != uniqueNums[right + 1]) {

left = right + 1;

}

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

}

return maxLen + 1;

}

};