Soln1.

Leetcode Assignment Accepted Solutions!!

// TC: O(n), SC: O(n), where n is size of array nums

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

vector<int> ans;

unordered\_map<int, int> mp;

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

{

if(mp.find(target - nums[i]) != mp.end())

{

ans.push\_back(mp[target - nums[i]]);

ans.push\_back(i);

break;

}

mp[nums[i]] = i;

}

return ans;

}

Soln 2:

// SC: O(log10(n)), TC: O(log10(n)), where n is the number

bool isPalindrome(int x)

{

string str = to\_string(x);

string rev = str;

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

return rev == str;

}

Soln3:

// using indexes to solve, actually if we can keep removing the correct matches and storing only the wrong parentheses index, then at the end, in between those wrong indexes we have correct parentheses substring, hence simply calculate at the end.

int longestValidParentheses(string s) {

if(s.empty() || s.length() == 1)

return 0;

stack<int> st;

//storing the index where you got invalid ) , as in middle of )s there is valid string

// just subtract between invalid indexes to get maximum lenth valid substring

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

{

//if opening

if(s[i] == '(')

{

st.push(i);

}

else if(s[i] == ')' && !st.empty() && s[st.top()] == '(')

{

st.pop();

}

else

st.push(i);

}

if(st.empty()) //if whole string is valid, then stack should be empty

return (int)s.size();

int maxi = 0;

int endInvalid = s.size();

while(!st.empty())

{

int startInvalid = st.top();

st.pop();

maxi = max(maxi, endInvalid - startInvalid - 1);

endInvalid = startInvalid;

}

maxi = max(maxi, endInvalid);

return maxi;

}

Soln4:

string longestPalindrome(string s) {

int n = s.size();

vector<vector<int>> dp(n, vector<int>(n, 0));

int st = -1;

int len = 0;

for(int diag = 0; diag < n; diag++)

{

for(int i = 0, j = diag; j < n; i++, j++)

{

if(diag == 0)

dp[i][j] = 1;

else if(diag == 1)

dp[i][j] = (s[i] == s[j]);

else if(s[i] == s[j])

dp[i][j] = dp[i + 1][j - 1];

if(dp[i][j] && j - i + 1 > len)

{

len = diag + 1;

st = i;

}

}

}

return s.substr(st, len);

}