**Time Complexity from low to high**:

* 0(1)
* 0(logn)
* 0(n)
* 0(nlogn)
* 0(n^2)
* 0(n^2 logn)

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

{

// n times

// TC = 0(n)

}

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

{

// n/2 times

// TC = 0(n)

}

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

{

// n times

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

{

// n times

// TC = 0(n^2)

}

}

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

{

// n/2 times

for (int j=0; j<n; j=j+2)

{

// n/2 times

// TC = 0(n^2)

}

}

for (int i=1; i<n; i\*2)

{

// log2n times

// exponential by 2

}

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

{

// n times

for (int j=0; j<n; j\*2)

{

// log2 times

// TC = 0(n log2n)

}

}

Int max\_count = 0;

Int max\_count\_ele = int\_min;

Vector<int> arr = {1,2,3,4,2,3,3,2,2,2};

Map<int, int> mp;

Int n = arr.size();

For i=0 … n

Mp[arr[i]++];

For (auto p : mp){

If (p.second > max\_count){

}

}

#include <iostream>

#include<bits/stdc++.h>

#include <vector>

using namespace std;

int sumOfN\_number(int n) {

if (n == 0)

return n;

else

return n + sumOfN\_number(n - 1);

}

int printN\_Time(int val, int n){

if (n==1){

return 1;

}

return printN\_Time(val, n-1);

}

// bool isPal(string s)

// {

// int len = s.length();

// int left = 0;

// int right = len-1;

// while(left<=right){

// if (s[left++] != s[right--]){

// return false;

// }

// }

// return true;

// }

// bool isPalUsingRec(string s, int left, int right){

// if (left >= right)

// return true;

// if (s[left] != s[right]) return false;

// return isPalUsingRec(s, left++, right--);

// }

bool isPalUsingRec(char str[], int s, int e) {

if (s == e)

return true;

if (str[s] != str[e])

return false;

if (s < e + 1)

return isPalUsingRec(str, s + 1, e - 1);

return true;

}

bool isPalindrome(char str[]) {

int n = strlen(str);

if (n == 0)

return true;

return isPalUsingRec(str, 0, n - 1);

}

// Recursive function to reverse an array

void revOfArr(int i,int arr[], int n){

if(i>=n/2)

return;

swap(arr[i],arr[n-1-i]);

revOfArr(i+1,arr,n);

}

// Function to find the largest number in an array

int largest(int arr[], int n) {

int max = arr[0]; // Initialize max with the first element

// Traverse array elements from the second and compare

// every element with the current max

for (int i = 1; i < n; i++) {

if (arr[i] > max) {

max = arr[i]; // Update max if a greater element is found

}

}

return max;

}

// bruce code

int mostFrequent(int\* arr, int n) {

int maxcount = 0;

int element\_having\_max\_freq;

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

int count = 0;

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

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

count++;

}

if (count > maxcount) {

maxcount = count;

element\_having\_max\_freq = arr[i];

}

}

return element\_having\_max\_freq;

}

// sorting approach

int mostFrequentBySorting(int arr[], int n) {

sort(arr, arr + n);

int max\_count = 1, res = arr[0], curr\_count = 1;

for (int i = 1; i < n; i++) {

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

curr\_count++;

else

curr\_count = 1;

if (curr\_count > max\_count) {

max\_count = curr\_count;

res = arr[i - 1];

}

}

return res;

}

// hashing

int mostFrequentByHashing(int arr[], int n) {

unordered\_map<int, int> hash;

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

hash[arr[i]]++;

int max\_count = 0, res = -1;

for (auto i : hash) {

if (max\_count < i.second) {

res = i.first;

max\_count = i.second;

}

}

return res;

}

// Function to find the largest number in array

int largestEle(int arr[], int n)

{

int i;

int max = arr[0];

for (i = 1; i < n; i++)

if (arr[i] > max)

max = arr[i];

return max;

}

// red red green blue green ----> red red green green blue

void RGB(std::vector<char>& colors) {

int low = 0;

int mid = 0;

int high = colors.size() - 1;

while (mid <= high) {

switch (colors[mid]) {

case 'R':

swap(colors[low], colors[mid]);

++low;

++mid;

break;

case 'G':

++mid;

break;

case 'B':

swap(colors[mid], colors[high]);

--high;

break;

}

}

}

vector<int> find\_union(vector<int> a1, vector<int> a2){

vector<int> res;

set<int> s1;

for(int x:a1)

s1.insert(x);

for(int x:a2)

s1.insert(x);

for(int x:s1)

res.push\_back(x);

return res;

}

vector<int> find\_union1(vector<int> a1, vector<int> a2){

{

vector<int> res;

int i=0, j=0, k=-1;

int n1 = a1.size();

int n2 = a2.size();

while(i<n1 && j<n2){

if(a1[i]<a2[j]){

if (res[k]!=a1[i])

res[++k] = a1[i];

i++;

}else{

if (res[k]!=a2[j])

res[++k] = a2[j];

j++;

}

}

while(i<n1){

res[k++] = a1[i++];

}

while(j<n2){

res[k++] = a2[j++];

}

}

int main()

{

vector<int> a1 = {1,2,4,3,9};

vector<int> a2 = {7,8,99};

// int n;

// int arr[] = [1,2,3,4], n=4;

// cout<<"Enter n numbers to sum";

// cin>>n;

// cout<<sumOfN\_number(n);

// cout<<isPalUsingRec("abba", 0, 3);

// char str[] = "geeg";

// if (isPalindrome(str))

// cout << "Yes"<<endl;

// else

// cout << "No"<<endl;

// int arr[]={5,4,8,3,6,1};

// revOfArr(0,arr,6);

// for(int i=0;i<6;i++)

// {

// cout<<arr[i]<<" ";

// }

// int arr[] = {40, 50, 30, 40, 50, 30, 30};

// int n = sizeof(arr) / sizeof(arr[0]);

// cout << mostFrequent(arr, n);

// cout << mostFrequentByHashing(arr, n);

// cout << mostFrequentBySorting(arr, n);

// int arr[] = { 10, 324, 45, 90, 9808 };

// int n = sizeof(arr) / sizeof(arr[0]);

// cout << "Largest in given array is " << largestEle(arr, n);

// calling RGB

// vector<char> colors = {'R','G', 'R', 'G', 'B', 'R', 'R', 'B', 'G', 'G', 'R'};

// RGB(colors);

// for (char c : colors) {

// cout << c << " ";

// }

return 0;

}

// Online C++ compiler to run C++ program online

#include <iostream>

#include <vector>

#include<bits/stdc++.h>

using namespace std;

// find unique element through brute force

void getUniqueElementUsingBruteForce(){

// brute

int arr[10] = {1,2,3,1,2,3,4,5,4,4};

// int n = sizeof(arr) / sizeof(arr[0]);

int n = 10,i=0;

while(i<n){

int count =0;

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

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

count+=1;

}

}

if(count==1){

cout<<arr[i]<<endl;

}

i++;

}

}

int getUniqueElementUsingX0R(int arr[]){

int n = 6;

int res = 0;

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

res = res ^ arr[i]; // XOR operator returns only unique values

}

return res;

}

int maxAlt(vector<int>& gain)

{

int currentAlt = 0;

int maxAlt = 0;

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

currentAlt += gain[i];

maxAlt = max(maxAlt, currentAlt);

}

return maxAlt;

}

// 238 prblm leetcode

vector<int> prod\_except\_self(vector<int> v)

{

int n = v.size();

vector<int> prefix\_prod(n);

vector<int> suffix\_prod(n);

vector<int> result(n);

prefix\_prod[0] = 1; suffix\_prod[n-1] = 1;

for(int i=1; i<n; i++)

prefix\_prod[i] = prefix\_prod[i-1] \* v[i-1];

for(int i=n-2; i>=0; i--)

suffix\_prod[i] = suffix\_prod[i+1] \* v[i+1];

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

result[i] = prefix\_prod[i] \* suffix\_prod[i];

return result;

}

// 525

int max\_contiguous(vector<int> v){

// vector<int> v = {1,1,0,0,1,0,1};

int n = v.size();

for(int i=1; i<n; i++){

if(v[i]==0){

v[i]=-1;

}

}

int max\_len = 0;

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

int sum=0;

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

sum+=v[j];

if(sum==0){

int current\_len = j-i+1;

max\_len = max(current\_len, max\_len);

}

}

}

return max\_len;

}

// 525 prblm solve by unordered\_map

int findMaxLength(vector<int>& nums) {

unordered\_map<int, int> sumToIndex;

int cumulativeSum = 0;

int maxLength = 0;

// Initialize the map with sum 0 at index -1

sumToIndex[0] = -1;

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

// Convert 0 to -1

if (nums[i] == 0)

cumulativeSum -= 1;

else

cumulativeSum += 1;

// Check if cumulative sum exists in the map

if (sumToIndex.find(cumulativeSum) != sumToIndex.end()) {

// Update maximum length

maxLength = max(maxLength, i - sumToIndex[cumulativeSum]);

} else {

// Store cumulative sum and its index

sumToIndex[cumulativeSum] = i;

}

}

return maxLength;

}

int longestUniqueSubsttr(string str) {

int n = str.size();

int res = 0; // result

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

vector<bool> visited(256, false);

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

if (visited[str[j]])

break;

else {

res = max(res, j - i + 1);

visited[str[j]] = true;

}

}

visited[str[i]] = false; // Remove the first character of previous window

}

return res;

}

int main() {

// Write C++ code here

// int arr[] = {2,3,4,2,3};

// getUniqueElementUsingBruteForce();

// cout<<getUniqueElementUsingX0R(arr);

// int arr[] = {2,3,4,5,5,7};

// int i, dupe = 0;

// for (i = 0; i < 5; i++) {

// dupe = dupe ^ arr[i] ^ i;

// }

// cout<<dupe<<endl;

// vector<int> gain = {-5, 1, 5, 0, -7};

// int result = maxAlt(gain);

// cout << "Maximum altitude gain: " << result << endl;

// call prod\_except\_self

// vector<int> in = {1,2,3,4};

// vector<int> res = prod\_except\_self(in);

// for(auto x : res)

// cout<<x<<" ";

// vector<int> v = {1,1,0,0,1,0,1};

// // for (auto a : )

// cout<<max\_contiguous(v)<<" ";

vector<int> nums = {0, 1, 0};

cout << "Maximum length of contiguous subarray: " << findMaxLength(nums) << endl;

string str = "geeksforgeeks";

cout << "The input string is " << str << endl;

int len = longestUniqueSubsttr(str);

cout << "The length of the longest non-repeating "

<< "character substring is " << len;

return 0;

}

**Symmetric pair in array**

**void** findSymPairs(**int** arr[][2], **int** row)

{

    // This loop for selection of one pair

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

      // This loop for searching of symmetric pair

**for** (**int** j = i + 1; j < row; j++) {

            // Condition of symmetric pair

**if** (arr[i][0] == arr[j][1]

                and arr[i][1] == arr[j][0])

            {

                cout << "(" << arr[i][0] << ", "

                     << arr[i][1] << ")" << endl;

            }

        }

    }

O(n^2)

**void** findSymPairs(**int** arr[][2], **int** row)

{

    // Creates an empty hashMap hM

    unordered\_map<**int**, **int**> hM;

    // Traverse through the given array

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

    {

        // First and second elements of current pair

**int** first = arr[i][0];

**int** sec   = arr[i][1];

        // If found and value in hash matches with first

        // element of this pair, we found symmetry

**if** (hM.find(sec) != hM.end() && hM[sec] == first)

            cout << "(" << sec << ", " << first << ")" <<endl;

**else**  // Else put sec element of this pair in hash

            hM[first] = sec;

    }

}

O(n)