**C++ Problems:**

1. **Maximum Average subarray 1:(Sliding Window)**
2. class Solution {
3. public:
4. double findMaxAverage(vector<int>& nums, int k) {
5. double windowsum = 0;
6. for(int i=0; i<k; i++){
7. windowsum += nums[i];
8. }
9. double maxsum = windowsum;
10. for(int i=k; i<nums.size(); i++){
11. windowsum =windowsum + nums[i] - nums[i - k];
12. maxsum = max(maxsum, windowsum);
13. }
14. // cout << windowsum;
15. return maxsum / k;
16. }
17. };

**2) Sort Colors:**

class Solution {

public:

void sortColors(vector<int>& nums) {

int n = nums.size() - 1;

int temp;

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

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

if(nums[j] > nums[j+1]){

swap(nums[j], nums[j+1]);

}

}

}

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

}

};

**3) Find Duplicate number:**

#include<bits/stdc++.h>

using namespace std;

int findduplicate(int nums[], int n){

        for(int i=0; i<n; i++){

           int val = abs(nums[i]);

            if(nums[val-1] < 0) return val;

            nums[val-1] = nums[val-1] \* -1;

        }

        return -1;

}

int main(){

    int nums[] = {1,3,4,5,4};

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

    cout << findduplicate(nums, n);

}

4) [**560. Subarray Sum Equals K**](https://leetcode.com/problems/subarray-sum-equals-k/)

class Solution {

public:

    int subarraySum(vector<int>& nums, int k) {

        unordered\_map<int,int> PrefixsumCount;

        PrefixsumCount[0] = 1;

        int currentsum = 0, count = 0;

        for(int num : nums){

            currentsum += num;

            if(PrefixsumCount.find(currentsum - k) != PrefixsumCount.end()){

                count += PrefixsumCount[currentsum - k];

            }

            PrefixsumCount[currentsum]++;

        }

        return count;

    }

};

5) [**14. Longest Common Prefix**](https://leetcode.com/problems/longest-common-prefix/) ---- (String)

class Solution {

public:

    string longestCommonPrefix(vector<string>& strs) {

        if(strs.empty()) return "";

        string prefix = strs[0];

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

            while(strs[i].find(prefix) != 0){

                prefix = prefix.substr(0, prefix.length()-1);

            }

        }

        return prefix;

    }

};

[**28. Find the Index of the First Occurrence in a String**](https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/)

class Solution {

public:

    int strStr(string haystack, string needle) {

        int haylen = haystack.length();

        int neelen = needle.length();

        if(haylen < neelen) return -1;

        for(int i=0; i<haylen; i++){

            int j = 0;

            while(j < neelen && haystack[i+j] == needle[j]){

                j++;

                if(j == neelen){

                    return i;

                }

            }

        }

        return -1;

    }

};

[**1249. Minimum Remove to Make Valid Parentheses**](https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/) ---- (Stack)

class Solution {

public:

    string minRemoveToMakeValid(string s) {

        stack<int> st;

        unordered\_set<int> remove;

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

            if(s[i] == '('){

                st.push(i);

            }

            else if(s[i] == ')'){

                if(!st.empty()){

                    st.pop();

                }

                else{

                    remove.insert(i);

                }

            }

        }

        while(!st.empty()){

            remove.insert(st.top());

            st.pop();

        }

        string result = "";

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

            if(remove.count(i) == 0){

                result += s[i];

            }

        }

        return result;

    }

};