Coding practices solution -----------

**75. Sort Colors**

class Solution {

public:

void sortColors(vector<int>& nums) {

int lo=0;

int hi =nums.size()-1;

int mid=0;

while(mid<=hi)

{

switch(nums[mid])

{

case 0:

swap(nums[lo++],nums[mid++]);

break;

case 1:

mid++;

break;

case 2:

swap(nums[mid],nums[hi--]);

break;

}

}

}

};

26. Remove Duplicates from Sorted Array

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

Input: nums = [1,1,2]

Output: 2, nums = [1,2,\_]

Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

class Solution {

public:

int removeDuplicates(vector<int>& nums) {

int n=nums.size();

int k=1;

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

for(int i=0;i<=n-1;i++)

{

if(nums[i]!=nums[k-1]){

nums[k++]=nums[i];

}

}

return k;

}

};

duplicate number; sheets=1

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

i/p=[1,2,3,1]

o/p= true

Input: nums = [1,2,3,4]

Output: false

class Solution {

public:

bool containsDuplicate(vector<int>& nums) {

int n=nums.size();

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

for(int i=0;i<n-1;i++)

{

if(nums[i]==nums[i+1])

return true;

}

return false;

}

};

timecomlexity o(nlogn)

**121. Best Time to Buy and Sell Stock**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 5

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

class Solution {

public:

int maxProfit(vector<int>& prices) {

int maxPro=0;

int minPrice= INT\_MAX;

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

{

minPrice=min(minPrice,prices[i]);

maxPro= max(maxPro, prices[i] - minPrice);

}

return maxPro;

}

};

**122. Best Time to Buy and Sell Stock II**

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return the ***maximum*** profit you can achieve.

**Input:** prices = [7,1,5,3,6,4]

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

class Solution {

public:

int maxProfit(vector<int>& prices) {

int profit=0;

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

{

if(prices[i]>=prices[i-1])

{

profit+=prices[i]-prices[i-1];// adding profit in profit

}

}

return profit;

// int prof = 0;

// for(int i=1;i<prices.size();i++)

// {

// if(prices[i]>=prices[i-1])

// prof+=prices[i]-prices[i-1];

// }

**// return prof;**

**}**

**};**

1

**509. Fibonacci Number**

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

**Explanation:** F(2) = F(1) + F(0) = 1 + 0 = 1.

F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

**Input:** n = 2

**Output:**

class Solution {

public:

int dp[32];

int fib(int n) {

if(n<1) return 0;

if(n==1) return 1;

if(dp[n]>0) return dp[n];

dp[n]=fib(n-2)+fib(n-1);

return dp[n];

}

};