***Week – 1 (28.03.2021 – 04.04.2021)***

***Topics Completed:***

1. Time and space complexity:

* Big/small O
* Big/small Omega
* Big/small theta
* Master Theorem
* Euclidean algorithm

2- STL:

* vector
* map
* unordered\_map
* set
* unordered\_set
* stack
* queue
* algorithm
* priority\_queue

3- Sorting:

* Bubble sort
* Insertion sort
* Selection sort
* Merge sort
* Heap sort
* Quick sort
* Counting sort
* Time Complexity
* Space Complexity

4. Hashing

***15 Solved Random Problems:***

|  |  |  |
| --- | --- | --- |
| S.No. | Problem Name | Problem Number in Leetcode |
| 1. | Missing Number | 268 |
| 2. | First Missing Positive | 41 |
| 3. | Sort Color | 75 |
| 4. | Implement strStr() | 28 |
| 5. | Robot Return to Origin | 657 |
| 6. | Sqrt(x) | 69 |
| 7. | To Lower Case | 709 |
| 8. | Power of Two | 231 |
| 9. | Power of Three | 326 |
| 10. | Excel Sheet Column Number | 171 |
| 11. | Valid Perfect Square | 367 |
| 12. | Plus One | 66 |
| 13. | Maximum Product of Three Numbers | 628 |
| 14. | Sort an Array | 912 |
| 15. | Squares of sorted Array | 977 |
| 16. | Sort Array by Parity | 905 |
| 17. | Single Element in Sorted Array | 540 |
| 18. | Kth Smallest Element in Sorted Array | 378 |

***LEETCODE CODES:***

1. ***Missing Number:***

class Solution {

public:

int missingNumber(vector<int>& nums) {

int i=0;

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

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

{

if(i != nums[i]) return i;

}

return nums.size();

}

};

1. ***First Missing Positive:***

class Solution {

public:

int firstMissingPositive(vector<int>& nums) {

int i, j=1;

bool c = false;

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

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

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

{

if(nums[i] < 0) continue;

if(nums[i] > 0)

{

if(nums[i] != j) return j;

j++;

}

}

return j;

}

};

1. ***Sort Color:***

class Solution {

public:

void sortColors(vector<int>& nums) {

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

}

};

1. ***Implement strStr():***

class Solution {

public:

int strStr(string haystack, string needle) {

if((haystack.size()==0 && needle.size() == 0) || (haystack.size()!=0 && needle.size()==0)) return 0;

if(haystack.size()==0 && needle.size()!=0) return -1;

if(haystack.size() < needle.size()) return -1;

if(haystack.size() == needle.size() && haystack == needle) return 0;

if(haystack.size() == needle.size() && haystack != needle) return -1;

int i,j=0;

string str;

for(i=0; i<=haystack.size()-needle.size()+1; i++)

{

int k = i;

j =0;

str = "";

while(j<needle.size() && haystack[i] == needle[j])

{

str = str + haystack[i];

i++;

j++;

}

if(str == needle) return k;

i = k;

}

return -1;

}

};

1. ***Robot Return to Origin:***

class Solution {

public:

bool judgeCircle(string moves) {

int i, x=0, y=0;

for(i=0; i<moves.size(); i++)

{

if(moves[i] == 'U') y = y+1;

if(moves[i] == 'D') y = y-1;

if(moves[i] == 'L') x = x-2;

if(moves[i] == 'R') x = x+2;

}

if(x == 0 && y == 0) return true;

else return false;

}

};

1. ***Sqrt(x):***

class Solution {

public:

int mySqrt(int x) {

return sqrt(x);

}

};

1. ***To Lower Case:***

#include <cctype>

class Solution {

public:

string toLowerCase(string str) {

transform(str.begin(), str.end(), str.begin(), ::tolower);

return(str);

}

};

1. ***Power of Two:***

class Solution {

public:

bool isPowerOfTwo(int n) {

int i;

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

{

if(pow(2,i) == n) return true;

if(pow(2,i) > n) return false;

}

return false;

}

};

1. ***Power of Three:***

class Solution {

public:

bool isPowerOfThree(int n) {

int i;

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

{

if(pow(3,i) == n) return true;

if(pow(3,i) > n) return false;

}

return false;

}

};

1. ***Excel Sheet Column Number:***

class Solution {

public:

int titleToNumber(string columnTitle) {

long int col = 0;

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

{

col = (col \* 26) + columnTitle[i] - 'A' + 1;

}

return col;

}

};

1. ***Valid Perfect Square:***

class Solution {

public:

bool isPerfectSquare(int num) {

if(num == 0 || num == 1) return true;

long int i=2;

while(i\*i <= num)

{

if(num == (i\*i)) return true;

i++;

}

return false;

}

};

1. ***Plus One:***

class Solution {

public:

vector<int> plusOne(vector<int>& digits) {

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

digits[0] = digits[0] + 1;

if(digits[0] > 9)

{

int i = 1;

int carry = digits[0]/10;

digits[0] = digits[0]%10;

while(i<digits.size())

{

int k = digits[i] + carry;

digits[i] = k % 10;

carry = k/10;

i++;

}

if(carry > 0) digits.push\_back(carry);

}

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

return digits;

}

};

1. ***Maximum Product of Three Numbers:***

class Solution {

public:

int maximumProduct(vector<int>& nums) {

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

return max(nums[0] \* nums[1] \* nums[nums.size() - 1], nums[nums.size() - 1] \* nums[nums.size() - 2] \* nums[nums.size() - 3]);

}

};

1. ***Sort an Array:***

class Solution {

public:

vector<int> sortArray(vector<int>& nums) {

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

return nums;

}

};

1. ***Squares of sorted Array:***

class Solution {

public:

vector<int> sortedSquares(vector<int>& nums) {

int i;

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

nums[i] = nums[i] \* nums[i];

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

return nums;

}

};

1. ***Sort Array by Parity:***

class Solution {

public:

vector<int> sortArrayByParity(vector<int>& nums) {

vector<int> even, odd, res;

int i, j=0,k=0;

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

{

if(nums[i]%2 == 0) even.push\_back(nums[i]);

else odd.push\_back(nums[i]);

}

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

sort(odd.begin(), odd.end(), greater<int>());

i=0;

while(i<even.size())

{

res.push\_back(even[i]);

i++;

}

i=0;

while(i<odd.size())

{

res.push\_back(odd[i]);

i++;

}

return res;

}

};

1. ***Single Element in Sorted Array:***

class Solution {

public:

int singleNonDuplicate(vector<int>& nums) {

if(nums.size() == 1) return nums[0];

int i;

for(i=0; i<nums.size(); i=i+2)

{

if(nums[i] != nums[i+1]) return nums[i];

}

return -1;

}

};

1. ***Kth Smallest Element in Sorted Array:***

class Solution {

public:

int kthSmallest(vector<vector<int>>& matrix, int k) {

vector<int> n;

int i, j;

for(i=0; i<matrix.size(); i++)

for(j=0; j<matrix[i].size(); j++)

n.push\_back(matrix[i][j]);

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

return n[k-1];

}

};